Internet Draft
Expiration: June 2000
File: draft-ietf-rap-pr-01.txt

Francis Reichmeyer
Shai Herzog
IPHighway
Kwok Ho Chan
John Seligson
Nortel Networks
David Durham
Raj Yavatkar
Intel
Silvano Gai
Keith McCloghrie
Cisco Systems
Andrew Smith
Extreme Networks

COPS Usage for Policy Provisioning

October 22, 1999

Status of this Memo

Copyright Notice

Abstract

This document introduces a new client type for the COPS protocol to support policy provisioning. Use of this new client type is independent of the type of policy being managed and it assumes a data model that is based on the concept of named policy information as found in a Policy Information Base, or PIB.

Table of Contents

Glossary

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

1. Introduction

The IETF RSVP Admission Policy (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 (PEP). 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 Provisioning.

The Outsourcing model addresses the kind of events at the PEP that require instantaneous policy decision (authorization). The PEP, being aware that it must perform a policy decision. However, being unable to carry the task itself, the PEP delegates responsibility to an external policy server (PDP). For example, in [COPS-RSVP] when a reservation message arrives, the PEP is aware that it must decide whether to admit or reject the request. It sends a specific query to the PDP, and in most case, waits for a decision before admitting the outstanding reservation.

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 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.

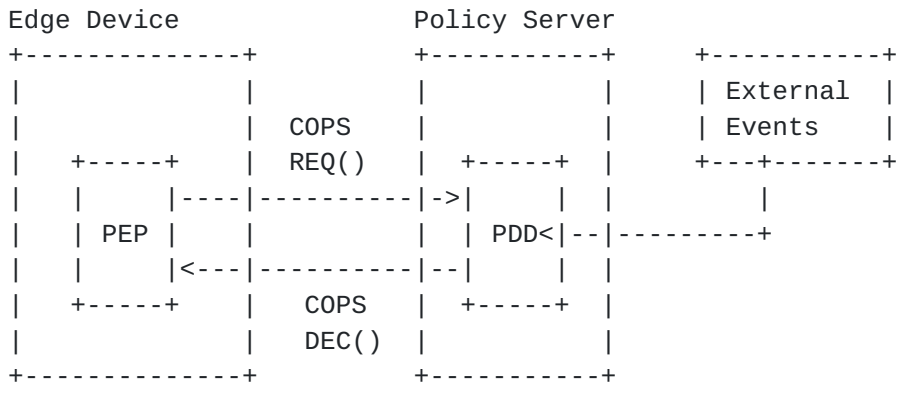


Figure 1: COPS Provisioning Model

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 cannot be mapped directly to requests, and are issued mostly when the PDP responds to external events or PDP events (policy/SLA updates).

This draft describes a new client type ("Provisioning") for COPS to support policy provisioning. This new client type is independent of the type of policy (QoS, VPNs, Security, etc.) and it is based on the concept of PIBs (Policy Information Bases [PIB]).

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

1.1. Why not SNMP?

SNMP is a very popular network management protocol. One may question using COPS-PR, rather than extending SNMP for policy provisioning.

There are several aspects intrinsic to SNMP that prevents it from being a successful policy protocol.

SNMP uses a transactional model, and does not support the concept of long term Client/Server connection. As a by product, servers may not know that devices failed and vice versa. A hello polling may be a cumbersome replacement, however it may not solve the problem if a device may reboot in between polling messages.

The SNMP transactional model allows multiple servers to simultaneously modify state of a network device. Given that SNMP does not have resource locking facilities, a policy server would

have to constantly poll and verify that no other networking management software or humans changed ANY of the configured resources.

SNMP is based on UDP and is thus unreliable. The lack of reliability is unacceptable for a policy protocol [RAP]. Provisioning policy is assumed quite large and diverse. It is desired that a provisioning protocol would be based on state sharing between client and server such that only differential updates are sent. Such state sharing requires a reliable transport mechanism.

Last, SNMP was not designed as a real-time operations protocol. Its trap mechanism is inefficient and cumbersome and there is no performance guarantees.

COPS was designed to overcome these shortcomings, based on the requirements defined in [RAP]. It has a single connection between client and server, it guarantees only one server updates the

policy configuration at any given time (and these are locked, even from console configuration, while COPS is connected to a server). 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 high priority (real-time) mechanism for the PEP device.

The COPS protocol is already used for policy control over RSVP. It is highly desirable to use a single policy control protocol for Quality of Service (QoS) mechanisms (if possible), rather than invent a new one for each type of policy problem.

At the same time, useful mechanisms from SNMP were adopted. COPS-PR uses a named Policy Information Base (PIB) which the model of SMI and MIB and BER [BER] data encoding. This allows reuse of experience, knowledge, tools and some code from the SNMP world.

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 which 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 and/or deletes) in policy to the PEP, and the PEP updates its local QoS mechanisms 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 sends this unsolicited new information to the PDP. On receiving this new information, the PDP sends to the PEP any additional provisioned policies now needed by the PEP.

2. Policy Information Base (PIB)

The data carried by COPS-PR is a set of policy rules. The protocol uses 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. The PIB name space is common to both the PEP and the PDP and names within this space are unique within the scope of a given PDP/PEP/ClientType communication channel. Note that a give device might implement multiple PEPs or multiple ClientTypes and the name space then only has uniqueness within each separate channel.

The PIB can be described as a conceptual tree data structure where the branches of the tree represent types of rules or Policy Rule Classes (PRCs), while the leaves represent the contents of Policy Rule Instances (PRIs). There may be multiple instances of rules (PRIs) for any given rule type (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:

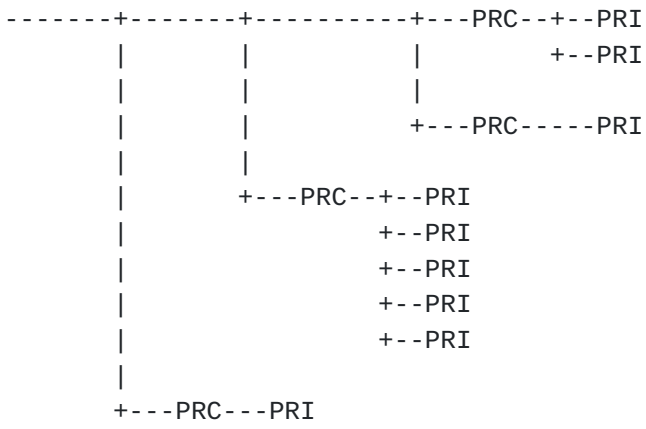


Figure 2: The PIB Tree

Instances of the policy rules (PRIs) are each identified by a Policy Rule Identifier (PRID). A PRID is a name, carried in a COPS <Named ClientSI> object, which identifies a particular instance of a rule.

2.1. PIB Syntax

The provisioning PIB syntax is based on SMI and MIBs, based on the ASN.1 data definition language [ASN1]. The decision to use this format as a basis opens-up the possibility of leveraging SNMP SMI and MIB knowledge, experience and tools. In order to simplify the implementation and allow re-use of SNMP encoding/decoding code, the wire representation of the policy information (PRIDs and BPDs) in the COPS protocol objects follows the Basic Encoding Rules (BER) [BER] - the object syntax definitions appear in section 4.

PRCs and their PRIs are identified by PRIDs, which are unique within the scope of a given PDP/PEP/ClientType channel. PRIDs have a hierarchical structure of the form a.b.c.d (e.g. 1.3.4.7), where a prefix identifies the PRC (e.g., 1.3 or 1.3.4) and the last component identifies the individual instance (e.g. 7).

Note that the instance values do not have to be consecutive although they must be unique to this PDP/PEP/ClientType communication. The actual values for the indices may be chosen by the PDP and they may or may not have significance to the PDP as real values; they have no significance to the PEP other than as instance identifiers. Note also the intentional similarity to SNMP's SMI syntax and semantics [V2SMI]. There is no need for a "context" mechanism, such as that in SNMP, to disambiguate different PRIs containing the same data: the instance numbers are chosen by the PDP and the semantics of contexts can, therefore, be encoded in the PRC definitions themselves.

Given that most provisioning operations require multiple attributes, COPS-PR does not support operations on individual attributes within a PRC (e.g. filterSrcPort above). Updates and deletions are performed on a granularity of per-PRC only.

The policy tree names all the policy rule classes and instances and this creates a common view of the policy organization between the client (PEP) and the server (PDP). The PIB data on its own is self- descriptive such that the receiving PEP understands the required provisioning.

2.2. PIB Example

Consider the following simple example of a set of policy rule class to represent filters for marking IP traffic with a certain

diff-serv code point (DSCP). Each filter has the following attributes: Protocol number, source address, source port, destination address, destination port, and DSCP value to set. This might be represented by the following class definition:

```
    filterTable OBJECT-TYPE
        SYNTAX         SEQUENCE OF FilterEntry
        POLICY-ACCESS  install
        STATUS         current
        DESCRIPTION
            "Filter PRC."
    ::= { pib 1 }

    filterEntry OBJECT-TYPE
        SYNTAX         FilterEntry
        STATUS         current
        DESCRIPTION
            "An instance of the filter class."
        INDEX { filterIndex }
    ::= { filterTable 1 }

    FilterEntry ::= SEQUENCE {
            filterIndex        INTEGER, -- arbitrary index
            filterProtocol     INTEGER,

            filterSrcAddr      IpAddress,
            filterSrcPort      INTEGER,
            filterDstAddr      IpAddress,
            filterDstL4Port    INTEGER,
            filterDscp         Integer32
    }
    etc.
```

Let us assume that the base "pib" has a prefix in the policy tree of 1.2.3. So, the first filter instance might have a PRID of pib.filterTable.filterEntry.10, or 1.2.3.1.1.10. The next filter instance might then get the PRID 1.2.3.1.1.99. This PIB segment might be shown diagramatically as:

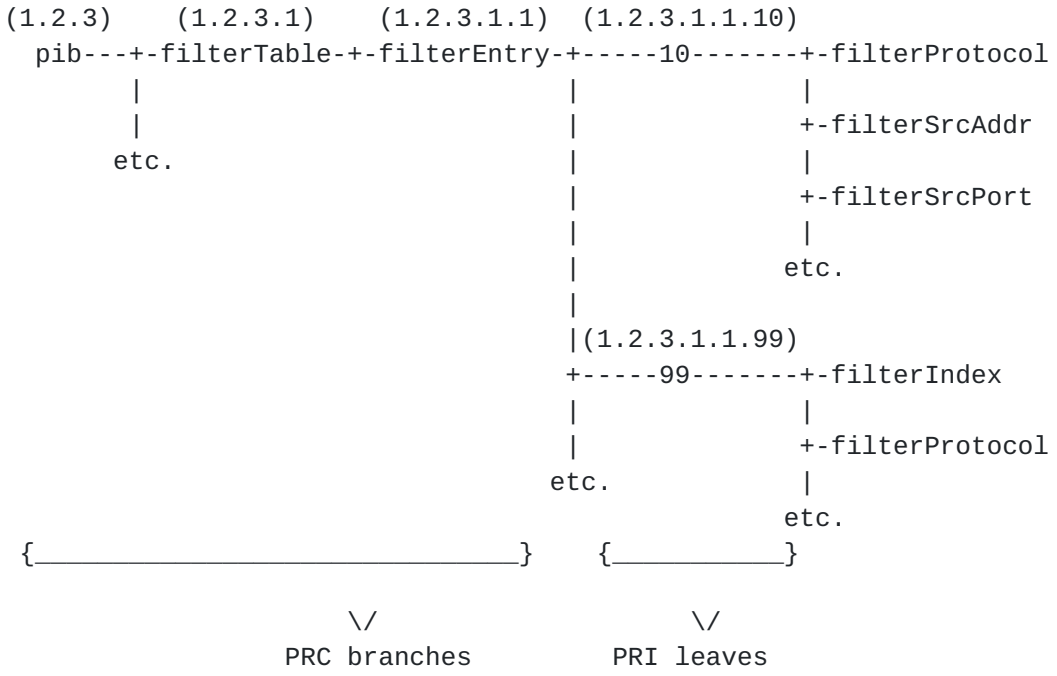


Figure 3: A PIB Example for a DiffServ Filter

The numbers in parentheses represent the location of the PRC or PRI in the tree. Note that the last digit of the PRCs (which in SMI would describe the individual class attributes) is dropped from the PRID since COPS-PR only supports operations on complete classes, not on individual attributes.

2.3. Rules for Modifying and Extending PIBs

As experience is gained with policy 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 existing one deprecated.

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

(3) An existing PRC can be extended by "augmenting" it 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.

2.3.1. 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 OIDs under the module OID.

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. The PEP SHOULD ignore these unsupported PRI. However, it MAY return and error to the PDP. In the latter case, the PDP must restructure its policy decisions to exclude the unsupported PRCs.

Similarly, existing PRCs can be deprecated from a PIB. In this case, the PEP ignores any PRIs sent 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.

2.3.2. Adding or Deprecating Attributes of a 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, 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. It MAY 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 and error back to the PDP. In all cases the missing attributes are assumed to correspond to the last attributes of the PRC.

2.3.3. Augmenting a PRC with another PRC

Rather than extending a PRC by modifying the PIB and adding attributes to that PRC, a new PRC can be defined, perhaps in a different PIB module to augment an existing PRC. This is especially useful for independent enterprises to independently augment an existing class.

An augmenting PRC has its own OID. However, an instance of this PRC can only be created if there is a corresponding instance (with the same instance ID) of the base PRC. The base PRC, on the other hand, can be configured by a PDP without the PDP also configuring the augmenting PRC (or PRCs). In this case, the PEP MUST asume some default values for the attributes of the augmenting PRC.

When the PDP deletes an instance of a base PRC, the instances of the corresponding augmented PRCs are also deleted.

Augmenting standard PIB attributes with enterprise specific extensions introduces interoperability issues regarding policy servers that are unaware of the proprietary additions. Under this scenario, the DEFVAL clause SHOULD be used to provide default values for the proprietary attributes. All attribute definitions in a class the augments a base class SHOULD include a DEFVAL clause specifying a reasonable default value. This helps to ensure that a PDP may adequately provision a PEP based solely on standard PIB attributes.

Rules governing the usage and specification of the DEFVAL clause are defined in the SMIv2 [SNMP-SMI].

2.4. COPS Operations Supported for a Policy Rule Instance

A Policy Rule Instance (PRI) may contain multiple leaf attributes and is identified uniquely, within the scope of a given COPS ClientType on a PEP, by a Policy Rule 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 a BER-encoded Policy Instance Data (BPD) object with the new/updated values. The PRID value MUST uniquely identify a single PRI (i.e. PRID/PRC prefix values are illegal).

- 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.

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.

3.1. Request (REQ) PEP -> PDP

The REQ message is sent by policy provisioning clients to issue a 'config request' to the PDP. The Client Handle associated with the REQ message originated by a provisioning client must be unique for that client but otherwise has no protocol significance at this time.

The config request message serves as a request from the PEP to the PDP for provisioning policy data which 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.

The config 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 resource (e.g. queuing capabilities) and default policy configuration (e.g. default role combinations) information as well as existing policy (i.e. PIB) incarnation data. This information typically does not include state previously installed by a PDP. This information from the client assists the server in deciding what types of policy the PEP can install and enforce. The format of the Provisioning ClientSI data is described in the policy information base (see section 5).

Note that the config request message is regenerated and sent to the PDP in response to the receipt of a Synchronize State Request (SSQ) message.

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 config 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: Provisioning >]
              [<Integrity>]
```


Note that the COPS objects IN-Int, OUT-Int and LDPDecisions 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 REQ message.

The DEC message is sent as an immediate response to a config request with the solicited decision flag set. Subsequent DEC messages may also be sent at any time after the original DEC message to supply the PEP with additional/updated policy information. Updated policy data carried in DEC message is correlated with the previous DEC by matching the policy ID information in the provisioning client decision data.

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

A COPS-PR DEC message contains a single "transaction", i.e. either all the decisions in a DEC message succeed or they all fail. 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(s)>]+ | <Error>
                       [<Integrity>]

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

Note that only Named Decision Data (Provisioning) is included in a COPS-PR Decision. Other types of COPS decision data (e.g. Stateless, Replacement) are not supported.

For each decision on the DEC message, the PEP performs the operation specified in the Flags field 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 sends a RPT message 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 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 = 'Installed') or not (Report-Type = 'Not Installed').

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:

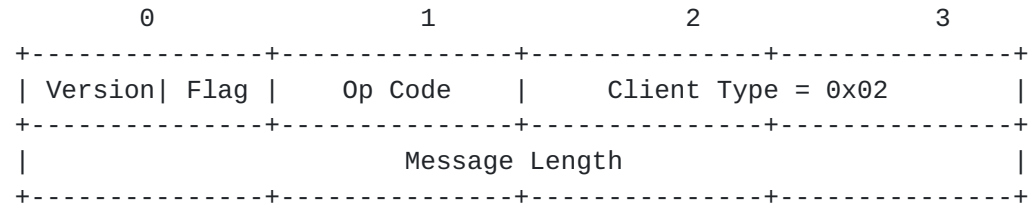
```
<Report State> ::= <Common Header>
                   <Client Handle>
                   <Report Type>
                   [<Named ClientSI: Provisioning >]
                   [<Integrity>]
```

4. COPS-PR Protocol Objects

We define a new COPS client type for the policy provisioning client:

Client Type = 2; Policy Provisioning Client

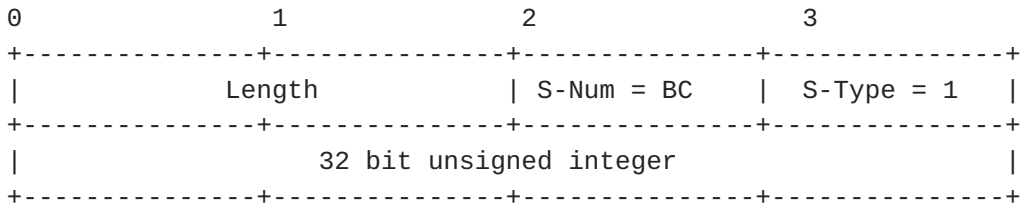
COPS messages sent between a Policy Provisioning client and a COPS server contain a COPS Common Header with this Policy Provisioning Client type specified:



The COPS Policy Provisioning client uses several new COPS protocol objects that carry named client-specific information. This section defines those new objects.

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

The format for these new objects is as follows:



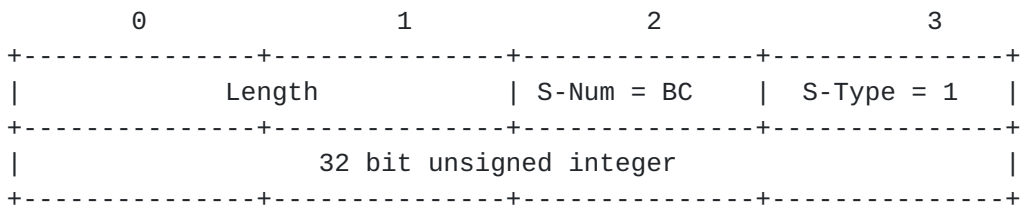
S-Num and S-Type are similar to the C-Num and C-Type used in the base COPS objects. The difference is that S-Num and S-Type are used only for ClientSI specific objects.

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.

4.1. Binding Count (BC)

S-Num = 1, S-Type = 1, Length = 8.

This object specifies the number of Bindings that are contained in the message.



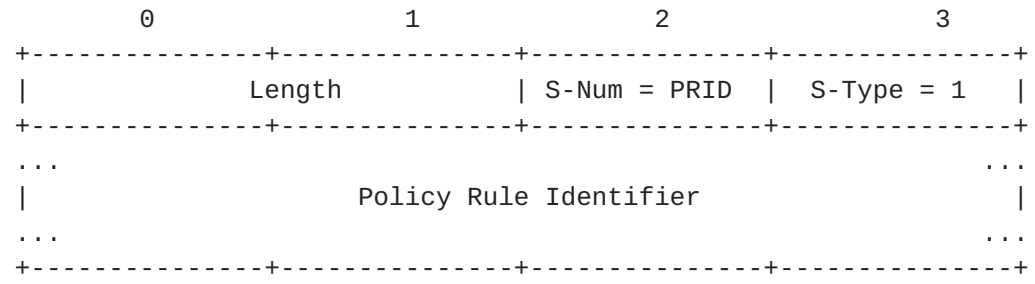
4.2. Policy Rule Identifier (PRID)

4.2.1. Complete PRID

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

This object is used to carry the identifier, or PRID, of a Policy Rule 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
02                          - S-Num
01                          - S-Type (Complete PRID)
06 07 2B 06 01 02 02 08 01  - Encoded PRID
00 00 00                    - Padding
```

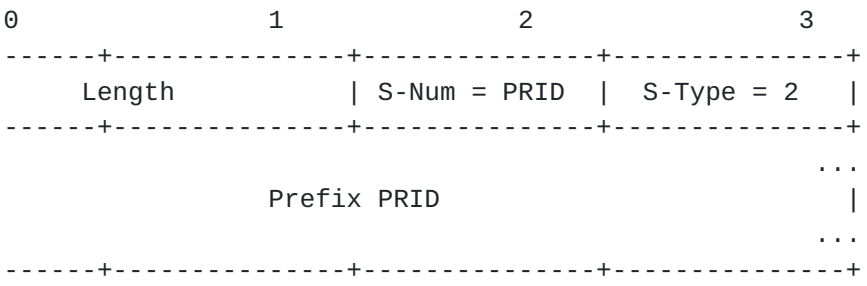
4.2.2. Prefix PRID

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. 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.

The specification of a prefix is performed using the Policy Rule Identifier object with an S-Type equal to 2 (Prefix PRID).

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



Continuing with the previous example, a PRC prefix 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

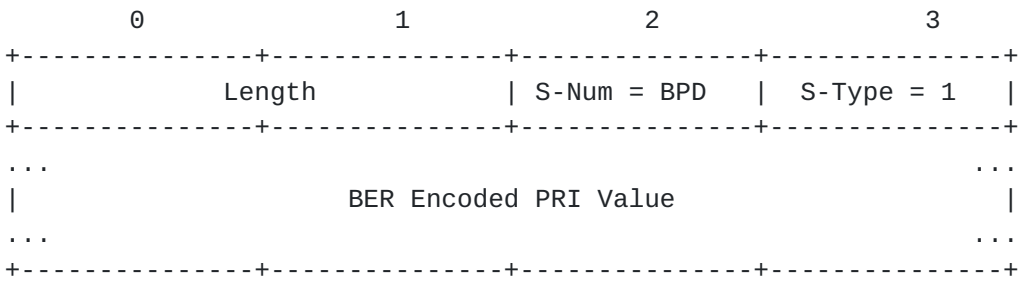
The entire PRID object would be encoded as follows:

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

4.3. BER Encoded Policy Instance Data (BPD)

S-Num = 3, S-Type = 1, Length = variable.

This object is used to carry the BER encoded value of a Policy Data Instance. This object is used to carry the BER encoded value of a Policy Rule Instance. The PRI value, which contains all of the individual values of the attributes that comprise the class, 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.



As an example, an instance of the qosIpAce class, defined in the QoS Policy IP PIB [PIB], would be encoded as follows:


```
02 01 08            :qosIpAceIndex/INTEGER/Value = 8
40 04 C0 39 01 05   :qosIpAceDstAddr/IpAddress/Value =
                                          192.57.1.5
40 04 FF FF FF FF   :qosIpAceDstMask/IpAddress/Value =
                                          255.255.255.255
40 04 00 00 00 00   :qosIpAceSrcAddr/IpAddress/Value = 0.0.0.0
40 04 00 00 00 00   :qosIpAceSrcMask/IpAddress/Value = 0.0.0.0
02 01 FF            :qosIpAceDscp/Integer32/Value = -1 (not used)
02 01 06            :qosIpAceProtocol/INTEGER/Value = 6 (TCP)
05 00               :qosIpAceDstL4PortMin/NULL/not supported
05 00               :qosIpAceDstL4PortMax/NULL/not supported
05 00               :qosIpAceSrcL4PortMin/NULL/not supported
05 00               :qosIpAceSrcL4PortMax/NULL/not supported
02 01 01            :qosIpAcePermit/TruthValue/Value = 1 (true)
```

The entire BPD object would be encoded as follows:

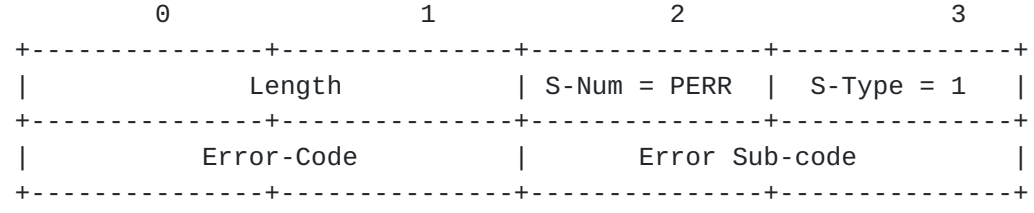
```
00 30                       - Length
```

```
03                          - S-Num = BPD
01                          - S-Type
02 01 08                    - qosIpAceIndex
40 04 C0 39 01 05           - qosIpAceDstAddr
40 04 FF FF FF FF           - qosIpAceDstMask
40 04 00 00 00 00           - qosIpAceSrcAddr
40 04 00 00 00 00           - qosIpAceSrcMask
02 01 FF                    - qosIpAceDscp
02 01 06                    - qosIpAceProtocol
05 00                       - qosIpAceDstL4PortMin
05 00                       - qosIpAceDstL4PortMax
05 00                       - qosIpAceSrcL4PortMin
05 00                       - qosIpAceSrcL4PortMax
02 01 01                    - qosIpAcePermit
```

Note that attributes not supported within a class are still returned in the BPD 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. Provisioning Error Object (PERR)

S-Num = 4, S-Type = 1, Length = 8.



The 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 policy provisioning client also adds the following error code:

Error Code 14 = Provisioning Error

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 named decision data, request data and report data. The actual content of the data is defined by the policy information base for the provisioning client type (see below).

5.1. Named Decision Data

The Named Decision Data for the policy provisioning client consists of two types of decisions: Install and Remove, used with the 'Install' and 'Remove' Command-Code, respectively, in the COPS Decision Object. The data, in general, is composed of one or more bindings. Each binding associates a PRID object and a BPD object. The PRID object is always present in both install and remove decisions, the BPD 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 the provisioning client named decision data is as follows:

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

<Install Decision>    ::= <BC> <PRID> <BPD> [<PRID> <BPD>]+


<Remove Decision>     ::= <BC> <PRID> [<PRID>]+
```


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 as follows:

<ClientSI: Named Request> ::= <BC> <PRID> <BPD> [<PRID> <BPD>]+

5.3. Policy Provisioning Report Data

The provisioning client report data is used in the RPT message in conjunction with the accompanying COPS Report Type object. Report types can be 'Commit' or 'No-Commit' 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 policy rule. In a 'Commit' RPT message, 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). In a 'No Commit' RPT message, this is an error code specific to a binding.

Global errors are not tied to a specific PRID. In a 'Commit' RPT message, a global error is an indication of a general warning at the PEP level (e.g., memory low). In a 'No Commit' RPT message, this is an indication of a general error at the PEP level (e.g., memory exhausted).

In the case of a 'No Commit' the PEP MUST report at least the first error and should report as many errors as possible.

<ClientSI: Named Report> ::= [<global-error>] [report]+

<global-error> ::= <Error>

<report> ::= <PRID> <specific-error>
[<BC>[<PRID><BPD>[<PRID><BPD>]+]]

<specific-error> ::= <Error>

6. Common Operations

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 with the Client-Type = 2, Policy Provisioning client. If the PDP supports the 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.

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 with and, optionally, any relevant client specific information for the PEP. The information provided by the PEP should include client resource (e.g., supported classes/attributes) and default policy configuration information as well as existing policy (i.e., PIB) incarnation data. The config 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.

Also, the config request is a request to asynchronously send policy data to the PEP, as the PDP decides is necessary. This asynchronous data may be new policy data or an update to policy data sent previously.

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 Decision object. If no filters are defined, the DEC message will simply specify that there are no filters using the "NULL Decision" Decision Flags object. The PEP MUST specify a client handle in the request message. The PDP MUST process the client handle and copy it in the decision message. 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 existing state by sending subsequent DEC message(s) to the PEP, with the same Client Handle. The PEP is responsible for removing the Client handle when it is no longer needed, for example when the interface goes down, and informing the PDP that the handle is to be deleted.

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) must 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, must be communicated to the PEP by the PDP. This information is sent to the PEP using the Decision message containing Policy Provisioning client specific 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 DEC messages containing the same Client Handle and the updated Policy Provisioning request state. Updates can specify that policy data is to be deleted or installed.

The PEP acknowledges the DEC message and action taken by sending a RPT message with a "Commit" or "No-Commit" Report-Type object. This serves as an indication to the PDP that the requestor (e.g. H.323 server) can be notified that the request has been accepted by the network. If the PEP needs to reject the DEC operation for any reason, a RPT message is sent with a Report-Type of value "No-Commit" and optionally a Client Specific Information object specifying the policy data that was rejected. The PDP can then respond to the requestor accordingly.

The PEP can report to the PDP the local status of any installed request state when appropriate. This information is sent in a Report-State (RPT) message with the "Accounting" flag set. The state being reported on is referenced by the Client Handle associated with the request state and the client specific data identifier.

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 at this time to be manually configured 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 included the last PDP address information. Based on this information, the PDP may

request the PEP to re-synch its current state information (SSQ message). If, after re-connecting, the PDP does not request the synchronization, the client can assume the server recognizes it and the current state at the PEP is correct. Any state changes which occurred at the PEP while the connection was lost must be reported to the PDP in a RPT message. If re-synchronization is requested, the PEP MUST reissue any REQ messages it generated during initial connection establishment 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 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 (TTL: Time To Live, see Section 3.3), the request state is to be deleted and the associated Handles removed. The same holds true for the PDP; upon detecting a failed TCP connection, the time-out timer is started for the request state associated with the PEP and the state is removed after the specified period without a connection.

7.1. Security Considerations

The use of COPS for Policy Provisioning introduces no new security issues over the base COPS protocol [COPS]. The security mechanism described in that document should be deployed in a COPS-PR environment.

8. References

[COPS] Boyle, J., Cohen, R., Durham, D., Herzog, S., Raja, R., Sastry, A., "The COPS (Common Open Policy Service) Protocol", IETF <draft-ietf-rap-cops-07.txt>, August 1999.

[RAP] Yavatkar, R., et al., "A Framework for Policy Based Admission Control",IETF <draft-ietf-rap-framework-03.txt>, April 1999.

[E2E] Bernet, Y., Yavatkar R., Ford, P., Baker, F., Nichols, K., Speer, M., "A Framework for End-to-End QoS Combining RSVP/Intserv and Differentiated Services", IETF <draft-ietf-DiffServ-rsvp-01.txt>, November 1998.

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

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

[PIB] M. Fine, K. McCloghrie, S. Hahn, K. Chan, A. Smith, "An Initial Quality of Service Policy Information Base for COPS-PR Clients and Servers", draft-mfine-cops-pib-02.txt, October 1999.

[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.

9. Author Information

Francis Reichmeyer
Phone: (201) 585-0800
Email: FranR@iphighway.com

IPHighway Inc.
Parker Plaza, 16th Floor
400 Kelby St.
Fort-Lee, NJ 07024

Shai Herzog
Phone: (201) 585-0800
Email: Herzog@iphighway.com

Kwok Ho Chan
Phone: (978) 916-8175
Email: kchan@nortelnetworks.com

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

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

Intel
2111 NE 25th Avenue
Hillsboro, OR 97124

Raj Yavatkar
Phone: (503) 264-9077
Email: raj.yavatkar@intel.com

Silvano Gai
Phone: (408) 527-2690
Email: sgai@cisco.com

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

Keith McCloghrie

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

Andrew Smith
Phone: +1 408 579 2821
Email: andrew@extremenetworks.com

Extreme Networks
3585 Monroe St.
Santa Clara CA 95051
USA

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

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

10. Full Copyright Notice