Internet Draft
Expiration: August 2000
File: draft-mameli-issll-cops-api-00.txt

Roberto Mameli
CoRiTeL

The CCAPI (COPS Client Application Programming Interface)

February 22, 2000

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document focuses on the Admission Control functionality performed

by the Edge Router in the IntServ/DiffServ interworking scenario described in [COPS-ISDS]. More precisely it describes the interaction between the RSVP and the COPS protocols in the Edge Router and

introduces an API (Application Programming Interface) aimed at allowing the intercommunication between them. Anyway, the API described here is designed as a flexible interface, and should be able to support communication between a generic application and the COPS Client Type described in [COPS-ODRA].

Table of Contents

Glossary

API	Application Programming Interface
BB	Bandwidth Broker
CCAPI	COPS Client Application Programming Interface
COPS	Common Open Policy Service
DSCP	Differentiated Services Code Point
ER	Edge Router
IPC	Inter Process Communication
PDP	Policy Decision Point.
PEP	Policy Enforcement Point.
QoS	Quality of Service
RSVP	ReSerVation Protocol
SLS	Service Level Specification

1. Introduction

One of the possible scenarios for end-to-end QoS provisioning relies on a proper combination of both the Integrated Services (IntServ) ([INTSERV] and [RFC2210]) and the Differentiated Services (DiffServ) ([2BIT] and [DSARCH]) architectures. The description of such a model is beyond the scope of this document; a detailed explanation can be found in [INTDIF] and in [COPS-ISDS]. However it is worth observing that the router placed at the boundary between the IntServ stub domain and the

DiffServ core, i.e. the Edge Router (ER), provides several interworking functionality (described in details in the references above).

One of the most important functionality managed by the ER is related to admission control. The original DiffServ architecture provides a sort of implicit admission control, in the form of SLS negotiated between neighboring domains. The introduction of end-to-end signaling by means of the RSVP protocol allows explicit admission control on micro-flow basis, as explained in details in [INTDIF].

Note, however, that the are at least two ways to perform Admission Control in the Edge Router. In fact, it can be realized either in a distributed way by means of information locally available, or in a centralized way by querying an Admission Control Server. As stated in [COPS-ISDS], the first approach is easier to implement, but it is nevertheless characterized by inaccuracy, since each ER does not have an overall knowledge of the network resource utilization. Moreover, in some situations (failures, etc.), consistency among information stored in different ERs could not be assured. For this reason [COPS-ISDS] focuses on the second solution. However, in such a scenario the communication between the ER and the centralized server must happen according to a proper protocol. COPS represents a possible choice, since it is a simple and extensible protocol; [COPS-ODRA] proposes an extension suited at the purpose.

Besides the COPS extension, needed for the communication between the centralized server and the Edge Router, another communication interface should be defined. In fact, the Edge Router supports both the COPS-ODRA PEP and the RSVP daemon. As explained in [COPS-ISDS], the entire mechanism requires proper interaction between them; for example the reception of a RSVP RESV message by the ingress ER should trigger an admission control query towards the server (PDP/BB). This observation leads to the definition of a new interface between the COPS-ODRA client and the RSVP daemon within the Edge Router.

This document describes an API that can be used at the purpose. This

API, called CCAPI (COPS Client API), is based on a client library statically linked with the RSVP daemon. The latter can trigger queries to the PDP/BB server when needed and can receive responses from it either synchronously or asynchronously, as explained in the following paragraph. Note, however, that the CCAPI is designed in a flexible way, in order to be usable with applications other than RSVP, that could need to interact with the COPS-ODRA PEP for whatever reason. For this reason in the remaining part of the document we will use the term _CCAPI client_ to refer to such an application, thus avoiding to mention explicitly RSVP, even if it obviously represents the natural choice.

2. CCAPI generalities

As stated above, the CCAPI is realized by means of a client library statically linked with the CCAPI client. The procedures implemented in the CCAPI library use a proper inter-process communication (IPC) mechanism to interact with the COPS client, which in turn relies on the COPS protocol to communicate with the server. The situation is depicted in the following figure:

```
  +----------------------------------------------+
  | Edge        +-----------+  +-----------+     |        +-----------+
  | Device      |  CCAPI    |  |COPS-ODRA |_______________|COPS-ODRA |
  |             |  Client   |  |  Client   |     | COPS   |  Server   |
  |             +-----------+  +-----------+     |        +-----------+
  | User Level       ||             ||           |
  |=================||=============||===========|
  | Kernel Level     |----------------| IPC       |
  |                  ----------------- Mechanism |
  +----------------------------------------------+
```

Note that there is no need to standardize the inter-process communication mechanism, since it could vary due to several reasons, such as Operating System characteristics. The CCAPI proposed here does not assume anything about it, even if the actual implementation relies on the Linux socket mechanism. The definition of the interface specifies only the visible part of it, i.e. the set of routines made available to the CCAPI client. They are listed and explained in the following.

The API can be used to manage events both in a synchronous and in an asynchronous way. In the first case the CCAPI client triggers a query to the PDP/BB and blocks indefinitely waiting for a response. This is the easiest way to use the CCAPI, but it could lead to undesirable behavior in the case of no response from the server. Let us consider as an example the case of an admission control request from RSVP; if the server does not respond for whatever reason, the daemon would wait indefinitely. This is obviously undesirable, since normal processing of other requests should not be blocked by a pending request (e.g. timeout of installed PATH and RESV states would expire, and so on).

For this reason the CCAPI has been designed to manage events asynchronously, by means of an _upcall_ or _callback_ mechanism. In this way the CCAPI client triggers the request from the PEP to the PDP/BB without waiting for response. When a response from the PDP/BB is available, the PEP notifies the CCAPI client (e.g. the RSVP daemon) that, in turn, manages it by calling a proper callback routine.

A synchronous error in a CCAPI library routine returns an appropriate error code. Asynchronous errors are delivered to the application via the CCAPI upcall routine. Text messages for synchronous and asynchronous error codes can be found in the file cops_err.h.

3. CCAPI Description

This paragraph reports a brief description of the sequence of operations needed by the CCAPI client and lists all the CCAPI calls along with their explanation.

3.1. CCAPI Outline

The CCAPI client must include cops_api.h and cops_err.h and must be linked with cops_api.c. It begins by opening the session to the COPS-ODRA PEP via the cops_api_open_session() call; when it issues this call it can optionally specify a pointer to an appropriate callback routine (if any). The session associates the CCAPI Client with the PEP, meaning that the latter cannot have more sessions opened at the same time. If this is the need, several PEP processes should be instantiated together on the same Edge Router, and each of them should have a single session opened towards the CCAPI client.

After the cops_api_open_session() call, the CCAPI client can ask for resource request, release and/or modify by means of the bandwidth_request(), bandwidth_release() and bandwidth_modify() calls, with proper parameters. In order to get a response the cops_api_dispatch() call can be used. In the _blocking_ case, the cops_api_dispatch() can be preceded by a select() system call, in order to wait indefinitely for an event; when such an event occurs, the select is unblocked and the cops_api_dispatch() can be used to obtain the response. No callback routine is needed in this case. In contrast, in the _non blocking_ case, a proper callback routine is specified in the cops_api_open_session(). The cops_api_dispatch() is periodically called inside the CCAPI client main loop, and it polls the PEP to see if a response has arrived; if so, the callback routine is executed. The latter receives also an optional argument that can be specified when opening the session through the cops_api_open_session() call.

Whatever the mechanism we choose, _blocking_ or _non blocking_, at the end of all the operations the session can be closed via the cops_api_release_session() call. In the following a brief description of all the CCAPI calls is reported.

3.2. CCAPI calls

3.2.1. cops_api_open_session()

The cops_api_open_session() call is used to open a session with the COPS-ODRA PEP. It returns a cops_api_error (i.e. an unsigned int), which could be any of the following values:

```
   COPS_API_OK     - session opened without problems
```



```
   COPS_API_NOCOPS _ COPS-ODRA PEP is not running on the ER
   COPS_API_OPEN   - session already opened
```

The definition of the function is the following:

```
   cops_api_error
   cops_api_open_session ( cops_event_rtn event_rtn,
                           void *event_rtn_arg)
```

The parameters are:

```
   event_rtn     - is a pointer to the callback function specified
                   by the CCAPI client. It could be NULL if the latter
                   is not interested in managing asynchronous events
                   by the upcall mechanism.
```

event_rtn_arg - is a pointer to an optional parameter that is passed to the callback routine whenever it is called.

3.2.2. cops_api_getfd()

It may be used by the CCAPI client to retrieve a file descriptor; this, in turn, could be used in a select() call immediately before the dispatch(), so as to realize a blocking mechanism. The function is specified as follows:

```
int
cops_api_getfd()
```

It doesn't require parameters and returns the file descriptor, or _1 if the session has not been opened before.

3.2.3. cops_api_release_session()

The cops_api_release_session() is used to close the session with the COPS-ODRA PEP. It returns a cops_api_error of the following types:

```
COPS_API_OK     - session closed without problems
COPS_API_CLOSE  - session already closed
COPS_API_SYSERR _ system error
```

The function prototype is:

```
cops_api_error
cops_api_release_session ()
```

No parameters are requested.

3.2.4. bandwidth_request()

The bandwidth_request() library call is used by the CCAPI client in order to instruct the PEP to query the PDP/BB for bandwidth request; referring to RSVP as an example, this could happen whenever a new flow is admitted at the ingress Edge Router. The function prototype is the following:

```
cops_api_error
bandwidth_request ( cops_req *request)
```

It takes in input a pointer to a cops_req structure, that contains information about the request. The definition of the cops_req structure is reported in paragraph 5. Note that it also contains a request identifier, that is used in the _non-blocking_ case to associate requests made by the CCAPI client to responses provided by the PEP. In fact, in such a case, the CCAPI client can issue several requests to the PEP, and a way to relate them to corresponding responses is obviously needed. The library call can return one of the following values:

```
COPS_API_OK     - request successfully delivered to the PEP
COPS_API_INVREQ _ invalid request; session not in place
COPS_API_INVRID _ invalid request identifier (already in use)
COPS_API_TOOMF  - excessive number of pending requests
COPS_API_SYSERR _ system error
```

3.2.5. bandwidth_release()

This is the complementary function to the previous one. The CCAPI client uses it in order to instruct the PEP to communicate bandwidth release to the PDP/BB. If the CCAPI client is represented by RSVP, bandwidth_release() is called when a reservation is released, e.g. upon the reception of a RESV TEAR message by the ingress Edge Router.

```
cops_api_error
bandwidth_release ( cops_req *request)
```

The parameters and the return values of this function are the same of the bandwidth_request().

3.2.6. bandwidth_modify()

The bandwidth_modify() call has been introduced with reference to the situation where the CCAPI client is represented by RSVP. It can be used to change dynamically a reservation without first releasing resources and then allocating them again. In this way there are two advantages: first of all the reservation is changed with a single message. Moreover, in the case of rejection of the new request, the old one remains in place. The function is defined as follows:

```
cops_api_error
bandwidth_modify ( cops_req *request)
```


The parameter is a pointer to a cops_req structure; differently from the case of bandwidth_request() and bandwidth_release() this request now contains a pair of bandwidth values, instead of a single one. The return values are the same of bandwidth_request().

3.2.7. cops_api_dispatch()

Applications use the cops_api_dispatch() library call to receive notifications of COPS events, e.g. responses to their queries. The function prototype is the following:

```
cops_api_error
cops_api_dispatch ( cops_resp *response)
```

The cops_api_dispatch() polls the PEP for a response and retrieves it for the CCAPI client. The parameter is a pointer to a cops_resp structure; if not NULL, it is eventually filled with the response. If the latter is not available the object referenced by this parameter is left unchanged. Moreover the callback function specified in the cops_api_open_session() (if any) is called. An explanation of the upcall mechanism can be found in paragraph 4. Note however that the cops_api_dispatch() is a non blocking call; if a response is not available, it immediately returns control to the calling function. Possible return values of cops_api_dispatch() are:

```
COPS_API_OK     - dispatch successfully executed
COPS_API_NOCOPS - COPS-ODRA PEP is not running on the ER
COPS_API_INVREQ _ invalid request; session not in place
COPS_API_SYSERR _ system error
```

3.2.8. cops_api_version()

The cops_api_version() call returns the version number of the CCAPI in the form major*100+minor. Current version is 1.00.

4. Upcall mechanism

An upcall is invoked by cops_api_dispatch(), which executes the procedure specified by the event_rtn parameter of the cops_api_open_session() call (if specified).

The upcall function has the following synopsis:

```
int
ccapi_callback ( cops_resp *response,
                 void *arg)
```

It receives from cops_api_dispatch() the response just arrived and the user specified parameter (represented by the event_rtn_arg parameter in the cops_api_open_session() call). Based on these parameters it executes an application specified routine. There are basically two types of events that can trigger an upcall:

- DECISION EVENT: this event notifies the CCAPI client about a response from the PDP/BB, which could be either positive (i.e. request accepted) or negative (i.e. the request was rejected for some reason, e.g. bandwidth unavailability or unsupported service).
- ERROR EVENT: it is used to signal error events directly recognized by the PEP, e.g. invalid request or excessive number or pending requests.

The type of event is contained in the response that is passed to the ccapi_callback() (see paragraph 5). It also contains a code that specifies the particular reason for that event. Possible codes for both DECISION EVENTS and ERROR EVENTS are contained in tables 1 and 2 below:

Code	Reason that triggered the event
COPS_OK	Request accepted
COPS_NOBW	Unavailable resources
COPS_NODSCP	Unsupported service
COPS_NOIED	Invalid Ingress Edge Device Address
COPS_NOEED	Invalid Egress Edge Device Address

Table 1: Codes for DECISION EVENTS (PDP/BB responses)

Code	Reason that triggered the event
COPS_API_OK	Operation successfully completed
COPS_API_INVRID	Invalid Request Identifier
COPS_API_INVREQ	Invalid Request or Session not in place
COPS_API_NOCOPS	COPS Client (PEP) not running on the ER
COPS_API_OPEN	Session already opened
COPS_API_CLOSE	Session already closed
COPS_API_TOOMF	Excessive number of requests

```
| COPS_API_SYSERR | System Error                             |
+-----------------+------------------------------------------+
```

Table 2: Codes for ERROR EVENTS

5. Definition of CCAPI objects

This appendix defines the main CCAPI objects, reported below along with a brief explanation.

- cops_req object

```
typedef struct Cops_Req
{
    unsigned int     request_ID;
    unsigned int     request_type;
    struct in_addr   IED_addr,EED_addr;
    unsigned int     dscp[2];
    unsigned int     msr_interval[2];
    unsigned int     token_size[2];
} cops_req;
```

This object is used by the CCAPI Client to specify information that the PEP will insert in the COPS REQ message when querying the PDP/BB. The fields have the following meaning:

- request_ID: request identifier. Each pair request/response is characterized by a unique request identifier, which is used to correlate the response with the corresponding request. The same request_ID cannot be contemporarily used for issuing different requests, otherwise COPS_API_INVRID is returned.
- request_type: type of request. The following list reports the possible values for this parameter. Each of these values must be used when the corresponding library routine on the right is called:

```
BW_REQUEST  _ bandwidth_request()
BW_RELEASE  _ bandwidth_release()
BW_MODIFY   _ bandwidth_modify()
CCAPI_CLOSE _ cops_api_release_session()
```

```
   - IED_addr:     IP address of the ingress Edge Device
   - EED_addr:     IP address of the egress Edge Device
   - dscp[]:       two-element vector. In the case of
                   bandwidth_request() and bandwidth_release()
                   calls, the first element contains the DSCP of
                   the requested service, while the second is
                   unspecified. In the case of bandwidth_modify()
                   the two elements contains respectively the old
                   and the new value for the DSCP.
   - token_size[], msr_interval[]: a pair of two element vectors.
                   The corresponding bandwidth is given by the
                   ratio:
                     token_size[n]/msr_interval[n]   n=0,1
                   In the case of bandwidth_request() and
```



```
                   bandwidth_release() only the first two
                   elements are meaningful; they contains the
                   bandwidth to be requested/released. In the
                   case of bandwidth_modify() the values
                   token_size[0]/msr_interval[0] and
                   token_size[1]/msr_interval[1] contains
                   respectively the old and the new value for the
                   bandwidth.
```

- cops_resp object

```
typedef struct Cops_Resp {
    unsigned int request_ID;
    cops_event_type resp_type;
    cops_error resp_errcode;
} cops_resp;
```

This object contains the responses received by the CCAPI Client. The fields have the following meaning:

```
   - request_ID:  is the same value contained in the request. It
                  is returned by the PEP to the CCAPI Client in
                  order to associate the response to the
                  corresponding request.
   -resp_type:    specifies the type of response. Two types are
                  currently supported:

                    DECISION_EVENT
                    ERROR_EVENT
```

The difference is explained in the previous paragraph.

-resp_errcode: depending on resp_type, it gives detailed information about the event that triggered the response. Possible values are reported in table 1 and table 2.

6. References

[INTSERV] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview", IETF RFC 1633, June 1994

[RFC2210] J. Wroclawski, "The Use of RSVP with Integrated Services", IETF RFC 2210, September 1997

[RFC2205] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification ", IETF RFC 2205, September 1997

[2BIT] K. Nichols, V. Jacobson, L. Zhang "A Two-bit Differentiated Services Architecture for the Internet", IETF RFC 2638, July 1999

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

[INTDIF] Y. Bernet, R. Yavatkar, P. Ford, F. Baker, L. Zhang, M. Speer, R. Braden, J. Wrocklaski, E. Felstaine, "A Framework for Integrated Services Operation Over DiffServ Networks", IETF <draft-ietf-issll-diffserv-rsvp-03.txt>, September 1999, Work in Progress

[IWQOS99] O. Schelen, A. Nilsson, J. Norrgard, S. Pink, "Performance of QoS Agents for Provisioning Network Resources", Proceedings of IFIP Seventh Internation Workshop on QoS (IWQoS'99), London, UK, June 1999

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

[COPS-ODRA] S. Salsano, "COPS Usage for Outsourcing Diffserv Resource Allocation", <draft-salsano-issll-cops-odra-00.txt>, February 2000, Work in Progress

[COPS-ISDS] S. Salsano, R. Mameli, "Integrated services over DiffServ network using COPS-ODRA", <draft-mameli-issll-is-ds-cops-00.txt>, February 2000, Work in Progress

7. Author Information and Acknowledgements

The author would like to thank Stefano Salsano, Eleonora Manconi and Luca Dell'Uomo for their support and their contribution in the prototype implementation.

Roberto Mameli
CoRiTeL consortium
Via di Tor Vergata 135
00133 _ Roma (Italy)

Phone: +39 06 20410038
EMail: mameli@coritel.it