Internet Draft
Expires February 11, 1999
File: draft-ietf-rsvp-rapi-01.ps

R. Braden
ISI
D. Hoffman
Sun Microsystems
August 1998

RAPI – An RSVP Application Programming Interface Version 5

August 11, 1998

Status of Memo

Abstract

This memo describes version 5 of RAPI, a specific API (application programming interface) for RSVP. The RAPI interface is one realization of the generic API contained in the RSVP Functional Specification document, and it is being published for information only. The RAPI interface is based upon a client library, whose calls are described here.

1 Introduction

An Internet application uses some *API* (Application Programming Interface) in order to request enhanced quality-of-service (QoS). A local RSVP control program will then use the RSVP protocol to propagate the QoS request through the routers along path(s) for the data flow. Each router may accept or deny the request, depending upon its available resources. In the case of failure, the local RSVP control program will return the decision to the requesting application via the API.

```
                         H O S T           .     R O U T E R
             (RAPI)                        .
               v                           .
  -------------v                           .
 |             |_______                    .
 |             x RSVP  |                   .
 |             x Client|                   .
 | Application x Lib   |   ------------    .     --------------
 |             x  rtns |  |            |   .     |             |
 |             x_______|  |    RSVP    |   .     |   RSVP      |
 |             |   |      |   Daemon   |   .     |  Daemon     |
 |_____________|   |      |____________|   .     |_____________|
            |      |        |   .    |     .       |    |    |
   USER     |      |        |   .    |     .       |    |    |
 ===========|======|========|===V====|=====.       |    |    |
   KERNEL   |      |_<____>_|   .    |___<_____>___|    |    |__>
            |       UNIX Pipe   .      RSVP messages    |   RSVP
            |                   .          .            |
            |              _____V______   .       _____V______
            |             |            |   .      |            |
            |             | Packet     |   .      |  Packet    |
            += DATA====>  | Classifier | =DATA==> | Classifier | ==>
                          |& Scheduler |   .      |& Scheduler |
                          |  (if any)  |   .      |            |
                          |____________|   .      |____________|
```

This document describes a particular RSVP API implementation known as *RAPI*. RAPI is based on a client library linked with the application. This document describes the calls to that library. There is at least one other documented API for RSVP, based on sockets.

The following diagram shows RAPI's implementation model. RSVP is implemented on a host by a user-level daemon program. The procedures of the RSVP client library module interact with the local RSVP daemon program through a Unix-domain socket. RAPI refers to the interface between the application and the RSVP client library.

1.1 Reservation Model

RSVP performs the signaling necessary to make a resource reservation for a simplex data flow sent to a unicast or multicast destination address. Although RSVP distinguishes senders from receivers, the same application may act in both roles.

RSVP assigns QoS to an specific multipoint-to-multipoint data flow known as a *session*. A session is defined by a particular transport protocol, IP destination address, and destination port. In order to receive data packets for a particular multicast session, a host must have joined the corresponding IP multicast group using the setsockopt call IP_ADD_MEMBERSHIP.

A data source, or *sender*, is defined by an IP source address and a source port. A given session may have multiple senders S1, S2, ... Sn, and if the destination is a multicast address, multiple *receivers* R1, R2, ... Rm. In the current version of RSVP, the ports used by RSVP for defining sessions and senders are restricted to be TCP/UDP port numbers.

Under RSVP, QoS requests are made by the data receivers. A QoS request contains a *flowspec* together with a *filter spec*. The flowspec includes an *Rspec*, which defines the desired QoS and is used to control the packet scheduling mechanism in the router or host, and also a *Tspec*, which defines the traffic expected by the receiver. The filter spec controls packet classification to determine which sender(s)' data packets receive the corresponding QoS.

The detailed manner in which reservations from different receivers are shared in the Internet is controlled by a reservation parameter known as the *reservation style*. The RSVP Functional Specification contains a definition and explanation of the different reservation styles.

1.2 API Outline

Using the RAPI interface, an application uses the rapi_session() call to define an *API session* for sending a single simplex data flow and/or receiving such a data flow. The rapi_sender() call may then be used to register as a data sender, and/or the rapi_reserve() call may be used to make a QoS reservation as a data receiver.The rapi_sender and/or rapi_reserve calls may be repeated with different parameters to dynamically modify the state at any time or they can be issued in null forms that retract the corresponding registration. The application can call rapi_release() to close the session and delete all of its resource reservations. The relationship among the RAPI library calls is summarized by the RAPI state diagram shown below. Rapi_sender(0) and rapi_reserve(0) represent null calls in that diagram.

Note that a single API session, defined by a single rapi_session call, can define only one sender at a time. More than one API session may be established for the same RSVP session. For example, suppose an application sends multiple UDP data flows, distinguished by source port. It will call rapi_session and rapi_sender separately for each of these flows.

The rapi_session call allows the application to specify an *upcall* (or "callback") routine that will be invoked to signal RSVP state change and error events. There are five types of upcalls.

- PATH_EVENT and RESV_EVENT upcalls signal the arrival or change of path state and reservation state, respectively, and deliver the relevant state information to the application.

- PATH_ERROR and RESV_ERROR upcalls signal the corresponding errors.
- PATH_CONFIRM upcalls signal the arrival of a CONFIRM message.

The upcall routine is invoked indirectly (and synchronously) by the application, using the following mechanism.

- The application issues the RAPI library call rapi_getfd() to learn the file descriptor of the Unix socket used by the API.
- The application detects read events on this file descriptor, either passing it directly in a select call or passing it to the notifier of another library (such as XLib, tk/tcl, RPC, etc.).
- When a read event on the file descriptor is signaled, the application calls rapi_dispatch(). This drives the API to execute the upcall routine if appropriate.

A synchronous error in a RAPI library routine returns an appropriate error code. Asynchronous RSVP errors are delivered to the application via the RAPI upcall routine. Text messages for synchronous and asynchronous error codes will be found in the file *rapi_err.h*.

The first rapi_session() call in a particular instance of the RAPI library opens a Unix-domain RAPI socket to the RSVP daemon and passes the session registration request across it. If the application (or the daemon) crashes without properly closing the RAPI socket, the other side will be notified to perform a cleanup. In particular, if the user process terminates without explicitly closing the RAPI session, the daemon will delete the corresponding reservation state from the routers.

```
                        RAPI State Diagram

                            +---------+
                            | Closed  |<-----+
                            +---------+      |
                                 |           |
                           rapi_session      |
                                 |           |
                        +----+   |     rapi_release
             PATH_EVENT |    V   V           |
                        |   +---------+      |
                        +---|         |------+
                            | Session |
                 +----------|         |----------+
           rapi_sender      +---------+    rapi_reserve
                 |           ^       ^           |
rapi_sender or   |           |       |           |   rapi_reserve or
PATH_EVENT or    | rapi_sender(0)    |           |   PATH_EVENT or
RESV_EVENT or    |           |       |           |   CONFIRM_EVENT
PATH_ERROR       |           |       |           |   or RESV_ERROR
        +----+   |           |  rapi_reserve(0)  |   +----+
        |    V   V           |       |           V   V    |
        |   +---------+      |       |      +---------+   |
        +---|         |------+       +------|         |---+
            |  Send   |                     |   Rcv   |
    +-------|         |<-----+       +----->|         |-------+
    |       +---------+      |       |      +---------+       |
    |            |           |       |           |            |
    |            | rapi_reserve(0)   |           |            |
    |            |           |       |           |            |
    |            |           |  rapi_sender(0)   |            |
    |            |           |       |           |            |
    |     rapi_reserve       |       |      rapi_sender       |
    |            |          +---------+          |            |
    |            +--------->|         |<---------+            |
    |                       | SendRcv |                       |
    |    rapi_sender or +---|         |                       |
    |   rapi_reserve or |   +---------+                       |
    |     PATH_EVENT or |    ^   |                            |
    |     RESV_EVENT or +----+   |                            |
    |     CONFIRM_EVENT          |                            |
    |     or PATH_ERROR          |                            |
    |     or RESV_ERROR          |                            |
    |                      rapi_release              rapi_release
  rapi_release                   V                            |
    |                        +--------+                       |
    +----------------------->| Closed |<----------------------+
                             +--------+
```

2 CLIENT LIBRARY SERVICES

The RSVP API provides the client library calls defined in this section. To use these calls, the application should include the file *rapi_lib.h* and *rsvp_intserv.h*.

A. Create a Session

The **rapi_session** call creates an API session. If it succeeds, the call returns an opaque but non-zero session handle for use in subsequent calls related to this API session. If the call fails synchronously, it returns zero (NULL_SID) and stores a RAPI error code into an integer variable pointed to by the *errnop* parameter.

After a successful rapi_session call has been made, the application may receive upcalls of type RAPI_PATH_EVENT for the API session.

unsigned int **rapi_session**(

```
struct sockaddr *Dest,      /* Session: (Dst addr, port) */
int         Protid,         /* Protocol Id               */
int         flags,          /* flags                     */
int         (*Event_rtn)(), /* Address of upcall routine */
void        *Event_arg,     /* App argument to upcall    */
int         *errnop         /* Place to return error code*/
)
```

The parameters are as follows.

- *Dest*

  This required parameter points to a sockaddr structure defining the destination IP (V4 or V6) address and a port number to which data will be sent. The *Dest* and *Protid* parameters define an RSVP session. If the *Protid* specifies UDP or TCP transport, the port corresponds to the appropriate transport port number.
- *Protid*

  The IP protocol ID for the session. If it is omitted (i.e., zero), 17 (UDP) is assumed.
- *flags*

  RAPI_GPI_SESSION (0x40) – If set, this flag requests that this API session be defined in the GPI format used by the IPSEC extension of RSVP. If this flag is set, the port number included in *Dest* is considered "virtual" (see the IPSEC specification for details), and any sender template and filter specifications must be in GPI format.

  RAPI_USE_INTSERV (0X10) – If set, IntServ formats are used in upcalls; otherwise, the Simplified format is used (see Section 4 below).

- *Event_rtn*

  This parameter is a pointer to an upcall routine that will be invoked to notify the application of RSVP errors and state change events. This parameter is required.

- *Event_arg*

  This optional parameter points to an argument that will be passed in any invocation of the upcall routine.

- *errnop*

  The address of an integer into which a RAPI error code will be returned.

An application can have multiple API sessions registered for the same or different RSVP sessions at the same time. There can be at most one sender associated with each API session; however, an application can announce multiple senders for a given RSVP session by announcing each sender in a separate API session.

Two API sessions for the same RSVP session, if they are receiving data, are assumed to have joined the same multicast group and will receive the same data packets. At present, if two or more such sessions issue *rapi_reserve* calls, their reservation parameters must agree or the results will be undefined. There is no check for such a conflict. Furthermore, the code does not disallow multiple API sessions for the same sender (defined by the host interface and the local UDP port) within the same RSVP session, i.e., for the same data flow. If these API sessions are created by different application processes on the local host, the data packets they send will be merged but their sender declarations will not be.

B. Specify Sender Parameters

An application must issue a **rapi_sender** call if it intends to send a flow of data for which receivers may make reservations. This call defines, redefines, or deletes the parameters of that flow. A **rapi_sender** call may be issued more than once for the same API session; the most recent one takes precedence.

If there is a synchronous error, **rapi_sender()** returns a RAPI error code; otherwise, it returns zero. Once a successful rapi_sender call has been made, the application may receive upcalls of type RAPI_RESV_EVENT or RAPI_PATH_ERROR.

int **rapi_sender**(

```
    int              Sid,            /* Session ID         */
    int              flags,          /* Flags              */
    struct sockaddr *LHost,          /* Local Host         */
    rapi_filter_t   *SenderTemplate, /* Sender template    */
    rapi_tspec_t    *SenderTspec,    /* Sender Tspec       */
    rapi_adspec_t   *SenderAdspec,   /* Sender Adspec      */
    rapi_policy_t   *SenderPolicy,   /* Sender policy data */
    int              TTL;            /* Multicast data TTL */
)
```

The parameters are as follows.

- *Sid*

  This required parameter must be a session ID returned by a successful **rapi_session** call.

- *flags*

  No flags are currently defined for this call.

- *LHost*

  This optional parameter may point to a sockaddr structure specifying the IP source address and the UDP source port from which data will be sent, or it may be NULL.

  If the IP source address is INADDR_ANY, the API will use the default IP (V4 or V6) address of the local host. This is sufficient unless the host is multihomed. The port number may be zero if the protocol for the session does not have ports.

  A NULL *LHost* parameter indicates that the application wishes to withdraw its registration as a sender. In this case, the following parameters will all be ignored.

- *SenderTemplate*

  This optional parameter may be a pointer to a RAPI filter spec structure (see Section 4) specifying the format of data packets to be sent, or it may be NULL.

  If this parameter is omitted (NULL), a sender template will be created internally from the *Dest* and *LHost* parameters. If a *SenderTemplate* parameter is present, the *LHost* parameter will be ignored.

  This parameter is required in order to declare the sender template for a session using IPSEC, i.e., a session created with the RAP_GPI_SESSION flag set.

- *SenderTspec*

  This required parameter is a pointer to a Tspec that defines the traffic that this sender will create.

- *SenderAdspec*

  This optional parameter may point to a RAPI Adspec structure (see Section 4), or it may be NULL.

- *SenderPolicy*

  This optional parameter may be a pointer to a sender policy data structure, or it may be NULL.

- *TTL*

  This parameter specifies the IP TTL (Time-to-Live) value with which multicast data will be sent. It allows RSVP to send its control messages with the same TTL scope as the data packets.

C. Make, Modify, or Delete a Reservation

The **rapi_reserve** procedure is called to make, modify, or delete a resource reservation for a session. The call may be repeated with different parameters, allowing the application to modify or remove the reservation; the latest call will take precedence. Depending upon the parameters, each call may or may not result in new Admission Control calls, which could fail asynchronously.

If there is a synchronous error in this call, **rapi_reserve()** returns a RAPI error code; otherwise, it returns zero. Once this call has been successful, the application may receive an upcall of type RAPI_RESV_ERROR or RAPI_RESV_CONFIRM.

An admission control failure (e.g., refusal of the QoS request) will be reported asynchronously by an upcall of type RAPI_RESV_ERROR. A *No Path State* error code indicates that RSVP state from one or more of the senders specified in *filter_list* has not (yet) propagated all the way to the receiver; it may also indicate that one or more of the specified senders has closed its API and that its RSVP state has been deleted from the routers.

int **rapi_reserve**(

```
int              Sid,          /* Session ID       */
int              flags,
struct sockaddr *RHost,        /* Receive host addr*/
int              StyleId,      /* Style ID         */
rapi_stylex_t   *Style_Ext,    /* Style extension  */
rapi_policy_t   *Rcvr_Policy,  /* Receiver policy  */

int              FilterSpecNo,    /* # of filter specs */
rapi_filter_t   *FilterSpec_list, /* List of filt specs*/
int              FlowspecNo,      /* # of flowspecs   */
rapi_flowspec_t *Flowspec_list    /* List of flowspecs*/
)
```

The parameters are as follows:

- *Sid*

  This required parameter must be a session ID returned by a successful **rapi_session** call.

- *flags*

  Setting the RAPI_REQ_CONFIRM flag will request confirmation of the reservation, by means of a confirmation upcall (type RAPI_RESV_CONFIRM).

- *RHost*

This optional parameter may be used to define the interface address on which data will be received. It is useful for a multi-homed host. If it is omitted or the host address is INADDR_ANY, the default interface will be assumed.

- *StyleId*

  This required parameter specifies the reservation style id (values defined below).

- *Style_Ext*

  This optional parameter is a pointer to a style-dependent extension to the parameter list, if any.

- *Rcvr_Policy*

  This optional parameter is a pointer to a policy data structure, or it is NULL.

- *FilterSpec_list*, *FilterSpecNo*

  The *FilterSpec_list* parameter is a pointer to an area containing a sequential vector of RAPI filter spec objects. The number of objexts in this vector is specified in *FilterSpecNo*. If *FilterSpecNo* is zero, the *FilterSpec_list* parameter will be ignored.

- *Flowspec_list*, *FlowspecNo*

  The *Flowspec_list* parameter is a pointer to an area containing a sequential vector of RAPI flow spec objects. The number of objects in this vector is specified in *FlowspecNo*. If *FlowspecNo* is zero, the *Flowspec_list* parameter will be ignored.

If FlowspecNo is zero, the **rapi_reserve** call will remove the current reservation(s) for the specified session, and FilterSpec_list and Flowspec_list will be ignored. Otherwise, the parameters depend upon the style, as follows.

- Wildcard Filter (WF)

  Use *StyleId* = RAPI_RSTYLE_WILDCARD. The *Flowspec_list* parameter may be empty (to delete the reservation) or else point to a single flowspec. The *FilterSpec_list* parameter may be empty or it may point to a single filter spec containing appropriate wildcard(s).

- Fixed Filter (FF)

  Use *StyleId* = RAPI_RSTYLE_FIXED. *FilterSpecNo* must equal *FlowspecNo*. Entries in *Flowspec_list* and *FilterSpec_list* parameters will correspond in pairs.

- Shared Explicit (SE)

  Use *StyleId* = RAPI_RSTYLE_SE. The *Flowspec_list* parameter should point to a single flowspec. The *FilterSpec_list* parameter may point to a list of any length.

D. Remove a Session

The **rapi_release** call removes the reservation, if any, and the state corresponding to a given session handle. This call will be made implicitly if the application terminates without closing its RSVP sessions. If the session handle is invalid, the call returns a corresponding RAPI error code; otherwise, it returns zero.

```
int {\bf rapi\_release}( unsigned int Sid )
```

E. Get File Descriptor

The **rapi_getfd** call may be used by the application to obtain the file descriptor associated with the Unix socket connected to the RSVP daemon, after a **rapi_session()** call has completed successfully and before **rapi_release()** is called. When a socket read event is signaled on this file descriptor, the application should call **rapi_dispatch()**, described below.

```
int {\bf rapi\_getfd} ( unsigned int Sid )
```

If Sid is illegal or undefined, this call returns -1; otherwise, it returns the file descriptor.

F. Dispatch API Event

The application should call this routine whenever a read event is signaled on the file descriptor returned by **rapi_getfd()**. Rapi_dispatch() may be called at any time, but it will generally have no effect unless there is a pending event associated with the Unix pipe. Calling this routine may result in one or more upcalls to the application from any of the open API sessions known to this instance of the library.

int **rapi_dispatch**( )

If this call encounters an error, **rapi_dispatch()** returns a RAPI error code; otherwise, it returns zero,

G. RAPI Version

int **rapi_version**( )

This call returns a single integer that defines the version of the interface. The returned value is composed of a major number and a minor number, encoded as 100*major + minor. This call may be used by an application to adapt to different versions.

The API described in this document has major version number 5.

H. Upcalls

An upcall (invoked by a call to rapi_dispatch()) executes the procedure whose address was specified by the *Event_rtn* in the **rapi_session()** call.

event_upcall(

```
unsigned int   Sid,          /* Session ID  */
int            EventType,    /* Event type  */
```

```
int             Style,          /* Resv style  */
int             ErrorCode,      /* (error event): err code */
int             ErrorValue,     /* (error event): err value*/
struct sockaddr *ErrorNode,     /* Node that detected error*/
unsigned char   ErrorFlags,

int             FilterSpecNo,  /* # of filter specs in list*/
rapi_filter_t *FilterSpec_list,
int             FlowspecNo,     /* # of flowspecs in list  */
rapi_spec_t    *Flowspec_list,
int             AdspecNo,       /* # of ADSPECs in list    */
rapi_adspec_t *Adspec_list,
void           *Event_arg       /* Supplied by application */
```

The following parameters are used in the upcall:

- *Sid*

  This parameter must be a session ID returned by a successful **rapi_session()** call.

- *EventType*

  Upcall event types.

- *Style*

  This parameter contains the style of the reservation; it is non-zero only for a RAPI_RESV_EVENT or RAPI_RESV_ERROR upcall.

- *ErrorCode, ErrorValue*

  These values encode the error cause, and they are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. ErrorCode values are defined in *rapi_lib.h* and corresponding text strings are defined in *rapi_err.h*.

- *ErrorNode*

  This is the IP (V4 or V6) address of the node that detected the error, and it is set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event.

- *ErrorFlags*

  These error flags are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event.

  RAPI_ERRF_InPlace (0x01) – The reservation failed, but another reservation (presumably smaller) reservation is still in place on the same interface.

  RAPI_ERRF_NotGuilty (0x02) – The reservation failed, but the request from this client was merged with a larger reservation upstream, so this client's reservation might not have caused the failure.

- *FilterSpec_list, FilterSpecNo*

The *FilterSpec_list* parameter is a pointer to a malloc'd area containing a sequential vector of RAPI filter spec or sender template objects. The number of objexts in this vector is specified in *FilterSpecNo*. If *FilterSpecNo* is zero, the *FilterSpec_list* parameter will be NULL.

- *Flowspec_list*, *FlowspecNo*

  The *Flowspec_list* parameter is a pointer to an area containing a sequential vector of RAPI flowspec or Tspec objects. The number of objects in this vector is specified in *FlowspecNo*. If *FlowspecNo* is zero, the *Flowspec_list* parameter will be NULL.

- *Adspec_list*, *AdspecNo*

  The *Adspec_list* parameter is a pointer to an area containing a sequential vector of RAPI adspec objects. The number of objects in this vector is specified in *AdspecNo*. If *AdspecNo* is zero, the *Adspec_list* parameter will be NULL.

- *Event_arg*

  This is the value supplied in the **rapi_session()** call.

When the application's upcall procedure returns, the API will free any areas pointed to by *Flowspec_list* or *FilterSpec_list*; the application must copy any values it wants to save.

The specific parameters depend upon *EventType*, which may have one of the following values.

- RAPI_PATH_EVENT

  A path event indicates that RSVP sender ("Path") state from a remote node has arrived or changed at the local node. A RAPI_PATH_EVENT upcall containing the complete current list of senders (or possibly no senders, after a path teardown) in the path state for the specified session will be triggered whenever the path state changes.

  FilterSpec_list, *Flowspec_list*, and *Adspec_list* will be of equal length, and corresponding entries will contain sender templates, sender Tspecs, and Adspecs, respectively, for all senders known at this node. In general, a missing object will be indicated by an empty RAPI object.

  RAPI_PATH_EVENT upcalls are enabled by the initial **rapi_session** call.

- RAPI_RESV_EVENT

  A reservation event upcall indicates that reservation state has arrived or changed at the node, implying (but not assuring) that reservations have been established or deleted along the entire data path to one or more receivers. RAPI_RESV_EVENT upcalls containing the current reservation state for the API session will be triggered whenever the reservation state changes.

  Flowspec_list will either contain one flowspec object or be empty (if the state has been torn down), and *FilterSpec_list* will contain zero or more corresponding filter spec objects. *Adspec_list* will be empty.

  RAPI_RESV_EVENT upcalls are enabled by a **rapi_sender** call; the sender template from the latter call will match the filter spec returned in a reservation event upcall.

- RAPI_PATH_ERROR

  A path error upcall indicates that an asynchronous error has been found in the sender information specified in a **rapi_sender** call.

  The *ErrorCode* and *ErrorValue* parameters will specify the error. *FilterSpec_list* and *Flowspec_list* will each contain one object, the sender template and corresponding sender Tspec (if any) in error, while *Adspec_list* will be empty. If there is no sender Tspec, the object in *Flowspec_list* will be an empty RAPI object. The *Adspec_list* will be empty.

  Path Error upcalls are enabled by a **rapi_sender** call, and the sender Tspec in that call will match the sender Tspec returned in a subsequent path error upcall.

- RAPI_RESV_ERROR

  A reservation error upcall indicates that an asynchronous reservation error has occurred.

  The *ErrorCode* and *ErrorValue* parameters will specify the error. *Flowspec_list* will contain one flowspec, while *FilterSpec_list* may contain zero or more corresponding filter specs. *Adspec_list* will be empty.

- RAPI_RESV_CONFIRM

  A confirmation upcall indicates that a reservation has been made at least up to an intermediate merge point, and probably (but not necessarily) all the way to at least one sender. A confirmation upcall is enabled by a **rapi_reserve** call with the RAPI_REQ_CONFIRM flag set, and at most one confirmation upcall will result from each such call.

  The parameters of a confirmation upcall are the same as those for a reservation event upcall.

The accompanying table summarizes the upcalls; here n is a non-negative integer.

Upcall	Enabled by	FilterSpecNo	FlowspecNo	AdspecNo
Path event	rapi_session	n	n	n or 0
Path error	rapi_sender	1	1	0
Resv event	rapi_sender	n	1 or 0	0
Resv error	rapi_reserve	n	1	0
Confirm	rapi_reserve	1	1	0

Table 1: Summary of Upcall Types

3 RAPI FORMATTING ROUTINES

For convenience of applications, RAPI includes standard routines for displaying the contents of a RAPI flowspec object or Tspec object. To use these routines, include the file *rapi_lib.h*.

A. Format a Flowspec

The **rapi_fmt_flowspec()** call formats a given RAPI flowspec into a buffer of given address and size. The output is truncated if the size is too small.

void **rapi_fmt_flowspec**(

```
rapi_flowspec_t  *specp,   /* Addr of RAPI flowspec*/
char             *buffer,  /* Addr of buffer       */
int               length   /* Length of buffer     */
)
```

B. Format a Tspec

The **rapi_fmt_tspec()** call formats a given RAPI Tspec into a buffer of given address and size. The output is truncated if the size is too small.

void **rapi_fmt_tspec**(

```
rapi_tspec_t    *tspecp,  /* Addr of RAPI Tspec  */
char            *buffer,  /* Addr of buffer      */
int              length   /* Length of buffer    */
)
```

C. Format an Adspec

The **rapi_fmt_adspec()** call formats a given RAPI Adspec into a buffer of given address and size. The output is truncated if the size is too small.

void **rapi_fmt_adspec**(

```
rapi_adspec_t    *adspecp,  /* Addr of RAPI Adspec */
char             *buffer,   /* Addr of buffer      */
int               length    /* Length of buffer    */
)
```

D. Format a Filter Spec

The **rapi_fmt_filtspec()** call formats a given RAPI Filter Spec into a buffer of given address and size. The output is truncated if the size is too small.

void **rapi_fmt_filtspec**(

```
rapi_filter_t    *filtp,    /* Addr of RAPI Filt Spec*/
char             *buffer,   /* Addr of buffer        */
int               length    /* Length of buffer      */
)
```

4 RAPI OBJECTS

Flowspecs, filter specs, sender templates, and sender Tspecs are encoded as variable-length RAPI objects.

Every RAPI object begins with a header consisting of two words, the total length of the object in bytes and the type, respectively. An empty object consists only of a header, with type zero and length 8 bytes.

Integrated services data structures are defined in: draft-ietf-intserv-rsvp-01.txt.

- Flowspecs

  There are two formats for RAPI flowspecs. For more details, see *rapi_lib.h* and *rsvp_intserv.h*.

  - RAPI_FLOWSTYPE_Simplified

    This is a "simplified" format. It consists of a simple list of parameters needed for either Guaranteed or Controlled Load service, using the service type QOS_GUARANTEED or QOS_CNTR_LOAD, respectively. The RAPI client library routines will map this format to/from an appropriate Integrated Services data structure.
  - RAPI_FLOWSTYPE_Intserv

    This flowspec must be a fully formatted Integrated Services flowspec data structure.

- Sender Tspecs

  There are two formats for RAPI Sender Tspecs. For more details, see *rapi_lib.h* and *rsvp_intserv.h*.

  - RAPI_TSPECTYPE_Simplified

    This is a "simplified" format, consisting of a simple list of parameters with the service type QOS_TSPEC. The RAPI client library routines will map this format to/from an appropriate Integrated Services data structure.
  - RAPI_TSPECTYPE_Intserv

    This flowspec must be a fully formatted Integrated Services Tspec data structure.

- Adspecs

  There are two formats for RAPI Adspecs. For more details, see *rapi_lib.h* and *rsvp_intserv.h*.

  - RAPI_ADSTYPE_Simplified

    This is a "simplified" format, consisting of a list of Adspec parameters for all possible services. The RAPI client library routines will map this format to/from an appropriate Integrated Services data structure.

  - RAPI_ADSTYPE_Intserv

    This flowspec must be a fully formatted Integrated Services Tspec data structure.

  In an upcall, a flowspec, sender Tspec, or Adspec is by default delivered in simplified format; however, if the RAPI_USE_INTSERV flag is set in the rapi_session call, then the IntServ format is used in upcalls.

- Filter Specs and Sender Templates

  There are two formats for these objects.

  - RAPI_FILTERFORM_BASE (RAPI_FILTERFORM_BASE6)

    This object consists of only a socket address structure, defining the IP (V4 or V6) address and port.

  - RAPI_FILTERFORM_GPI (RAPI_FILTERFORM_GPI6)

    This object consists of only an address structure, defining the IP (V4 or V6) address and a 32-bit Generalized Port Identifier. It is recommended for all IPSEC applications. Other non-TCP/non-UDP transports may also utilize this format in the future.

- Policy Data Objects

  (Not yet supported)

A Implementation

This section contains some general remarks on the implementation of this API that is distributed with the ISI release of RSVP code.

A.1 Protocols

There are three protocol interfaces involved in invoking RSVP via the API.

1. Procedure Call Interface to Application

   The term "RAPI" (RSVP API) is used for the procedure call interface to applications, and for the data structures (*objects*) used in that interface. This document is primarily concerned with the RAPI interface. This interface is realized by procedures included in the library routine librsvp.a, which is compiled from rapi_lib.c and rapi_fmt.c.

2. Application - Daemon Protocol

   The term "API" is used in the code for the local protocol across the Unix socket between the librsvp.a routines and the RSVP daemon rsvpd. This protocol generally use RSVP object bodies but RAPI object framing.

3. RSVP Protocol

   The RSVP protocol is used in the Internet between RSVP daemon programs.

The code is organized to make these three interfaces logically independent, so they can be changed independently. Each of these three protocol interfaces has an independent version number, defined in rapi_lib.h, rsvp_api.h, and rsvp.h for RAPI, API, and RSVP, respectively.

The RAPI call library librsvp.a includes routines that convert objects between RAPI and API formats. Similarly, the file rsvp_api.c included in the RSVP daemon includes routines that convert between the API representation and the RSVP representation. In some cases, these conversion procedures are identity transformations (i.e., pure copies); however, they provide the structure to allow any of the three interfaces to be changed in the future.

There are two different object framing conventions. RAPI and API objects have a two-word header – a total length in bytes and a format code – and a body. RSVP objects have a one-word header. In general, objects in the API interface (i.e., across the Unix socket) carry the two-word RAPI object header, but their body is that of the corresponding RSVP object. Therefore, the API¡-¿RSVP conversion in rsvp_api.c simply maps the framing convention.

In the RAPI interface, the application is given some choice of data formats. For example, QoS control objects (i.e., flowspecs, Tspecs, and Adspecs) can be represented in either the RSVP (really

Int-Serv) format, which has complex packing, or in the more convenient Simplified format. The RAPI library routines map between Simplified format and Int-Serv format, which is used across the API.

A.2 RAPI Sessions

Each instance of the RAPI library routines keeps a local (to the application process) table of open RAPI sessions; the index into this table is the session handle (a_sid) used locally. The RSVP daemon keeps its own table of RAPI sessions. From the daemon's viewpoint, a RAPI session is defined by the triple: (fd, pid, a_sid), where fd is the file descriptor for the Unix socket, pid is the Unix process id, and a_sid is an application session id received over fd from pid.

B Implementation Restrictions

This Appendix summarizes the features of the interface that have not been implemented in the latest (4.1a6) release of the ISI reference implementation of RSVP.

- The RAPI_FILTERFORM_GPI and RAPI_FILTERFORM_GPI objects and the session flag RAPI_GPI_SESSION are implemented in RAPI and the API, but the IPSEC extensions are not yet fully implemented in RSVP.
- The *SenderAdspec*, and *SenderPolicy* parameters in a **rapi_sender** call are not implemented.
- The *Style_Ext* and *Rcvr_Policy* parameters in a **rapi_reserve** call are not implemented.

C CHANGES

This document describes major version 5 of RAPI. This version has the following differences from previous versions:

- The "Legacy" format has been removed.
- The rapi_fmt_filtspec() routine has been added.
- The two session flags RAPI_GPI_SESSION and RAPI_USE_INTSERV have been defined.
- The ErrorNode parameter in an upcall has been changed from a sockaddr to a pointer to a sockaddr structure, to accommodate IPv6.
- IPv4-specific Socket structures sockaddr_in have been changed to the more general form sockaddr, to accomodate IPv6. The calling application should supply the appropriate form, sockaddr_in or sockaddr_in6, and cast it into a sockaddr for the call.
- The Event_rtn parameter to rapi_session call is now required.