Internet Draft
Expiration: January 25, 1998
File: draft-ietf-rsvp-pepci-00.txt

Jim Boyle
MCI
Ron Cohen
Class Data Systems
Laura Cunningham
MCI
David Durham
Intel
Arun Sastry
Cisco
Raj Yavatkar
Intel

Protocol for Exchange of PoliCy Information (PEPCI)

July 25, 1997

Status of this Memo

Abstract

This document describes a simple client/server model for supporting policy for RSVP, and is designed to be extensible so that other kinds of client types may be supported in the future. The model does not make any assumptions about the algorithm of the policy server, but is based on the server returning a single priority value in response to a policy request. The objective is to use this very basic model to begin policy experimentation.

1. Introduction

This document describes Protocol for Exchange of PoliCy Information (PEPCI) which can be used to exchange policy information between a policy server and its clients. The policy clients are expected to be RSVP routers, that must exercise policy-based admission control over RSVP usage. We assume that at least one policy server exists in each routing domain. The basic model of interaction between a policy server and its clients is compatible with the RSVP extensions for policy control [EXT].

A chief objective of our proposal is to begin with a simple design for easy/quick deployment, testing, and experimentation. The main characteristics of the proposed protocol include:

1. The protocol uses TCP as the transport protocol for reliable exchange of messages between policy clients and a server. Therefore, no additional mechanisms are necessary for reliable communication between a server and its clients.

2. The protocol is designed to leverage off existing RSVP implementations and makes extensive use of RSVP-like self-identifying objects.

3. Even though the protocol is mainly intended for administration and enforcement of policies in conjunction with RSVP, the protocol may be extended for administration of other policies such as multicast group access and network security.

4. The protocol relies on existing protocols for message authentication. Namely IPSEC [IPSEC] can be used to authenticate and secure the channel between the LPM and the server and RSVP MD5 message authentication [MD5] can be used for inter-node authentication.

5. Messages are exchanged asynchronously and there is no need for error control or specific sequencing of messages.

1.1. Basic Model

We assume that each participating router has a Local Policy Module (LPM) [LPM] and may communicate with a policy server for policy decisions. It is assumed that most communication with a policy server will be done by border routers upon entry of an RSVP message into a routing domain, although this protocol is not restricted to such a model.

A policy client establishes a TCP connection to the policy server to begin communication and uses the connection to send requests to and receive responses from the server. Communication between client and server is mainly in the form of a request/response exchange, though the server may occasionally send an unsolicited response to the client to force a change to a previously approved state.

The response from the server is in the form of Accept(Priority). The priority returned by the policy server is a non-negative integer indicating priority. Higher numbers indicate higher priority, and the LPM interprets 0 as an indicator to completely deny the request. A single policy value indicating priority enables the routers to sort and kill sessions without requiring server intervention. For example, suppose a router has already successfully admitted and installed a reservation with priority 5. Later, if a new reservation request comes in and is approved by the policy server at priority 10, but cannot be admitted due to local admission control at the client, the client can remove the previously admitted reservation (with priority 5) to make room for the newer, higher priority reservation.

The LPM keeps state of known RSVP messages and processes policy as part of admission control. In particular, the LPM keeps track of the priority associated with each reservation message received. When a new PATH or RESV message is received, the LPM sends a new request message to the server. The client includes RSVP objects from the message in question and establishes a request identification handle (RIH) for future reference to this message. It should be noted that this is done rather early in RSVP processing [RSVPPROC]. The server responds with ACCEPT/REJECT indication and may optionally include objects that provide for modification of the original message. If the message is accepted, it is further processed by RSVP including RESV merging with other messages if necessary. If RESV messages are merged, the client may end up with several policy objects to merge.

This is resolved by attaching the Policy Object of the "largest" downstream RESV to the forwarded RESV message. In the event of a "tie" (i.e. there are multiple reservations that can be considered the "largest" reservation), we will include the policy objects from all the reservations.

For example, if a router receives two RESV messages for the same session, it will check with the policy server separately for each message and then keep track of the priority received as part of the RSB for each message. When the two RESVs are successfully merged, the merged RESV is forwarded with the policy object of the "larger" original message. If the higher priority reservation is later torn down, the existing reservation would then revert to the next "largest" reservation. The RSVP implementation must keep track of the associated priority. This could result in the lower priority reservation "riding" the priority of a higher reservation and then being torn down once the higher priority reservation is gone and other reservations pre-empt the lower priority one. This is considered acceptable as a side effect of merging benefits.

Under this model, both the policy server and its client maintain state associated with a particular request. Failure of a client or the server is detected by the loss of a TCP connection. Upon failure, a client connects to a new server and the new server syncs up with the client.

2. The Protocol

This section describes the message formats and objects used by the LPM and Policy Server.

2.1 Common Header

Each PEPCI message consists of the PEPCI header followed by a number of client-specific objects.

```
               0              1              2              3
        +--------------+--------------+--------------+--------------+
        |Version|Flags |       Client Type           |    Op Code   |
        +--------------+--------------+--------------+--------------+
        |                            RIH                            |
        +--------------+--------------+--------------+--------------+
        |                      Message Length                       |
        +--------------+--------------+--------------+--------------+
```

The fields in the header are:

Version: 4 bits

PEPCI version number. Current version is 1.

Flags: 4 bits

Flag bits

Client Type: 16 bits

The type identification for the policy client Interpretation of all encapsulated objects is relative to the client type. The client type of 1 indicates an RSVP client using RSVP V1 objects. In the future, further types may be defined to accommodate types of policies other than bandwidth and to accommodate new versions of RSVP.

Op Code: 8 bits

The PEPCI operations:

1 = Request Query	(RQ)
2 = Request Response	(RR)
3 = Request Allowed	(RA)
4 = Delete Request	(DRQ)
5 = Synchronize State Req	(SSQ)
6 = Synchronize State Resp	(SSR)
7 = Unsolicited Response	(USR)

RIH (Request Identification Handle): 32 bits

Client side value to uniquely identify message/association. For example, an RSVP client will provide a handle to identify a reservation request so that subsequent operations that apply to the same message can be easily identified. Similarly, if PATH control is desired, an RSVP client would send the RSVP objects associated with the PATH to the server and supply a RIH handle for future references to this PATH state.

Message Length: 32 bits

Size of message in bytes. This includes all encapsulated objects, but not including the standard PEPCI header.

2.2 Object Formats

All the objects follow the RSVP object format; each object consists of one or more 32-bit words with a one-word header, with the following format:

```
            0             1              2             3
     +-------------+-------------+-------------+-------------+
     |       Length (bytes)      |  Class-Num  | C-Type      |
     +-------------+-------------+-------------+-------------+
     |                                                       |
     //                  (Object contents)                   //
     |                                                       |
     +-------------+-------------+-------------+-------------+
```

The Class Numbers are chosen to start with high values so as not to conflict with Class Number values already defined for RSVP objects. The choice of PEPCI specific class numbers ensures that PEPCI-specific objects are never forwarded beyond the policy client.

2.2.1 RSVP Objects

RSVP Objects can be copied as is into PEPCI messages. The first Request Query which initializes the RIH for the message includes a large portion of the RSVP message. In a PATH message, for instance, there is no relevant information in the Integrity object and the relevant information in the RSVP common header is included with the PEPCI context object. So, the initial PEPCI request query from the LPM to the server about an RSVP PATH message includes all the remaining RSVP objects starting with the session object. These are not encapsulated into PEPCI objects. RESV RSVP messages will contain the Session, Flowspec, Style, and, if applicable, the Filter objects. The server can return a Policy Object within a Request Response which the LPM must substitute for the Policy Object(s) that the router received within that RSVP message.

2.2.2 Priority Object

As stated earlier, the priority object is used to specify the relative priorities among different reservation requests. A priority of zero indicates a DENY.

A priority value is encoded as an unsigned, 16-bit integer value.

```
     Class-Num = 128, C-Type = 1
             0             1              2              3
     +--------------+--------------+--------------+--------------+
     |          Priority           |  ////    Reserved     ////  |
     +--------------+--------------+--------------+--------------+
```

2.2.3 Handle Object

The handle object is designed to carry a handle that identifies a particular association (such as a complete reservation request or parts such as a particular PATH state).

The handle is a 32-bit number chosen by a policy client at the time of sending a new request to the policy server.

Handles are optional for both the client and server, and there is no special negotiation needed (between the client and server) to determine the usage of the handle.

```
     Class-Num = 129, C-Type = 1

     +--------------+--------------+--------------+--------------+
     |              Request Identification Handle                |
     +--------------+--------------+--------------+--------------+
```

2.2.4 Reason Code Object

A one octet, integer value used to provide additional reasons for a particular response or a particular delete state notification.

```
     Class-Num = 130, C-Type = 1

     +--------------+--------------+--------------+--------------+
     | Reason code  | ///////        RESERVED ///////            |
     +--------------+--------------+--------------+--------------+
```

2.2.5 Hold Off Timer Object

The Hold Off Timer is used to specify the length of time for which a given policy is valid, or the length of time the LPM should wait before asking the policy server for a new policy value for a given RIH. This timer acts as a simple mechanism to prevent denial of service attacks on a policy server. It also works to ensure that policy information must be renewed periodically.

Times are encoded as 32-bit integer values and are in units of seconds. The time value is treated as a delta from the point at which the LPM receives the message containing the Hold Off Timer.

LPM implementation of this object is mandatory for clients, but its use by servers is optional.

```
     Class-Num = 131, C-Type = 1

     +--------------+--------------+--------------+--------------+
     |                      Hold Off Timer                       |
     +--------------+--------------+--------------+--------------+
```

2.2.6 Interface Object

The interface object is used to identify particular interfaces on a router. It is a 32-bit integer field whose value is the same as the SNMP ifIndex value for that interface.

There are two types of interfaces : incoming interfaces and outgoing interfaces. An incoming interface is the interface on which the RSVP message was received, and an outgoing interface is one on which the RSVP message is being forwarded.

```
      in-interface:
           Class-Num = 132, C-Type = 1

      out-interface
           Class-Num = 133, C-Type = 1

                0             1               2               3
       +--------------+--------------+--------------+--------------+
       |                         ifIndex                           |
       +--------------+--------------+--------------+--------------+
```

2.2.7 Context Object

The context object carries the RSVP message type (PATH, RESV, etc.) of the RSVP message that triggered the query.

```
        Class-Num = 134, C-Type = 1
                 0             1               2               3
        +--------------+--------------+--------------+--------------+
        | RSVP MsgType |       ////    Reserved     ////            |
        +--------------+--------------+--------------+--------------+
```

2.3 **Request Query (RQ)** LPM -> Policy Server

The client establishes a Request Identification Handle (RIH) which the server maintains a state for, and uses to refer to this RSVP message. It also sends portions of the RSVP message so Policy Servers of varying complexity can use any information from the message without requiring that the LPM make a determination of what to parse out and send to the server.

Once a RIH is established with a new request, any subsequent modifications of the request can be made using the RQ message with a previously established RIH. For example, when a change in a reservation happens on a refresh (or some other means such as SNMP-based state change on a router), the router will simply supply the new information in a RQ message with the existing RIH associated with the reservation state.

The format of the Request Query message is as follows:

```
           <Request Query> ::= <Common Header>
                               <Context><in-interface>
                               <RSVP Objects>
                               [<Additional objects>]
```

The additional objects are optional and can give more information on the RSVP state. For example, in queries with Path context, the additional objects may be a list of out-interface objects which specify the outgoing interfaces on which this Path message is going to be forwarded. Similarly, we can have a Reservation query with multiple objects for the associated PATH states, e.g.

```
           <Request Query> ::= <Common Header>
                               <Context=RESV><in-interface>
                               <RSVP RESV Objects>
                               <associated path handle #1>
                               <associated path handle #2>
                               <associated path handle #3>
```

2.4. **Request Response (RR)** Policy Server -> LPM

The server responds to the RQ with a RR message that includes the associated RIH and the response. The priority value included in the response indicates the result such as Deny (priority = 0 means Reject) or Accept. In addition, the response may optionally include policy objects (OUT_POLICY), whose structure is defined in [EXT], to replace the incoming policy object(s). This assumes wholesale replacement of a previously received policy object(s) with appropriate modifications.

In order to avoid the issue of keeping track of which Request Query a particular response belongs to, it is important that, for a given RIH, there be at most one outstanding response per query. This essentially means that the client should not issue more than one RQ (for a given RIH) before it receives a corresponding RR.

The format of the Request Response message is as follows:

```
         <Request Response> ::= <Common Header>
                                <Priority>
                                [ <Hold Off Timer> ]
                                [ <OUT_POLICY> ]
                                [ <Additional objects> ]
```

The additional objects are optional and can give more information on replacement of policy objects, and can permit the extension of policy enforcement capabilities. For example, the additional objects may carry <out-interface><OUT_POLICY> pairs, indicating that when forwarding the message out that particular interface, the policy object associated with this interface should supersede the policy object received. This may be useful in multicast cases where different policy objects should be forwarded out different interfaces. The exact format of additional objects is left for future work. A router that does not support these additional objects should ignore them.

2.5. Request Allowed (RA) LPM -> Policy Server

This message serves as an acknowledgment to the server that a particular request response has been acted upon.

```
        <Request Allowed> ::== <Common Header>
                               <Priority>
```

2.6. Delete Request (DRQ) LPM -> Policy Server

This message indicates to the server that the PATH or RESV state has been deleted. This will be used by the server to initiate appropriate clean up actions. Reasons may include: PATH_ or RESV_TEAR, pre-emption, SNMP, loss of soft state.

The format of the Delete Request message is as follows:

```
        <Delete Request>  ::= <Common Header>
                              <Reason Code Object>
```

Reason Code: 16 bits

Reason Code =	0	Unknown
	1	Priority Changed
	2	Pre-empted
	3	TEAR
	4	SNMP request
	5	Loss of Soft State

2.7. Synchronize State Request (SSQ) Policy Server -> LPM;

The server uses this message to request a list of state that has been approved and not yet deleted. A case where this would be used is in server connection startup time. A long or short response may be provided, and is indicated by a bit in the flag value. A short answer provides just a list of RIH values with their current priority and their context and incoming interface. A long answer additionally provides the original RSVP message along with the OUT_POLICY object.

```
   Flag:         LONG_ANSWER             0x1
```

Each RSVP message is sent in a separate policy message.

The format of the Synchronize State Query message is as follows:

```
                <Synchronize State> ::= <Common Header>
```

If the RIH is specified (a nonzero value), the server queries about the state of a particular request. RIH=0 indicates that server wishes to synchronize all the state.

2.8. Synchronize State Response (SSR) LPM -> Policy Server

The format of the Synchronize State Response message is as follows:

```
                <Synchronize State> ::= <Common Header>
                        <Priority #1><Handle #1>
                        <Context><in-Interface>
                        If Long:
                        <RSVP Object><OUT-POLICY>
                        endIf Long:
                        <Priority #2><Handle #2>
                        <Context><in-Interface>
                        If Long:
                        <RSVP Object><OUT-POLICY>
                        endIf Long:
```

2.9 Unsolicited Response (USR) Policy Server -> LPM

The server can also send an unsolicited response to a client. One example where this can happen is when a policy change is made at the server, and a corresponding change needs to be effected at the client (e.g. change a policy for a particular reservation to DENY, so that reservation needs to be deleted.)

The format for an USR is the same as that for a RR.

2.10 ResvErr and PathErr control

Policy control over RSVP Error messages is left as an option. RSVP error messages carry policy objects which may add information to the RSVP nodes along the way.

Policy error messages generated by the router after the server has denied a query request should carry the policy objects returned in the query response.

Error messages received from other routers are handled much like Path and Resv messages. Since policy error messages do not create states, the only PEPCI messages used are Request Query and Request Response. The router should use a temporary handle that will allow it to match reply to response, but otherwise has no significance.

3. Operation

This section lists some sample exchanges between policy servers and LPM clients.

3.1. Client receives a new RSVP message, gets permission from Policy server, and, later, deletes the state when RESV is torn down.

```
        Client -> Server: RQ
                "RIH=4, RSVP objects incl. policy data"
        Server -> Client: RR
                "RIH=4, Priority=5, OUT-POLICY"
        Client -> Server: RA
                "RIH=4, Priority=5"
        Client -> Server: DRQ
                "RIH=4, Reason Code = TEAR"
```

Client gets another RESV for the same session. We assume that the client first checks with the policy server and then does local merging, before forwarding the resulting policy objects within the merged RESV towards PHOP(s).

```
        Client -> Server: RQ
                "RIH=11, objects in new incoming RESV inc' policy data"
        Server -> Client: RR
                "RIH=11, Priority=6, OUT-POLICY"
        Client -> Server: RA
                "RIH=11, Priority=6"
```

3.2. Server changes priority of an existing request.

```
        Server -> Client: USR
                "RIH=4, NewPriority = 10, Reason Code = 1, OUT-POLICY"
        Client -> Server: RA
                "RIH=4, Priority=10"
```

or, Server decides to pre-empt or abort a request accepted earlier by sending an USR with priority zero

```
        Server -> Client: USR
                "RIH=4, NewPriority = 0, Reason Code = 1, OUT-POLICY"
        Client -> Server: DRQ
                "RIH=4, Reason Code = Preempt"
```

3.3. Example of use of handle objects

```
        Client receives a PATH message, first contact the server for
        PATH control

        Client -> Server: RQ
               "RIH = 100, RSVP objects in the PATH message,
                policy data"
        Server -> Client: RR
                "RIH = 100, OUT_POLICY"
```

Later, the client receives a RESV for the same session and wishes to include the PATH state info in its request. It uses RIH=100 (previously established handle) to associate the relevant PATH state with its NEW request as in:

```
        Client -> Server: RQ
                "RIH=7, RSVP objects, Policy data, Handle = 100"
```

Server examines the information in the Query and the information about the Path state stored from previous query on Path and reaches a policy decision:

```
        Server -> Client: RR
                "RIH=7, priority = 1, OUT-Policy"
```

3.4. Server inquires about RIH 4.

```
        Server -> Client: SSQ (Long)
                "RIH=4"
        Client -> Server: SSR (Long)
                "RIH=4, Priority=10, OUT-POLICY,
                RSVP Objects "
```

3.5 Server requests a list of state.

```
        Server -> Client: SSQ(Long, RIH=0)
        Client -> Server: SSR(Long)
                     "RIH=2, Context, In-Interface,
                           Priority=1, OUT-POLICY, RSVP_MESSAGE
                      RIH=4, Context, In-Interface,
                           Priority=10, OUT-POLICY, RSVP_MESSAGE
                      RIH=5, Context, In-Interface,
                           Priority=10, OUT-POLICY, RSVP_MESSAGE
                      RIH=8, Context, In-Interface,
                           Priority=100, OUT-POLICY, RSVP_MESSAGE"
```

3.6 State Torn Down

```
        Client -> Server: DRQ
         "RIH=4, Reason=Tear"
```

3.7 Admission error handling

The client receives a RESV message and determines that this reservation was previously admitted using handle 100. Assuming that the flowspec of the new reservation is different, we might have something like:

```
         Client -> Server:  RQ
                  "RIH=100 NewFlowSpec, Policy data"
         Server -> Client:  RR
                  "RIH=100, Priority=4"
```

The client tries to admit the reservation. If it fails, it tries to preempt installed reservations with lower priority. If it is still unable to admit the reservation, it does not send a RA indication, and performs the admission error operations as defined in [RSVP], including sending a ResvErr frame. According to [RSVP] the client should still keep the old active installed reservation. The client will send a DRQ to the server only if it deletes an active (RESV/PATH) state. In order to keep the server in sync, the client will reissue a query to approve its active state:

```
       Client -> Server:  RQ
                 "RIH=100 ActiveFlowSpec, Policy data"
       Server -> Client:  RR
                 "RIH=100, Priority=4"
       Client -> Server:  RA
                 "RIH=100, Priority=4"
```

3.8 ADSPEC control

The following example describes a possible use of PEPCI to control ADSPEC values. The receiver uses the ADSPEC values received in the PATH message to decide on what QoS parameters are sent in the RESV message. The server may want to update the parameter AVAILABLE-PATH BANDWIDTH [INSCH] in the ADSPEC. This value carries information about the maximum bandwidth the receiver can successfully reserve due to physical resources limitations and bandwidth policy limitations.

```
     Client -> Server:  RQ
            "RIH=12, Path objects including Adspec, out-intfc 2 "
```

Server pulls the ADSPEC from the request, and updates the Available path bandwidth parameter.

```
     Server -> Client: RR
             "RIH=12, Priority=2, out-intfc=2, newAdspec "
```

Client updates the values in newAdspec, if necessary, and sends it in the PATH message sent via interface 2.

4. Security

As mentioned in Section 2, security of RSVP messages is provided by inter-router MD5 authentication. This assumes a chain-of-trust model for inter LPM authentication. Security between LPM and server is provided by IPSEC.

To ensure an LPM is talking to the correct policy server involves two issues: authentication of the policy client and server using a shared secret, and consistent proof that the connection remains valid. The shared secret requires manual configuration of keys, which is a maintenance issue. For validation of the connection, IPSEC AH will be used.

5. Open issues

6. References

[RSVP] Braden, R. ed., "Resource ReSerVation Protocol (RSVP) - Functional Specification." Internet-Draft, draft-ietf-rsvp-spec-16.txt, June 1997.

[EXT] Herzog, S., "RSVP Extensions for Policy Control." Internet-Draft, draft-ietf-rsvp-policy-ext-02.txt, April 1997

[INSCH] Shenker, S., Wroclawski, J., "General Characterization Parameters for Integrated Service Network Elements" Internet-Draft, draft-ietf-intserv-charac-02.txt, October 1996

[IPSEC] Atkinson, R., "Security Architecture for the Internet Protocol." RFC1825, August 1995.

[MD5] Baker, F., "RSVP Cryptographic Authentication." Internet-Draft, draft-ietf-rsvp-md5-03.txt, May 1997.

[LPM] Herzog, S., "Local Policy Modules (LPM): Policy Control for RSVP." Internet-Draft, draft-ietf-rsvp-policy-lpm-01.ps, November 1996.

[RSVPPROC] Braden, R., Zhang, L., "Resource ReSerVation Protocol (RSVP) - Version 1 Message Processing Rules." Internet-Draft, draft-ietf-rsvp-procrules-00.txt, November 1996.

6. Author Information and Acknowledgments

Thanks Fred!

Jim Boyle
MCI
2100 Reston Parkway
Reston, VA 20191
703.715.7006
jboyle@mci.net

Ron Cohen
Class Data Systems
13 Hasadna St.
Ra'anana 43650 Israel
972.9.7462020
ronc@classdata.com

Laura Cunningham
MCI
2100 Reston Parkway
Reston, VA 20191
703.715.7085
lcunning@mci.net

David Durham
Intel
2111 NE 25th Avenue
Hillsboro, OR 97124
503.264.6232
David_Durham@ccm.jf.intel.com

Arun Sastry
Cisco Systems
210 W Tasman Drive
San Jose, CA 95134
408.526.7685
asastry@cisco.com

Raj Yavatkar
Intel
2111 NE 25th Avenue
Hillsboro, OR 97124
503.264.9077
yavatkar@ibeam.intel.com