                      RSVP Operation Over IP Tunnels

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

   This document describes an approach for providing RSVP protocol
   services over IP tunnels. We briefly describe the problem, the
   characteristics of possible solutions, and the design goals of our
   approach. We then present the details of an implementation which
   meets our design goals.

1. Introduction

   IP-in-IP "tunnels" have become a widespread mechanism to transport
   datagrams in the Internet. Typically, a tunnel is used to route
   packets through portions of the network which do not directly
   implement the desired service (e.g. IPv6), or to augment and modify
   the behavior of the deployed routing architecture (e.g. multicast
   routing, mobile IP, Virtual Private Net).

   Many IP-in-IP tunneling protocols exist today. [IP4INIP4] details a
   method of tunneling using an additional IPv4 header. [MINENC]
   describes a way to reduce the size of the "inner" IP header used in
   [IP4INIP4] when the original datagram is not fragmented. The generic
   tunneling method in [IPV6GEN] can be used to tunnel either IPv4 or
   IPv6 packets within IPv6. [RFC1933] describes how to tunnel IPv6

Terzis, et al.   Standards Track   [Page 1]

datagrams through IPv4 networks. [RFC1701] describes a generic
routing encapsulation, while [RFC1702] applies this encapsulation to
IPv4. Finally, [ESP] describes a mechanism that can be used to
tunnel an encrypted IP datagram.

From the perspective of traditional best-effort IP packet delivery, a
tunnel behaves as any other link. Packets enter one end of the
tunnel, and are delivered to the other end unless resource overload
or error causes them to be lost.

The RSVP setup protocol [RFC2205] is one component of a framework
designed to extend IP to support multiple, controlled classes of
service over a wide variety of link-level technologies. To deploy
this technology with maximum flexibility, it is desirable for tunnels
to act as RSVP-controllable links within the network.

A tunnel, and in fact any sort of link, may participate in an RSVP-
aware network in one of three ways, depending on the capabilities of
the equipment from which the tunnel is constructed and the desires of
the operator.

   1. The (logical) link may not support resource reservation or QoS
      control at all. This is a best-effort link. We refer to this as
      a best-effort or type 1 tunnel in this note.
   2. The (logical) link may be able to promise that some overall
      level of resources is available to carry traffic, but not to
      allocate resources specifically to individual data flows. A
      configured resource allocation over a tunnel is an example of
      this. We refer to this case as a type 2 tunnel in this note.
   3. The (logical) link may be able to make reservations for
      individual end-to-end data flows. We refer to this case as a
      type 3 tunnel. Note that the key feature that distinguishes
      type 3 tunnels from type 2 tunnels is that in the type 3 tunnel
      new tunnel reservations are created and torn down dynamically
      as end-to-end reservations come and go.

Type 1 tunnels exist when at least one of the routers comprising the
tunnel endpoints does not support the scheme we describe here. In
this case, the tunnel acts as a best-effort link. Our goal is simply
to make sure that RSVP messages traverse the link correctly, and the
presence of the non-controlled link is detected, as required by the
integrated services framework.

When the two end points of the tunnel are capable of supporting RSVP
over tunnels, we would like to have proper resources reserved along
the tunnel. Depending on the requirements of the situation, this
might mean that one client's data flow is placed into a larger
aggregate reservation (type 2 tunnels) or that possibly a new,

separate reservation is made for the data flow (type 3 tunnels).
Note that an RSVP reservation between the two tunnel end points does
not necessarily mean that all the intermediate routers along the
tunnel path support RSVP, this is equivalent to the case of an
existing end-to-end RSVP session transparently passing through non-
RSVP cloud.

Currently, however, RSVP signaling over tunnels is not possible.
RSVP packets entering the tunnel are encapsulated with an outer IP
header that has a protocol number other than 46 (e.g. it is 4 for
IP-in-IP encapsulation) and do not carry the Router-Alert option,
making them virtually "invisible" to RSVP routers between the two
tunnel endpoints. Moreover, the current IP-in-IP encapsulation
scheme adds only an IP header as the external wrapper. It is
impossible to distinguish between packets that use reservations and
those that don't, or to differentiate packets belonging to different
RSVP Sessions while they are in the tunnel, because no distinguishing
information such as a UDP port is available in the encapsulation.

This document describes an IP tunneling enhancement mechanism that
allows RSVP to make reservations across all IP-in-IP tunnels. This
mechanism is capable of supporting both type 2 and type 3 tunnels, as
described above, and requires minimal changes to both RSVP and other
parts of the integrated services framework.

2. The Design

2.1. Design Goals

   Our design choices are motivated by several goals.

      * Co-existing with most, if not all, current IP-in-IP tunneling
        schemes.
      * Limiting the changes to the RSVP spec to the minimum possible.
      * Limiting the necessary changes to only the two end points of a
        tunnel. This requirement leads to simpler deployment, lower
        overhead in the intermediate routers, and less chance of failure
        when the set of intermediate routers is modified due to routing
        changes.
      * Supporting correct inter-operation with RSVP routers that have
        not been upgraded to handle RSVP over tunnels and with non-RSVP
        tunnel endpoint routers. In these cases, the tunnel behaves as a
        non-RSVP link.

2.2. Basic Approach

   The basic idea of the method described in this document is to
   recursively apply RSVP over the tunnel portion of the path. In this
   new session, the tunnel entry point Rentry sends PATH messages and
   the tunnel exit point Rexit sends RESV messages to reserve resources
   for the end-to-end sessions over the tunnel.

   We discuss next two different aspects of the design: how to enhance
   an IP-in-IP tunnel with RSVP capability, and how to map end-to-end
   RSVP sessions to a tunnel session.

2.2.1. Design Decisions

   To establish a RSVP reservation over a unicast IP-in-IP tunnel, we
   made the following design decisions:

   One or more Fixed-Filter style unicast reservations between the two
   end points of the tunnel will be used to reserve resources for
   packets traversing the tunnel. In the type 2 case, these reservations
   will be configured statically by a management interface. In the type
   3 case, these reservations will be created and torn down on demand,
   as end-to-end reservation requests come and go.

   Packets that do not require reservations are encapsulated in the
   normal way, e. g. being wrapped with an IP header only, specifying
   the tunnel entry point as source and the exit point as destination.

   Data packets that require resource reservations within a tunnel must
   have some attribute other than the IP addresses visible to the
   intermediate routers, so that the routers may map the packet to an
   appropriate reservation. To allow intermediate routers to use
   standard RSVP filterspec handling, we choose to encapsulate such data
   packets by prepending an IP and a UDP header, and to use UDP port
   numbers to distinguish packets of different RSVP sessions. The
   protocol number in the outer IP header in this case will be UDP.

   Figure 1 shows RSVP operating over a tunnel. Rentry is the tunnel
   entry router which encapsulates data into the tunnel. Some number of
   intermediate routers forward the data across the network based upon
   the encapsulating IP header added by Rentry. Rexit is the endpoint
   of the tunnel. It decapsulates the data and forwards it based upon
   the original, "inner" IP header.

```
      ...........                ..............            ..............
      :          :     _____   :          :    _____   :
      :          :    |       |  :          :   |     |  :
  Intranet  :--|  Rentry|==================|Rexit|___:Intranet
      :     |_____|  :          :   |_____|  :
      :          :            :  Internet  :            :
      ..........:            :..............
                          |_____|
```
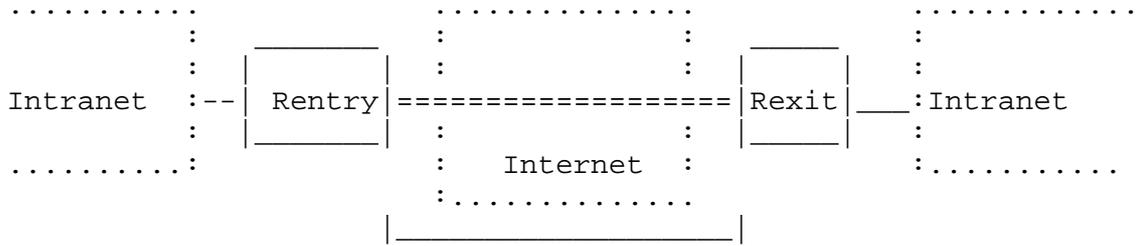
Figure 1.   An example IP Tunnel

2.2.2.   Mapping between End-to-End and Tunnel Sessions

   Figure 2 shows a simple topology with a tunnel and a few hosts. The
   sending hosts H1 and H3 may be one or multiple IP hops away from
   Rentry; the receiving hosts H2 and H4 may also be either one or
   multiple IP hops away from Rexit.

```
           H1                                             H2
            :                                              :
            :                                              :
       +--------+    +---+    +---+    +---+    +-------+
       |        |    |   |    |   |    |   |    |       |
  H3...  | Rentry |===================================| Rexit |..... H4
       |        |    |   |    |   |    |   |    |       |
       +--------+    +---+    +---+    +---+    +-------+
```
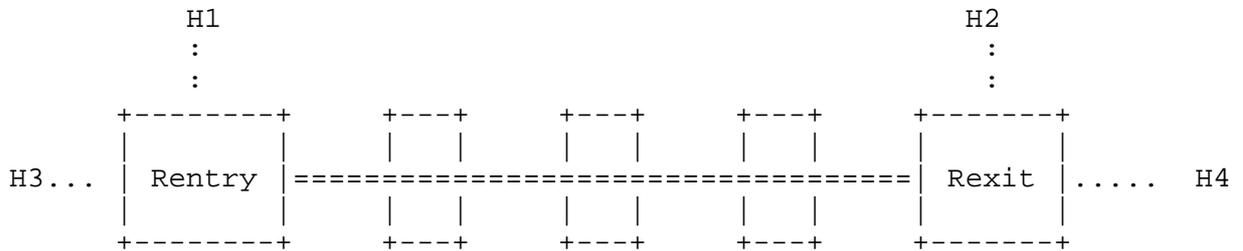
Figure 2: An example end-to-end path with
a tunnel in the middle.

   An RSVP session may be in place between endpoints at hosts H1 and H2.
   We refer to this session as the "end-to-end" (E2E for short) or
   "original" session, and to its PATH and RESV messages as the end-to-
   end messages. One or more RSVP sessions may be in place between
   Rentry and Rexit to provide resource reservation over the tunnel. We
   refer to these as the tunnel RSVP sessions, and to their PATH and
   RESV messages as the tunnel or tunneling messages. A tunnel RSVP
   session may exist independently from any end-to-end sessions. For
   example through network management interface one may create a RSVP
   session over the tunnel to provide QoS support for data flow from H3
   to H4, although there is no end-to-end RSVP session between H3 and
   H4.

   When an end-to-end RSVP session crosses a RSVP-capable tunnel, there
   are two cases to consider in designing mechanisms to support an end-
   to-end reservation over the tunnel: mapping the E2E session to an
   existing tunnel RSVP session (type 2 tunnel), and dynamically
   creating a new tunnel RSVP session for each end-to-end session (type

3 tunnel). In either case, the picture looks like a recursive
application of RSVP. The tunnel RSVP session views the two tunnel
endpoints as two end hosts with a unicast Fixed-Filter style
reservation in between. The original, end-to-end RSVP session views
the tunnel as a single (logical) link on the path between the
source(s) and destination(s).

Note that in practice a tunnel may combine type 2 and type 3
characteristics. Some end-to-end RSVP sessions may trigger the
creation of new tunnel sessions, while others may be mapped into an
existing tunnel RSVP session. The choice of how an end-to-end session
is treated at the tunnel is a matter of local policy.

When an end-to-end RSVP session crosses a RSVP-capable tunnel, it is
necessary to coordinate the actions of the two RSVP sessions, to
determine whether or when the tunnel RSVP session should be created
and torn down, and to correctly transfer error and ADSPEC information
between the two RSVP sessions. We made the following design
decision:

   * End-to-end RSVP control messages being forwarded through a
     tunnel are encapsulated in the same way as normal IP packets,
     e.g. being wrapped with the tunnel IP header only, specifying
     the tunnel entry point as source and the exit point as
     destination.

2.3. Major Issues

As IP-in-IP tunnels are being used more widely for network traffic
management purposes, it is clear we must support type 2 tunnels
(tunnel reservation for aggregate end-to-end sessions). Furthermore,
these type 2 tunnels should allow more than one (configurable,
static) reservation to be used at once, to support different traffic
classes within the tunnel. Whether it is necessary to support type 3
tunnels (dynamic per end-to-end session tunnel reservation) is a
policy issue that should be left open. Our design supports both
cases.

If there is only one RSVP session configured over a tunnel, then all
the end-to-end RSVP sessions (that are allowed to use this tunnel
session) will be bound to this configured tunnel session. However
when more than one RSVP session is in use over an IP tunnel, a second
design issue is how the association, or binding, between an original
RSVP reservation and a tunnel reservation is created and conveyed
from one end of the tunnel to the other. The entry router Rentry and
the exit router Rexit must agree on these associations so that

changes in the original reservation state can be correctly mapped
into changes in the tunnel reservation state, and that errors
reported by intermediate routers to the tunnel end points can be
correctly transformed into errors reported by the tunnel endpoints to
the end-to-end RSVP session.

We require that this same association mechanism work for both the
case of bundled reservation over a tunnel (type 2 tunnel), and the
case of one-to-one mapping between original and tunnel reservations
(type 3 tunnel). In our scheme the association is created when a
tunnel entry point first sees an end-to-end session's RESV message
and either sets up a new tunnel session, or adds to an existing
tunnel session. This new association must be conveyed to Rexit, so
that Rexit can reserve resources for the end-to-end sessions inside
the tunnel. This information includes the identifier and certain
parameters of the tunnel session, and the identifier of the end-to-
end session to which the tunnel session is being bound. In our
scheme, all RSVP sessions between the same two routers Rentry and
Rexit will have identical values for source IP address, destination
IP address, and destination UDP port number. An individual session is
identified primarily by the source port value.

We identified three possible choices for a binding mechanism:

   1. Define a new RSVP message that is exchanged only between two
      tunnel end points to convey the binding information.
   2. Define a new RSVP object to be attached to end-to-end PATH
      messages at Rentry, associating the end-to-end session with one
      of the tunnel sessions. This new object is interpreted by Rexit
      associating the end-to-end session with one of the tunnel
      sessions generated at Rentry.
   3. Apply the same UDP encapsulation to the end-to-end PATH
      messages as to data packets of the session. When Rexit
      decapsulates the PATH message, it deduces the relation between
      the source UDP port used in the encapsulation and the RSVP
      session that is specified in the original PATH message.

The last approach above does not require any new design. However it
requires additional resources to be reserved for PATH messages (since
they are now subject to the tunnel reservation). It also requires a
priori knowledge of whether Rexit supports RSVP over tunnels by UDP
encapsulation. If Rentry encapsulates all the end-to-end PATH
messages with the UDP encapsulation, but Rexit does not understand
this encapsulation, then the encapsulated PATH messages will be lost
at Rexit.

On the other hand, options (1) and (2) can handle this case
transparently. They allow Rexit to pass on end-to-end PATHs received
via the tunnel (because they are decapsulated normally), while
throwing away the tunnel PATHs, all without any additional
configuration. We chose Option (2) because it is simpler. We
describe this object in the following section.

Packet exchanges must follow the following constraints:

    1. Rentry encapsulates and sends end-to-end PATH messages over the
       tunnel to Rexit where they get decapsulated and forwarded
       downstream.
    2. When a corresponding end-to-end RESV message arrives at Rexit,
       Rexit encapsulates it and sends it to Rentry.
    3. Based on some or all of the information in the end-to-end PATH
       messages, the flowspec in the end-to-end RESV message and local
       policies, Rentry decides if and how to map the end-to-end
       session to a tunnel session.
    4. If the end-to-end session should be mapped to a tunnel session,
       Rentry either sends a PATH message for a new tunnel session or
       updates an existing one.
    5. Rentry sends a E2E Path containing a SESSION_ASSOC object
       associating the end-to-end session with the tunnel session
       above. Rexit records the association and removes the object
       before forwarding the Path message further.
    6. Rexit responds to the tunnel PATH message by sending a tunnel
       RESV message, reserving resources inside the tunnel.
    7. Rentry UDP-encapsulates arriving packets only if a
       corresponding tunnel session reservation is actually in place
       for the packets.

2.3.1. SESSION_ASSOC Object

   The new object, called SESSION_ASSOC, is defined with the following
   format:

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            length              |    class      |    c-type     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |         SESSION object  (for the end-to-end session)          |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |          Sender FILTER-SPEC (for the tunnel session)          |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
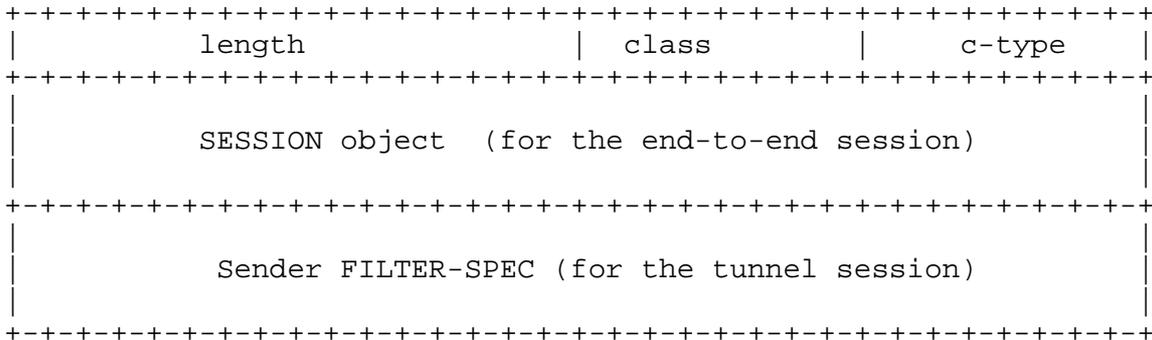
                        SESSION_ASSOC Object

   Length

      This field contains the size of the SESSION_ASSOC object in bytes.

   Class

      Should be 192.

   Ctype

      Should be sent as zero and ignored on receipt.

   SESSION object

      The end-to-end SESSION contained in the object is to be mapped to
      the tunnel session described by the Sender FILTER-SPEC defined
      below.

   Sender FILTER-SPEC

      This is the tunnel session that the above mentioned end-to-end
      session maps to over the tunnel. As we mentioned above, a tunnel
      session is identified primarily by source port. This is why we use
      a Sender Filter-Spec for the tunnel session, in the place of a
      SESSION object.

2.3.2. NODE_CHAR Object

   There has to be a way (other than through configuration) for Rexit to
   communicate to Rentry the fact that it is a tunnel endpoint
   supporting the scheme described in this document. We have defined for
   this reason a new object, called NODE_CHAR, carrying this
   information. If a node receives this object but does not understand
   it, it should drop it without producing any error report. Objects
   with Class-Num = 10bbbbbb ('b' represents a bit), as defined in the
   RSVP specification [RFC2205], have the characteristics we need. While
   for now this object only carries one bit of information, it can be
   used in the future to describe other characteristics of an RSVP
   capable node that are not part of the original RSVP specification.

   The object NODE_CHAR has the following format:

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            length             |     class     |    c-type     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Reserved                          |T|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
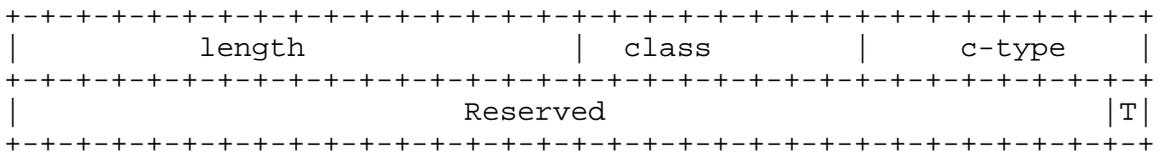
   Length

      This field contains the size of the NODE_CHAR object in bytes. It
      should be set to eight.

   Class

      An appropriate value should be assigned by the IANA. We propose
      this value to be 128.

   Ctype

      Should be sent as zero and ignored on receipt.

   T bit

      This bit shows that the node is a RSVP-tunnel capable node.

   When Rexit receives an end-to-end reservation, it appends a NODE_CHAR
   object with the T bit set, to the RESV object, it encapsulates it and
   sends it to Rentry. When Rentry receives this RESV message it deduces
   that Rexit implements the mechanism described here and so it creates
   or adjusts a tunnel session and associates the tunnel session to the
   end-to-end session via a SESSION_ASSOC object. Rentry should remove
   the NODE_CHAR object, before forwarding the RESV message upstream. If

on the other hand, Rentry does not support the RSVP Tunnels mechanism
it would simply ignore the NODE_CHAR object and not forward it
further upstream.

3. Implementation

   In this section we discuss several cases separately, starting from
   the simplest scenario and moving to the more complex ones.

3.1. Single Configured RSVP Session over an IP-in-IP Tunnel

   Treating the two tunnel endpoints as a source and destination host,
   one easily sets up a FF-style reservation in between. Now the
   question is what kind of filterspec to use for the tunnel
   reservation, which directly relates to how packets get encapsulated
   over the tunnel. We discuss two cases below.

3.1.1. In the Absence of End-to-End RSVP Session

   In the case where all the packets traversing a tunnel use the
   reserved resources, the current IP-in-IP encapsulation could be used.
   The RSVP session over the tunnel would simply specify a FF style
   reservation (with zero port number) with Rentry as the source address
   and Rexit as the destination address.

   However if only some of the packets traversing the tunnel should
   benefit from the reservation, we must encapsulate the qualified
   packets in IP and UDP. This allows intermediate routers to use
   standard RSVP filterspec handling, without having to know about the
   existence of tunnels.

   Rather than supporting both cases we choose to simplify
   implementations by requiring all data packets using reservations to
   be encapsulated with an outer IP and UDP header. This reduces special
   case checking and handling.

3.1.2. In the Presence of End-to-End RSVP Session(s)

   According to the tunnel control policies, installed through some
   management interface, some or all end-to-end RSVP sessions may be
   allowed to map to the single RSVP session over the tunnel. In this
   case there is no need to provide dynamic binding information between
   end-to-end sessions and the tunnel session, given that the tunnel
   session is unique and pre-configured, and therefore well-known.

   Binding multiple end-to-end sessions to one tunnel session, however,
   raises a new question of when and how the size of the tunnel
   reservation should be adjusted to accommodate the end-to-end sessions

mapped onto it. Again the tunnel manager makes such policy decision.
Several scenarios are possible. In the first, the tunnel reservation
is never adjusted. This makes the tunnel the rough equivalent of a
fixed-capacity hardware link. In the second, the tunnel reservation
is adjusted whenever a new end-to-end reservation arrives or an old
one is torn down. In the third, the tunnel reservation is adjusted
upwards or downwards occasionally, whenever the end-to-end
reservation level has changed enough to warrant the adjustment. This
trades off extra resource usage in the tunnel for reduced control
traffic and overhead.

We call a tunnel whose reservation cannot be adjusted a "hard pipe",
as opposed to a "soft pipe" where the amount of resources allocated
is adjustable. Section 5.2 explains how the adjustment can be carried
out for soft pipes.

3.2. Multiple Configured RSVP Sessions over an IP-in-IP Tunnel

It is straightforward to build on the case of a single configured
RSVP session over a tunnel by setting up multiple FF-style
reservations between the two tunnel endpoints using a management
interface. In this case Rentry must carefully encapsulate data
packets with the proper UDP port numbers, so that packets belonging
to different tunnel sessions will be distinguished by the
intermediate RSVP routers. Note that this case and the one described
before describe what we call type 2 tunnels.

3.2.1. In the Absence of End-to-End RSVP Session

Nothing more needs to be said in this case. Rentry classifies the
packets and encapsulates them accordingly. Packets with no
reservations are encapsulated with an outer IP header only, while
packets qualified for reservations are encapsulated with a UDP header
as well as an IP header. The UDP source port value should be properly
set to map to the corresponding tunnel reservation the packet is
supposed to use.

3.2.2. In the Presence of End-to-End RSVP Session(s)

Since in this case, there is more than one RSVP session operating
over the tunnel, one must explicitly bind each end-to-end RSVP
session to its corresponding tunnel session. As discussed
previously, this binding will be provided by the new SESSION_ASSOC
object carried by the end-to-end PATH messages.

3.3. Dynamically Created Tunnel RSVP Sessions

   This is the case of a type 3 tunnel. The only differences between
   this case and that of Section 4.2 are that:

      - The tunnel session is created when a new end-to-end session
        shows up.
      - There is a one-to-one mapping between the end-to-end and tunnel
        RSVP sessions, as opposed to possibly many-to-one mapping that
        is allowed in the case described in Section 4.2.

4. RSVP Messages handling over an IP-in-IP Tunnel

4.1. RSVP Messages for Configured Session(s) Over A Tunnel

   Here one or more RSVP sessions are set up over a tunnel through a
   management interface. The session reservation parameters never
   change for a "hard pipe" tunnel. The reservation parameters may
   change for a "soft pipe" tunnel. Tunnel session PATH messages
   generated by Rentry are addressed to Rexit, where they are processed
   and deleted.

4.2. Handling of RSVP Messages at Tunnel Endpoints

4.2.1. Handling End-to-End PATH Messages at Rentry

   When forwarding an end-to-end PATH message, a router acting as the
   tunnel entry point, Rentry, takes the following actions depending on
   the end-to-end session mentioned in the PATH message. There are two
   possible cases:

      1. The end-to-end PATH message is a refresh of a previously known
         end-to-end session.
      2. The end-to-end PATH message is from a new end-to-end session.

   If the PATH message is a refresh of a previously known end-to-end
   session, then Rentry refreshes the Path state of the end-to-end
   session and checks to see if this session is mapped to a tunnel
   session. If this is the case, then when Rentry refreshes the end-to-
   end session, it includes in the end-to-end PATH message a
   SESSION_ASSOC object linking this session to its corresponding tunnel
   session It then encapsulates the end-to-end PATH message and sends it
   over the tunnel to Rexit. If the tunnel session was dynamically
   created, the end-to-end PATH message serves as a refresh for the
   local tunnel state at Rentry as well as for the end-to-end session.

Otherwise, if the PATH message is from a new end-to-end session that
has not yet been mapped to a tunnel session, Rentry creates Path
state for this new session setting the outgoing interface to be the
tunnel interface. After that, Rentry encapsulates the PATH message
and sends it to Rexit without adding a SESSION_ASSOC message.

When an end-to-end PATH TEAR is received by Rentry, this node
encapsulates and forwards the message to Rexit. If this end-to-end
session has a one-to-one mapping to a tunnel session or if this is
the last one of the many end-to-end sessions mapping to a tunnel
session, Rentry tears down the tunnel session by sending a PATH TEAR
for that session to Rexit. If, on the other hand, there are remaining
end-to-end sessions mapping to the tunnel session, then Rentry sends
a tunnel PATH message adjusting the Tspec of the tunnel session.

4.2.2. Handling End-to-End PATH Messages at Rexit

Encapsulated end-to-end PATH messages are decapsulated and processed
at Rexit. Depending on whether the end-to-end PATH message contains a
SESSION_ASSOC object or not, Rexit takes the following steps:

   1. If the end-to-end PATH message does not contain a SESSION_ASSOC
      object, then Rentry sets the Non_RSVP flag at the Path state
      stored for this end-to-end sender, sets the global break bit in
      the ADSPEC and forwards the packets downstream. Alternatively,
      if tunnel sessions exist and none of them has the Non_RSVP flag
      set, Rexit can pick the worst-case Path ADSPEC params from the
      existing tunnel sessions and update the end-to-end ADSPEC using
      these values. This is a conservative estimation of the composed
      ADSPEC but it has the benefit of avoiding to set the break bit
      in the end-to-end ADSPEC before mapping information is
      available. In this case the Non_RSVP flag at the end-to-end
      Path state is not set.

   2. If the PATH message contains a SESSION_ASSOC object and no
      association for this end-to-end session already exists, then
      Rexit records the association between the end-to-end session
      and the tunnel session described by the object. If the end-to-
      end PATH arrives early before the tunnel PATH message arrives
      then it creates PATH state at Rexit for the tunnel session.
      When the actual PATH message for the tunnel session arrives it
      is treated as an update of the existing PATH state and it
      updates any information missing. We believe that this situation
      is another transient along with the others existing in RSVP and
      that it does not have any long-term effects on the correct
      operation of the mechanism described here.

Before further forwarding the message to the next hop along the
path to the destination, Rexit finds the corresponding tunnel
session's recorded state and turns on Non_RSVP flag in the
end-to-end Path state if the Non_RSVP bit was turned on for the
tunnel session. If the end-to-end PATH message carries an
ADSPEC object, Rexit performs composition of the
characterization parameters contained in the ADSPEC. It does
this by considering the tunnel session's overall (composed)
characterization parameters as the local parameters for the
logical link implemented by the tunnel, and composing these
parameters with those in the end-to-end ADSPEC by executing
each parameter's defined composition function. In the logical
link's characterization parameters, the minimum path latency
may take into account the encapsulation/decapsulation delay and
the bandwidth estimate can represent the decrease in available
bandwidth caused by the addition of the extra UDP header.
ADSPECs and composition functions are discussed in great detail
in [RFC2210].

If the end-to-end session has reservation state, while no
reservation state for the matching tunnel session exists, Rexit
send a tunnel RESV message to Rentry matching the reservation
in the end-to-end session.

If Rentry does not support RSVP tunneling, then Rexit will have no
PATH state for the tunnel. In this case Rexit simply turns on the
global break bit in the decapsulated end-to-end PATH message and
forwards it.

4.2.3. Handling End-to-End RESV Messages at Rexit

When forwarding a RESV message upstream, a router serving as the exit
router, Rexit, may discover that one of the upstream interfaces is a
tunnel. In this case the router performs a number of tests.

Step 1: Rexit must determine if there is a tunnel session bound to
the end-to-end session given in the RESV message. If not, the tunnel
is treated as a non-RSVP link, Rexit appends a NODE_CHAR object with
the T bit set, to the RESV message and forwards it over the tunnel
interface (where it is encapsulated as a normal IP datagram and
forwarded towards Rentry).

Step 2: If a bound tunnel session is found, Rexit checks to see if a
reservation is already in place for the tunnel session bound to the
end-to-end session given in the RESV message. If the arriving end-
to-end RESV message is a refresh of existing RESV state, then Rexit
sends the original RESV through tunnel interface (after adding the
NODE_CHAR object). For dynamic tunnel sessions, the end-to-end RESV

message acts as a refresh for the tunnel session reservation state,
while for configured tunnel sessions, reservation state never
expires.

If the arriving end-to-end RESV message causes a change in the end-
to-end RESV flowspec parameters, it may also trigger an attempt to
change the tunnel session's flowspec parameters. In this case Rexit
sends a tunnel session RESV, including a RESV_CONFIRM object.

In the case of a "hard pipe" tunnel, a new end-to-end reservation or
change in the level of resources requested by an existing reservation
may cause the total resource level needed by the end-to-end
reservations to exceed the level of resources reserved by the tunnel
reservation. This event should be treated as an admission control
failure, identically to the case where RSVP requests exceed the level
of resources available over a hardware link. A RESV_ERR message with
Error Code set to 01 (Admission Control failure), should be sent back
to the originator of the end-to-end RESV message.

If a RESV CONFIRM response arrives, the original RESV is encapsulated
and sent through the tunnel. If the updated tunnel reservation fails,
Rexit must send a RESV ERR to the originator of the end-to-end RESV
message, using the error code and value fields from the ERROR_SPEC
object of the received tunnel session RESV ERR message. Note that the
pre-existing reservations through the tunnel stay in place. Rexit
continues refreshing the tunnel RESV using the old flowspec.

Tunnel session state for a "soft pipe" may also be adjusted when an
end-to-end reservation is deleted. The tunnel session gets reduced
whenever one of the end-to-end sessions using the tunnel goes away
(or gets reduced itself). However even when the last end-to-end
session bound to that tunnel goes away, the configured tunnel session
remains active, perhaps with a configured minimal flowspec.

Note that it will often be appropriate to use some hysteresis in the
adjustment of the tunnel reservation parameters, rather than
adjusting the tunnel reservation up and down with each arriving or
departing end-to-end reservation. Doing this will require the tunnel
exit router to keep track of the resources allocated to the tunnel
(the tunnel flowspec) and the resources actually in use by end-to-end
reservations (the sum or statistical sum of the end-to-end
reservation flowspecs) separately.

When an end-to-end RESV TEAR is received by Rexit, it encapsulates
and forwards the message to Rentry. If the end-to-end session had
created a dynamic tunnel session, then a RESV TEAR for the
corresponding tunnel session is send by Rexit.

4.2.4. Handling of End-to-End RESV Messages at Rentry.

   If the RESV message received is a refresh of an existing reservation
   then Rentry updates the reservation state and forwards the message
   upstream. On the other hand, if this is the first RESV message for
   this end-to-end session and a NODE_CHAR object with the T bit set is
   present, Rentry should initiate the mapping between this end-to-end
   session and some (possibly new) tunnel session. This mapping is based
   on some or all of the contents of the end-to-end PATH message, the
   contents of the end-to-end RESV message, and local policies. For
   example, there could be different tunnel sessions based on the
   bandwidth or delay requirements of end-to-end sessions)

   If Rentry decides that this end-to-end session should be mapped to an
   existing configured tunnel session, it binds this end-to-end session
   to that tunnel session.

   If this end-to-end RSVP session is allowed to set up a new tunnel
   session, Rentry sets up tunnel session PATH state as if it were a
   source of data by starting to send tunnel-session PATH messages to
   Rexit, which is treated as the unicast destination of the data. The
   Tspec in this new PATH message is computed from the original PATH
   message by adjusting the Tspec parameters to include the tunnel
   overhead of the encapsulation of data packets. In this case Rentry
   should also send a PATH message from the end-to-end session this time
   containing the SESSION_ASSOC object linking the two sessions. The
   receipt of this PATH message by Rexit will trigger an update of the
   end-to-end Path state which in turn will have the effect of Rexit
   sending a tunnel RESV message, allocating resources inside the
   tunnel.

   The last case is when the end-to-end session is not allowed to use
   the tunnel resources. In this case no association is created between
   this end-to-end session and a tunnel session and no new tunnel
   session is created.

   One limitation of our scheme is that the first RESV message of an
   end-to-end session determines the mapping between that end-to-end
   session and its corresponding session over the tunnel. Moreover as
   long as the reservation is active this mapping cannot change.

5. Forwarding Data

   When data packets arrive at the tunnel entry point Rentry, Rentry
   must decide whether to forward the packets using the normal IP-in-IP
   tunnel encapsulation or the IP+UDP encapsulation expected by the
   tunnel session. This decision is made by determining whether there
   is a resource reservation (not just PATH state) actually in place for
   the tunnel session bound to the arriving packet, that is, whether the
   packet matches any active filterspec.

   If a reservation is in place, it means that both Rentry and Rexit are
   RSVP-tunneling aware routers, and the data will be correctly
   decapsulated at Rexit.

   If no tunnel session reservation is in place, the data should be
   encapsulated in the tunnel's normal format, regardless of whether
   end-to-end PATH state covering the data is present.

6. Details

6.1. Selecting UDP port numbers

   There may be multiple end-to-end RSVP sessions between the two end
   points Rentry and Rexit. These sessions are distinguished by the
   source UDP port. Other components of the session ID, the source and
   destination IP addresses and the destination UDP port, are identical
   for all such sessions.

   The source UDP port is chosen by the tunnel entry point Rentry when
   it establishes the initial PATH state for a new tunnel session. The
   source UDP port associated with the new session is then conveyed to
   Rexit by the SESSION_ASSOC object.

   The destination UDP port used in tunnel sessions should the one
   assigned by IANA (363).

6.2. Error Reporting

   When a tunnel session PATH message encounters an error, it is
   reported back to Rentry. Rentry must relay the error report back to
   the original source of the end-to-end session.

   When a tunnel session RESV request fails, an error message is
   returned to Rexit. Rexit must treat this as an error in crossing the
   logical link (the tunnel) and forward the error message back to the
   end host.

6.3. MTU Discovery

   Since the UDP encapsulated packets should not be fragmented, tunnel
   entry routers must support tunnel MTU discovery as discussed in
   section 5.1 of [IP4INIP4]. Alternatively, the Path MTU Discovery
   mechanism discussed in RFC 2210 [RFC2210] can be used.

6.4. Tspec and Flowspec Calculations

   As multiple End-to-End sessions can be mapped to a single tunnel
   session, there is the need to compute the aggregate Tspec of all the
   senders of those End-to-End sessions. This aggregate Tspec will the
   Tspec of the representative tunnel session. The same operation needs
   to be performed for flowspecs of End-to-End reservations arriving at
   Rexit.

   The semantics of these operations are not addressed here. The
   simplest way to do them is to compute a sum of the end-to-end Tspecs,
   as is defined in the specifications of the Controlled-Load and
   Guaranteed services (found at [RFC2211] and [RFC2212] respectively).
   However, it may also be appropriate to compute the aggregate
   reservation level for the tunnel using a more sophisticated
   statistical or measurement-based computation.

7. IPSEC Tunnels

   In the case where the IP-in-IP tunnel supports IPSEC (especially ESP
   in Tunnel-Mode with or without AH) then the Tunnel Session uses the
   GPI SESSION and GPI SENDER_TEMPLATE/FILTER_SPEC as defined in
   [RSVPESP] for the PATH and RESV messages.

   Data packets are not encapsulated with a UDP header since the SPI can
   be used by the intermediate nodes for classification purposes.
   Notice that user oriented keying must be used between Rentry and
   Rexit, so that different SPIs are assigned to data packets that have
   reservation and "best effort" packets, as well as packets that belong
   to different Tunnel Sessions if those are supported.

8. RSVP Support for Multicast and Multipoint Tunnels

   The mechanisms described above are useful for unicast tunnels.
   Unicast tunnels provide logical point-to-point links in the IP
   infrastructure, though they may encapsulate and carry either unicast
   or multicast traffic between those points.

Two other types of tunnels may be imagined. The first of these is a
"multicast" tunnel. In this type of tunnel, packets arriving at an
entry point are encapsulated and transported (multicast) to -all- of
the exit points. This sort of tunnel might prove useful for
implementing a hierarchical multicast distribution network, or for
emulating efficiently some portion of a native multicast distribution
tree.

A second possible type of tunnel is the "multipoint" tunnel. In this
type of tunnel, packets arriving at an entry point are normally
encapsulated and transported to -one- of the exit points, according
to some route selection algorithm.

This type of tunnel differs from all previous types in that the '
shape' of the usual data distribution path does not match the 'shape'
of the tunnel. The topology of the tunnel does not by itself define
the data transmission function that the tunnel performs. Instead,
the tunnel becomes a way to express some shared property of the set
of connected tunnel endpoints. For example, the "tunnel" may be used
to create and embed a logical shared broadcast network within some
larger network. In this case the tunnel endpoints are the nodes
connected to the logical shared broadcast network. Data traffic may
be unicast between two such nodes, broadcast to all connected nodes,
or multicast between some subset of the connected nodes. The tunnel
itself is used to define a domain in which to manage routing and
resource management - essentially a virtual private network.

Note that while a VPN of this form can always be implemented using a
multicast tunnel to emulate the broadcast medium, this approach will
be very inefficient in the case of wide area VPNs, and a multipoint
tunnel with appropriate control mechanisms will be preferable.

The following paragraphs provide some brief commentary on the use of
RSVP in these situations. Future versions of this note will provide
more concrete details and specifications.

Using RSVP to provide resource management over a multicast tunnel is
relatively straightforward. As in the unicast case, one or more RSVP
sessions may be used, and end-to-end RSVP sessions may be mapped onto
tunnel RSVP sessions on a many-to-one or one-to-one basis. Unlike the
unicast, case, however, the mapping is complicated by RSVP's
heterogeneity semantics. If different receivers have made different
reservation requests, it may be that the RESV messages arriving at
the tunnel would logically map the receiver's requests to different
tunnel sessions. Since the data can actually be placed into only one
session, the choice of session must be reconciled (merged) to select
the one that will meet the needs of all applications. This requires a
relatively simple extension to the session mapping mechanism.

Use of RSVP to support multipoint tunnels is somewhat more difficult.
In this case, the goal is to give the tunnel as a whole a specific
level of resources. For example, we may wish to emulate a "logical
shared 10 megabit Ethernet" rather than a "logical shared Ethernet".
However, the problem is complicated by the fact that in this type of
tunnel the data does not always go to all tunnel endpoints. This
implies that we cannot use the destination address of the
encapsulated packets as part of the packet classification filter,
because the destination address will vary for different packets
within the tunnel.

This implies the need for an extension to current RSVP session
semantics in which the Session ID (destination IP address) is used
-only- to identify the session state within network nodes, but is not
used to classify packets. Other than this, the use of RSVP for
multipoint tunnels follows that of multicast tunnels. A multicast
group is created to represent the set of nodes that are tunnel
endpoints, and one or more tunnel RSVP sessions are created to
reserve resources for the encapsulated packets. In the case of a
tunnel implementing a simple VPN, it is most likely that there will
be one session to reserve resources for the whole VPN. Each tunnel
endpoint will participate both as a source of PATH messages and a
source of (FF or SE) RESV messages for this single session,
effectively creating a single shared reservation for the entire
logical shared medium. Tunnel endpoints MUST NOT make wildcard
reservations over multipoint tunnels.

9. Extensions to the RSVP/Routing Interface

The RSVP specification [RFC2205] states that through the RSVP/Routing
Interface, the RSVP daemon must be able to learn the list of local
interfaces along with their IP addresses. In the RSVP Tunnels case,
the RSVP daemon needs also to learn which of the local interface(s)
is (are) IP-in-IP tunnel(s) having the capabilities described here.
The RSVP daemon can acquire this information, either by directly
querying the underlying network and physical layers or by using any
existing interface between RSVP and the routing protocol properly
extended to provide this information.

10. Security Considerations

   The introduction of RSVP Tunnels raises no new security issues other
   than those associated with the use of RSVP and tunnels. Regarding
   RSVP, the major issue is the need to control and authenticate access
   to enhanced qualities of service. This requirement is discussed
   further in [RFC2205]. [RSVPCRYPTO] describes the mechanism used to
   protect the integrity of RSVP messages carrying the information
   described here. The security issues associated with IP-in-IP tunnels
   are discussed in [IPINIP4] and [IPV6GEN].

11. IANA Considerations

   IANA should assign a Class number for the NODE_CHAR object defined in
   Section 3.3.2. This number should be in the 10bbbbbb range. The
   suggested value is 128.

12. Acknowledgments

   We thank Bob Braden for his insightful comments that helped us to
   produce this updated version of the document.

13. References

   [ESP]   Atkinson, R., "IP Encapsulating Security Payload (ESP)",
                 RFC 1827, August 1995.

   [IP4INIP4]   Perkins, C., "IP Encapsulation within IP", RFC 2003,
                 October 1996.

   [IPV6GEN]   Conta, A. and S. Deering, "Generic Packet Tunneling in
                 IPv6 Specification", RFC 2473, December 1998.

   [MINENC]   Perkins, C., "Minimal Encapsulation within IP", RFC
                 2004, October 1996.

   [RFC1701]   Hanks, S., Li, T., Farinacci, D. and P. Traina, "Generic
                 Routing Encapsulation (GRE)", RFC 1701, October 1994.

   [RFC1702]   Hanks, S., Li, T., Farinacci, D. and P. Traina, "Generic
                 Routing Encapsulation over IPv4 Networks", RFC 1702,
                 October 1994.

   [RFC1933]   Gilligan, R. and E. Nordmark, "Transition Mechanisms for
                 IPv6 Hosts and Routers", RFC 1933, April 1996.

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

   [RFC2211]   Wroclawski, J., "Specification of the Controlled-Load
                   Network Element Service", RFC 2211, September 1997.

   [RFC2212]   Shenker, S., Partridge, C. and R. Guerin, "Specification
                   of the Guaranteed Quality of Service", RFC 2212,
                   September 1997.

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

   [RSVPESP]   Berger, L. and T. O'Malley, "RSVP Extensions for IPSEC
                   Data Flows", RFC 2207, September 1997.

   [RSVPCRYPTO] Baker, F., Lindell, B. and M. Talwar, "RSVP
                   Cryptographic Authentication", RFC 2747, January 2000.

14. Authors' Addresses

   John Krawczyk
   ArrowPoint Communications
   50 Nagog Park
   Acton, MA 01720

   Phone: 978-206-3027
   EMail: jj@arrowpoint.com

   John Wroclawski
   MIT Laboratory for Computer Science
   545 Technology Sq.
   Cambridge, MA 02139

   Phone: 617-253-7885
   Fax: 617-253-2673
   EMail: jtw@lcs.mit.edu

   Lixia Zhang
   UCLA
   4531G Boelter Hall
   Los Angeles, CA 90095

   Phone: 310-825-2695
   EMail: lixia@cs.ucla.edu

   Andreas Terzis
   UCLA
   4677 Boelter Hall
   Los Angeles, CA 90095

   Phone: 310-267-2190
   EMail: terzis@cs.ucla.edu

15. Full Copyright Statement

Acknowledgement