IETF Next Steps in Signaling
Internet-Draft
Expires: September 7, 2006

C. Shen
H. Schulzrinne
Columbia U.
S. Lee
J. Bang
Samsung AIT

March 6, 2006

NSIS Operation Over IP Tunnels
draft-shen-nsis-tunnel-02.txt

Abstract

This draft presents an NSIS operation over IP tunnels scheme using QoS NSLP as the NSIS signaling application. Both sender-initiated and receiver-initiated reservation modes are discussed. The scheme proposes the use of separate signaling sessions inside the tunnel for the end-to-end sessions. Packets belonging to qualified tunnel

sessions are assigned special flow IDs to be distinguished from the rest of the tunnel traffic. The end-to-end session and its corresponding tunnel session are associated with each other when necessary; so that adjustment in one session may be reflected in the other.

Table of Contents

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [1].

2. Introduction

IP tunnel mechanisms are widely used in the Internet for various purposes. When a tunnel is used to transfer signaling messages, e.g. NSIS messages, the signaling messages themselves usually become invisible inside the tunnel. In other words, the tunnel behaves as a logical link that does not support signaling in the end-to-end path. If end-to-end NSIS signaling support is desired for a path containing tunnels, it is necessary to define a scheme that allows NSIS operation over IP tunnels. This draft describes such a scheme. We assume QoS NSLP as the NSIS signaling application.

2.1. Background

2.1.1. IP Tunneling Mechanisms

There are a number of common tunneling mechanisms used in the Internet. A non-exhausted list of them is as follows,

- o Generic Routing Encapsulation (GRE) [4] is a mechanism for encapsulating arbitrary packets within an arbitrary transport protocol. Generic Routing Encapsulation over IPv4 Networks (GREIP4) [5] addresses the case of using IPv4 as the delivery protocol or the payload protocol and the special case of IPv4 as both the delivery and payload. Generic Routing Encapsulation (GREIP4A) [17] presented a modified version of [4], in particular, some flag bits in the original specification have been deprecated.
- o IP Encapsulation within IP (IP4INIP4) [7] is a method of tunneling IPv4 packets using an additional IPv4 header. Minimal Encapsulation within IP (MINENC) [8] describes a way to reduce the size of the "inner" IP header used in [7] when the original datagram is not fragmented.
- o Generic Packet Tunneling in IPv6 Specification (IP6GEN) [11] specifies a method by which a packet is carried as payload within an IPv6 packet by being encapsulated in an IPv6 header, and optionally, a set of IPv6 extension headers.
- o IPv6 over IPv4 tunneling (IP6INIP4) [6] encapsulates IPv6 packets within IPv4 headers to carry them over IPv4 routing infrastructures.
- o IPSEC [9] has a tunnel mode with the use of Encapsulating Security Payload (ESP) [10]. The tunneled IP packets are encrypted and the

ESP is placed before the encapsulated IP header.

The above tunneling mechanisms fall into two broad categories according to the encapsulating (delivery) header format:

1. Normal IP in IP Encapsulation: the encapsulating header is just a standard IP header. This group includes IP4INIP4, IP6INIP4, IP6GEN.
2. Modified IP in IP Encapsulation: the encapsulating header is a standard IP header plus additional information. This group includes all GRE-related IP tunneling, MINENC and IPSEC tunneling mode. The additional information in these cases is the GRE header, minimum encapsulation header and ESP header respectively. This information is usually placed between the encapsulating IP header and the original IP header. (MINENC is an exception because it modifies the original IP header). Note that in the IPSEC case, the original IP header is also encrypted along with the original IP payload.

2.1.2. Different Signaling Capability of IP Tunnels

By default any end-to-end signaling messages arriving at the tunnel endpoint will be encapsulated the same way as data packets. Tunnel intermediate nodes do not identify them as signaling messages. Therefore the tunnel appears as a signaling unaware logical link to the end-to-end session.

A signaling aware tunnel may participate in a signaling network in various ways. For example, [18] identifies two types of QoS aware tunnels: a tunnel that can promise that some overall level of resources is available to carry traffic, but not to allocate resources specifically to individual data flows; or a tunnel that can make reservations for individual end-to-end data flows. An individual tunnel signaling session may be created and torn down dynamically as end-to-end session come and go. An aggregate tunnel sessions could be a pre-configured session that never gets changed, or could be dynamically adjusted as the actually used session resources increase or decrease.

A tunnel may also be a mixed one that combines the properties of both types of the tunnels.

2.2. NSIS Tunnel Operation Overview

This document presents a scheme to enable NSIS operation over IP tunnels with different tunnel capabilities. The design goals of the scheme are as follows,

- o For best effort tunnel, make sure NSIS messages traverse the link correctly, and the presence of the non-NSIS aware link is detected.
- o For signaling aware tunnels, make sure proper signaling is carried out when necessary, to set up the tunnel sessions for use by the end-to-end sessions.
- o Work with most, if not all, existing IP in IP tunneling schemes.
- o Place the specific tunnel related functionalities only in one or both of the tunnel endpoints.

The overall design of NSIS operation over IP tunnels is conceptually similar to RSVP operation over IP tunnels [18]. (A short summary of [18] is provided in appendix Section 9.1). However, the scheme described in this document also addresses the important differences of NSIS from RSVP, e.g.,

- o NSIS is based on a two-layer architecture, namely a signaling transport layer and a signaling application layer. It is designed as a generic framework to accommodate various signaling application needs. The basic RSVP protocol does not have a layer split and is only for QoS signaling.
- o NSIS QoS NSLP allows both sender-initiated and receiver-initiated reservations; RSVP only supports receiver-initiated reservations.
- o NSIS deals only with unicast; RSVP also supports multicast.
- o NSIS integrates new features, such as the Session ID, to facilitate operation in specific environments (e.g. mobility and multi-homing).

From a high level point of view, the main issues in a signaling operation over IP tunnel scheme are, how packet classification is performed inside the tunnel, and how signaling is carried out inside the tunnel.

Packets belonging to qualified data flows need to be recognized by tunnel intermediate nodes to receive special treatment. Packet classification is traditionally based on flow ID, which is derived from various fields in Message Routing Information (MRI). The problem is, after a typical IP-in-IP tunnel encapsulation, all packets going through the tunnel appear as having the same flow ID (which consists of the Tunnel Entry (Tentry) address and Tunnel Exit (Texit) address. Therefore the flow ID for signaled flows needs to contain further demultiplexing fields in order to be distinguished from non-signaled flows, and also from one another among all signaled flows.

The special flow ID for signaled flows inside the tunnel then needs to be carried in tunnel signaling messages to set up or modify the state information in tunnel intermediate nodes. The original end-to-

end signaling messages do not contain tunnel specific parameters such as the tunnel flow ID and tunnel adjusted QoS parameters. Therefore, separate tunnel signaling sessions are generated and maintained between the tunnel endpoints, as in the case of RSVP operation over IP tunnels [18]. When end-to-end signaling sessions and tunnel signaling sessions are carried out separately, it will be necessary in many cases to maintain the state association between the end-to-end session and its corresponding tunnel session so that any change to one session may be reflected in the other.

In the next section, we will illustrate details on packet classification over the tunnel, signaling over the tunnel as well as association of end-to-end and tunnel signaling.

3. Protocol Design Decisions

3.1. Packet Classification over the Tunnel

Tunnel flows need to be assigned special flow IDs in order to allow tunnel packet classification. A flow can be an individual flow or an aggregate flow. Possible flow ID formats that may be used to identify individual tunnel flows are grouped below:

o Selected fields from the base IP header of the tunnel encapsulated packet (outer IP header). For example, the IP source and destination address fields, which contain the IP addresses of Tentry and Texit, together with another field for tunnel-wide demultiplexing. This could be the IPv6 flow label field, or the Traffic Class (TC) field. Note that the TC field can also be used in DiffServ to carry DiffServ Code Point (DSCP) and thus represent an aggregate flow. As long as individual flow classification is processed before aggregate flow classification, or a longest match kind of packet classifier is used, the tunnel flow demultiplexing with TC field should work. In the rare cases where these conditions could not be satisfied, it is still possible to choose different range of DSCP values so that the values used for individual tunnel flow demultiplexing do not collide with those used for DiffServ aggregate flows. Compared to the IPv6 flow label approach, the tunnel flow ID containing DSCP can be applied to both IPv4 and IPv6 and is probably easier to deploy. Its drawback is that the small number of bits in the DSCP field limits the total number of individual flows that can be distinguished in the tunnel. Overall, these flow ID formats in this group enable efficient packet classification over the tunnel without introducing additional processing requirements on the existing infrastructure. They are also easy to deploy.

- Selected fields from the tunnel base IP header plus additional information outside the base IP header but still in the tunnel encapsulation header. This applies to modified IP-in-IP encapsulation as we defined in Section 2.1.1. An example of this additional information is the SPI field for IPSEC tunnels. Comparing with the first group, the flow ID formats in this group poses more requirements at the NSIS protocol side because it uses information unique to the specific tunnel mechanism. NSIS thus needs to be specifically tuned to recognize that information as part of a signaling message. This is similar to how [19] has extended RSVP to accommodate IPSEC.

- UDP header insertion. Inserting a new UDP header between the tunnel IP header and the tunnel payload provides additional demultiplexing information for a tunnel flow. The drawback of the flow ID format in this group, as compared to the above two, is the additional UDP header overhead both for bandwidth and processing. In addition, this approach modifies the basic tunneling mechanism at the Tentry, so Texit will also need to be aware of the special encapsulation in order to correctly decapsulate and forward packets further along the path.

Most of the above flow ID formats may also be used for aggregate tunnel flows. For example, a common aggregate flow ID contains the addresses of tunnel endpoints and the DSCP value. When additional interfaces at the tunnel endpoints are available, these addresses may also be used to form aggregate flow ID. For example, the IP address of an additional interface for a Tentry plus the IP address of the Texit, constitute an aggregate flow ID.

The choice of using which of the above flow ID format is left to a policy mechanism outside the scope of this document. Tunnel signaling is performed based on the chosen flow ID and Tentry should encapsulate all incoming packets for the specific data flows according to the chosen flow ID format.

3.2. Tunnel Signaling and its Association with End-to-end Signaling

Tunnel signaling messages contain tunnel specific parameters such as tunnel MRI and tunnel adjusted QoS parameters. But the formats of tunnel signaling messages are the same as end-to-end signaling messages and tunnel signaling is carried out according to the same signaling flows of the end-to-end signaling. The main challenge is therefore the interaction between tunnel signaling and end-to-end signaling. The interaction is achieved by special functionalities supported in the NSIS-aware tunnel endpoints. These special functionalities include assigning tunnel flow IDs, creating tunnel

session association, notifying the other endpoint about tunnel association, adjusting one session based on change of the other session, encapsulating (decapsulating) packets according to the chosen tunnel flow ID at Tentry (Texit), etc. In most cases, we expect to have bi-directional tunnels, where both endpoints are NSIS-tunnel aware.

When both Tentry and Texit are NSIS-tunnel aware, the endpoint that creates the tunnel session may need to notify the other endpoint of the association between the end-to-end and tunnel session. This is achieved by using the QoS NSLP BOUND_SESSION_ID object with a binding code indicating this binding is for tunnel handling. In the rest of this document, we refer to a BOUND_SESSION_ID object with the tunnel binding_code as a tunnel BOUND_SESSION_ID object or a tunnel binding object. The tunnel binding object is carried in the end-to-end signaling messages with the session ID of the corresponding tunnel session. The NSIS-tunnel aware endpoints that receive this tunnel BOUND_SESSION_ID object should perform tunnel related procedures and then remove it for any end-to-end signaling messages to be sent out of the tunnel.

3.3. Tunnel Signaling Capability Discovery

Tunnel signaling may only be initiated when both Tentry and Texit are NSIS-tunnel aware. When prior knowledge of the other endpoint's NSIS tunnel capability is not available, we need a discovery mechanism to find it out. This mechanism is expected to be the responsibility of the NSLP layer. One option is to define a ''Tunnel Capable'' bit in the INFO_SPEC object of its informational class and exchange it between the Tentry and Texit. More details will be provided in the next version of this document. The messaging flow diagrams in the current document assume that the tunnel capability discovery has already been made.

4. Protocol Operation with Dynamically Created Tunnel Sessions

4.1. Operation Scenarios

To dynamically create a mapping tunnel session upon receiving an end-to-end session, we identify four scenarios based on the sender-initiated and receiver-initiated reservation modes of NSIS QoS NSLP:

o A. End-to-end session is sender-initiated; tunnel session is sender-initiated.
o B. End-to-end session is receiver-initiated; tunnel session is receiver-initiated.
o C. End-to-end session is sender-initiated; tunnel session is

receiver-initiated.

o D. End-to-end session is receiver-initiated; tunnel session is sender-initiated.

In the following we present a typical NSIS end-to-end and tunnel signaling interaction during the tunnel set up phase in each of these four scenarios. The end-to-end QoS flow is assumed to be one that qualifies an individual dynamic tunnel session, whose reservation must be confirmed.

It should be noted that different flow requirements and policy assumptions may cause the timing sequence of the messaging flow to be slightly different, which will be discussed in Section 4.2.

Once the tunnel session has been created and associated with the end-to-end session, any subsequent changes (modification or termination) to either session may be communicated to the other one by the binding endpoint so the state of the two binding sessions may keep consistent. The exception is when the tunnel session is an aggregate session. In this case, after setup, the adjustment of the tunnel session should follow the rules for pre-configured aggregate tunnel adjustment in Section 5.

4.1.1. Sender-initiated Reservation for both End-to-end and Tunnel Signaling

```
     Sender     Tentry      Tnode      Texit     Receiver

       |          |          |          |          |
       | RESERVE  |          |          |          |
       +--------->|          |          |          |
       |          | RESERVE' |          |          |
       |          +=========>|          |          |
       |          |          | RESERVE' |          |
       |          |          +=========>|          |
       |          |       RESERVE       |          |
       |          +-------------------->|          |
       |          |          | RESPONSE'| RESERVE  |
       |          |          |<=========+--------->|
       |          | RESPONSE'|          |          |
       |          |<=========+          |          |
       |          |          |          | RESPONSE |
       |          |          |          |<---------+
       |          |       RESPONSE      |          |
       |          |<--------------------+          |
       | RESPONSE |          |          |          |
```

```
       |<---------+          |          |          |
       |          |          |          |          |
       |          |          |          |          |
```

Figure 1: Sender-initiated Reservation for both End-to-end and Tunnel Signaling

This scenario assumes both end-to-end and tunnel sessions are sender-initiated. Figure 1 shows the messaging flow of NSIS operation over IP tunnels in this case. Tunnel signaling messages are distinguished from end-to-end messages by a "'" after the message name. Tnode denotes an intermediate tunnel node that participates in tunnel signaling. The sender first sends an end-to-end RESERVE message which arrives at Tentry. If Tentry supports tunnel signaling and determines that an individual tunnel session needs to be established for the end-to-end session, it chooses the tunnel flow ID, creates the tunnel session and associates the end-to-end session with the tunnel session. It then sends a tunnel RESERVE' message matching the requests of the end-to-end session toward the Texit to reserve tunnel resources. Tentry also appends to the original RESERVE message a tunnel BOUND_SESSION_ID object containing the session ID of the tunnel session and sends it toward Texit using normal tunnel encapsulation.

The tunnel RESERVE' message is processed hop by hop inside the tunnel for the flow identified by the chosen tunnel flow ID. When Texit receives the tunnel RESERVE' message, reservation state for the tunnel session will be created. Texit may also send a tunnel RESPONSE' message to Tentry. On the other hand, the end-to-end RESERVE message passes through the tunnel intermediate nodes just like any other tunneled packets. When Texit receives the end-to-end RESERVE message, it notices the binding of a tunnel session and checks the state for the tunnel session. When the tunnel session state is available, it updates the end-to-end reservation state using the tunnel session state, removes the tunnel BOUND_SESSION_ID object and forwards the end-to-end RESERVE message further along the path towards the receiver. When the end-to-end reservation finishes, an end-to-end RESPONSE may be sent back from the receiver to the sender.

4.1.2. Receiver-initiated Reservation for both End-to-end and Tunnel Signaling

```
     Sender    Tentry      Tnode      Texit      Receiver
```

```
       |          |          |          |          |
       |  QUERY   |          |          |          |
       +--------->|          |          |          |
       |          |        QUERY        |          |
       |          +-------------------->|          |
       |          |  QUERY'  |          |          |
       |          +=========>|          |          |
       |          |          |  QUERY'  |          |
       |          |          +=========>|          |
       |          |          |          |  QUERY   |
       |          |          |          +--------->|
       |          |          |          | RESERVE  |
       |          |          |          |<---------+
       |          |       RESERVE       |          |
       |          |<--------------------+          |
       |          |          | RESERVE' |          |
       |          |          |<=========+          |
       |          | RESERVE' |          |          |
       |          |<=========+          |          |
       |  RESERVE | RESPONSE'|          |          |
       |<---------+=========>|          |          |
       |          |          | RESPONSE'|          |
       |          |          +=========>|          |
       | RESPONSE |          |          |          |
       +--------->|          |          |          |
       |          |       RESPONSE      |          |
       |          +-------------------->|          |
       |          |          |          | RESPONSE |
       |          |          |          +--------->|
       |          |          |          |          |
       |          |          |          |          |
```

Figure 2: Receiver-initiated Reservation for both End-to-end and Tunnel Signaling

This scenario assumes both end-to-end and tunnel sessions are receiver-initiated. Figure 2 shows the messaging flow of NSIS operation over IP tunnels in this case. When Tentry receives the first end-to-end QUERY message from the sender, it chooses the tunnel flow ID, creates the tunnel session and sends a tunnel QUERY' message matching the requests of the end-to-end session toward the Texit. Tentry also appends to the original QUERY message with a tunnel BOUND_SESSION_ID object containing the session ID of the tunnel session and sends it toward the Texit using normal tunnel encapsulation.

The tunnel QUERY' message is processed hop by hop inside the tunnel

for the flow identified by the chosen tunnel flow ID. When Texit receives the tunnel QUERY' message, it creates a reservation state for the tunnel session without sending out a tunnel RESERVE' message immediately.

The end-to-end QUERY message passes along tunnel intermediate nodes just like any other tunneled packets. When Texit receives the end-to-end QUERY message, it notices the binding of a tunnel session and checks state for the tunnel session. When the tunnel session state is available, Texit updates the end-to-end QUERY message using the tunnel session state, removes the tunnel BOUND_SESSION_ID object and forwards the end-to-end QUERY message further along the path.

When Texit receives the first end-to-end RESERVE message issued by the receiver, it finds the reservation state of the tunnel session and triggers a tunnel RESERVE' message for that session. Meanwhile the end-to-end RESERVE message will be appended with a tunnel BOUND_SESSION_ID object and forwarded towards Tentry. When Tentry receives the tunnel RESERVE', it creates the reservation state for the tunnel session and may send a tunnel RESPONSE' back to Texit. When Tentry receives the end-to-end RESERVE, it creates the end-to-end reservation state and updates it with information from the associated tunnel session reservation state. Then Tentry further forwards the end-to-end RESERVE upstream toward the sender.

4.1.3. Sender-initiated Reservation for End-to-end and Receiver-initiated Reservation for Tunnel Signaling

```
     Sender    Tentry      Tnode      Texit     Receiver
     |          |          |          |          |
     | RESERVE  |          |          |          |
     +--------->|          |          |          |
     |          |  QUERY'  |          |          |
     |          +=========>|          |          |
     |          |          |  QUERY'  |          |
     |          |          +=========>|          |
     |          |          | RESERVE' |          |
     |          |          |<=========+          |
     |          | RESERVE' |          |          |
     |          |<=========+          |          |
     |          | RESPONSE'|          |          |
     |          +=========>|          |          |
     |          |          | RESPONSE'|          |
     |          |          +=========>|          |
     |          |        RESERVE      |          |
     |          +-------------------->|          |
     |          |          |          | RESERVE  |
```

```
     |          |          |          +--------->|
     |          |          |          | RESPONSE |
     |          |          |          |<---------+
     |          |       RESPONSE      |          |
     |          |<--------------------+          |
     | RESPONSE |          |          |          |
     |<---------+          |          |          |
     |          |          |          |          |
     |          |          |          |          |
```

Figure 3: Sender-initiated Reservation for End-to-end and Receiver-initiated Reservation for Tunnel Signaling

This scenario assumes the end-to-end signaling mode is sender-initiated and the tunnel signaling mode is receiver-initiated. Figure 3 shows the messaging flow of NSIS operation over IP tunnels in this case. When Tentry receives the first end-to-end RESERVE message from the sender, it chooses the tunnel flow ID, creates the tunnel session and sends a tunnel QUERY' message matching the requests of the end-to-end session toward the Texit. This Tunnel QUERY' message should have the "RESERVE-INIT" bit set. Tentry also appends to the original QUERY message with a tunnel BOUND_SESSION_ID object containing the session ID of the tunnel session and sends it toward the Texit using normal tunnel encapsulation.

The tunnel QUERY' message is processed hop by hop inside the tunnel for the flow identified by the chosen tunnel flow ID. When Texit receives the tunnel QUERY' message, it creates a reservation state for the tunnel session and immediately send out a tunnel RESERVE' message back to Tentry.

When the Tentry receives the tunnel RESERVE' message it learns the outcome of the tunnel reservation. So it appends to the end-to-end RESERVE message a BOUND_SESSION_ID object containing the tunnel session ID and sends it over the tunnel with normal encapsulation. It may send out a tunnel RESPONSE' message if requested.

When Texit receives the end-to-end RESERVE message, it notices the binding of a tunnel session and creates the end-to-end reservation state with reference to the tunnel session state, removes the tunnel BOUND_SESSION_ID object and forwards the end-to-end RESERVE message further along the path towards the receiver. When the end-to-end reservation finishes, an end-to-end RESPONSE may be sent back from the receiver to the sender.

4.1.4. Receiver-initiated Reservation for End-to-end and Sender-initiated Reservation for Tunnel Signaling

```
     Sender    Tentry      Tnode      Texit     Receiver
     |          |          |          |          |
     |  QUERY   |          |          |          |
     +--------->|          |          |          |
     |          |        QUERY        |          |
     |          +-------------------->|          |
     |          |  QUERY'  |          |          |
     |          +=========>|          |          |
     |          |          |  QUERY'  |          |
     |          |          +=========>|          |
     |          |          |          |  QUERY   |
     |          |          |          +--------->|
     |          |          |          | RESERVE  |
     |          |          |          |<---------+
     |          |       RESERVE       |          |
     |          |<--------------------+          |
     |          |          | RESERVE' |          |
     |          |          +=========>|          |
     |          | RESERVE' |          |          |
     |          +=========>|          |          |
     |          |          | RESPONSE'|          |
     |          |          |<=========|          |
     |          | RESPONSE'|          |          |
     |          |<=========|          |          |
     | RESERVE  |          |          |          |
     |<---------|          |          |          |
     | RESPONSE |          |          |          |
     +--------->|          |          |          |
     |          |       RESPONSE      |          |
     |          +-------------------->|          |
     |          |          |          | RESPONSE |
     |          |          |          +--------->|
     |          |          |          |          |
     |          |          |          |          |
```

Figure 4: Receiver-initiated Reservation for End-to-end and Sender-initiated Reservation for Tunnel Signaling

This scenario assumes the end-to-end signaling mode is receiver-initiated and the tunnel signaling mode is sender-initiated. Figure 4 shows the messaging flow of NSIS operation over IP tunnels in this case. When Tentry receives the first end-to-end QUERY message from the sender, it chooses the tunnel flow ID, creates the tunnel session and sends a tunnel QUERY' message matching the requests of the end-to-end session toward the Texit. Tentry also appends to the original QUERY message a tunnel BOUND_SESSION_ID

object containing the session ID of the tunnel session and sends it toward the Texit using normal tunnel encapsulation.

The tunnel QUERY' message is processed hop by hop inside the tunnel for the flow identified by the chosen tunnel flow ID. When Texit receives the tunnel QUERY' message, it creates a reservation state for the tunnel session without sending out a tunnel RESERVE' message immediately.

The end-to-end QUERY message passes along tunnel intermediate nodes just like any other tunneled packets. When Texit receives the end-to-end QUERY message, it notices the binding of a tunnel session and checks state for the tunnel session. When the tunnel session state is available, Texit updates the end-to-end QUERY message using the tunnel session state, removes the tunnel BOUND_SESSION_ID object and forwards the end-to-end QUERY message further along the path.

When Texit receives the first end-to-end RESERVE message issued by the receiver, it finds the reservation state of the tunnel session. Texit appends to the end-to-end RESERVE message a tunnel BOUND_SESSION_ID object containing the matching tunnel session ID and sends it upstream to Tentry.

When Tentry receives the end-to-end RESERVE message, it notices the binding and immediately sends out a tunnel RESERVE' message matching the end-to-end RESERVE request over the tunnel. This RESERVE' message should include the Request Identification Information (RII) to trigger a RESPONSE' from Texit.

When Tentry receives the result of tunnel reservation from the tunnel RESPONSE' message, it updates the end-to-end RESERVE message and forwards the end-to-end RESERVE message upstream to the Sender. The Sender may send an end-to-end RESPONSE message to the receiver when the whole process completes.

4.2. Implementation Considerations

4.2.1. End-to-end and Tunnel Signaling Interaction

Given the two separate end-to-end and tunnel signaling sessions, there are many ways of integrating the signaling of each session. In general, different approaches can be grouped into two modes, sequential mode and parallel mode. In sequential mode, end-to-end signaling pauses when it is waiting for results of tunnel signaling, and resumes upon receipt of the tunnel signaling outcome; in parallel mode, end-to-end signaling continues outside the tunnel while tunnel signaling is still in process and its outcome is unknown. The operation outlined in Section 4.1 shows the sequential mode. While

this mode is ideal for a flow that requires hard guarantee of tunnel reservation. It may not be the best for a flow that can tolerate some QoS uncertainty but wants to establish signaling state on the path as fast as possible. The parallel mode is clearly for the latter case.

Therefore, an actual implementation may vary the timing sequence of the NSIS-tunnel signaling interaction by taking into account whether the end-to-end flow can tolerate the "reservation uncertainly". The root of this problem has to do with the possible racing condition that always exists in a separate end-to-end and tunnel session model. When an end-to-end session message carrying tunnel binding object arrives at one of the tunnel endpoints, if the corresponding tunnel session state has already been created, then the tunnel endpoint may refer to information from the tunnel session state (e.g. about tunnel reservation status, or tunnel resource availability) and construct an end-to-end signaling message to be sent out of the tunnel immediately.

On the other hand, if the tunnel endpoint receives an end-to-end signaling message carrying tunnel binding referring to a tunnel session not yet exists, it may either wait a short period and see if the tunnel signaling arrives, or forward the end-to-end session immediately but indicate in the outgoing end-to-end signaling message that there is a NON-QoSM aware link in the end-to-end path. The period that the node decides to wait is purely implementation specific. In normal cases we expect the tunnel signaling message to lag behind (if it does) by only some node processing time (because the end-to-end signaling messages are not processed by NSLP inside the tunnel). As to whether wait or not, the decision will be based on the flow's toleration about "reservation uncertainly". With the current QSPEC [14], one option is to wait shorter or does not wait for end-to-end reservations requirements that are downgradable, and to wait until the tunnel session status is known if the end-to-end reservations requirement is fixed. However, to really directly address this issue, we suggest that an explicit indication flag, e.g. "QoS Unknown - intolerable" be defined as part of the QSPEC.

In the situation where the end-to-end signaling missed the tunnel session state in the tunnel endpoint and proceeds as if the tunnel is NON-QoSM aware, the tunnel session may still be created (after some delay). Since the tunnel signaling message does not contain the end-to-end session's session ID, it cannot immediately change the state of the end-to-end session. However, the next refresh of the end-to-end signaling message will carry the tunnel binding information and thus update its own state. If the period waiting for end-to-end refresh is considered too long, the tunnel endpoint may choose to actively poll the session state table about the existence of tunnel

session before the refresh timer expires. In any case, once the end-to-end signaling session learns about the tunnel signaling it can send an immediate refresh out of the tunnel with the knowledge of tunnel session.

In addition to the broad concept of sequential and parallel modes of interaction. There are also other flexible aspects in the end-to-end and tunnel signaling interaction. One is tunnel session initiation location. e.g., it is possible to initiate the tunnel session from Texit instead of Tentry. Second is the tunnel session initiation time point. E.g. in cases when both end-to-end session and tunnel session are receiver-initiated, it is possible to start the tunnel session when Tentry receives the first end-to-end RESERVE message. The drawback of this scheme is that it will not allow the first end-to-end QUERY message to trigger a tunnel QUERY' and gather tunnel characteristics along with the rest of the end-to-end path. A third aspect of flexibility is how the tunnel signaling messages are used. e.g., in the case where end-to-end session is receiver initiated and tunnel session is sender initiated as in Section 4.1.4, the first tunnel QUERY' message sent after receiving end-to-end QUERY message by Tentry can be replaced by a tunnel RESERVE' message, if the application wants to trade temporary oversized or unnecessary (if the end-to-end reservation turns out to be unsatisfied) tunnel resource reservations for signaling setup delay. All these may be seen as local optimization issues. An implementation should at least support the basic scheme to allow interoperability.

4.2.2. Aggregate vs. Individual Tunnel Session Setup

The operation outlined in Section 4.1 applies to a flow that qualifies an individual dynamic tunnel session. For a tunnel that contains multiple end-to-end sessions, however, it is generally recommended to keep aggregate tunnel session rather than creating individual tunnel sessions for each end-to-end session whenever possible. This will save the cost of setting up a new session and avoid the set up latency as well as the session establishment racing conditions mentioned above. Therefore, when the tunnel endpoint creates the reservation for the tunnel session based on the individual end-to-end session, it is up to local policy that whether it wants to actually create an aggregate session by requesting more resources than the current end-to-end session requires. If it does, other end-to-end sessions arrived later may also make use of this tunnel session. The tunnel endpoint will also need to determine how long to keep the tunnel session if no end-to-end session is active. The decision may be based on knowledge of likelihood of traffic in the future. It should be noted that once these kinds of on-demand aggregate tunnel session is setup, it looks exactly the same as a pre-configured tunnel session to future end-to-end sessions.

Therefore, the adjustment of such aggregate sessions should follow Section 5.

Note that the session ID of an aggregate tunnel session should be different from that of the end-to-end session because they usually have separate lifetime. If the tunnel endpoint is certain that the tunnel session is for an individual end-to-end session alone, it may in some cases want to use the same session ID for both sessions. This may require additional manipulation of the NSLP state at the tunnel endpoints, since the NSLP state is usually keyed by the session ID.

5. Protocol Operation with Pre-configured Tunnel Sessions

A tunnel may be pre-configured through management interface with one or more tunnel sessions. One or more end-to-end sessions may be mapped to each of these pre-configured sessions. Therefore in most cases these pre-configured tunnel sessions are aggregate sessions.

5.1. Tunnel with Exactly One Pre-configured Aggregate Session

If only one aggregate session is configured in the tunnel and all traffic will receive the reserved tunnel resources, all the packets just need to be normal IP-in-IP encapsulated. If there is only one aggregate session configured in the tunnel and only some traffic should receive the reserved resources through that aggregate tunnel session, then the aggregate tunnel session should be assigned an appropriate flow ID. Qualified packets need to be encapsulated with this flow ID. The rest of the traffic will be normal IP-in-IP encapsulated.

5.2. Tunnel with Multiple Pre-configured Aggregate Sessions

If there are multiple configured aggregate sessions over a tunnel set up by the management interface, these sessions must be distinguished by their aggregate tunnel flow IDs based on appropriate flow ID. In this case it is necessary to explicitly bind the end-to-end sessions with the specific tunnel sessions. This binding is provided by the tunnel BOUND_SESSION_ID object which is carried in the end-to-end signaling messages. Once the binding has been established, Tentry should encapsulate qualified data packets from different flows according to the associated aggregate tunnel flow ID. Intermediate nodes in the tunnel will then be able to filter these packets to receive reserved resources.

5.3. Adjustment of Pre-configured Tunnel Sessions

The reservation of a configured tunnel session may or may not be adjustable. When the tunnel session is adjustable and there can be a many-to-one mapping to the tunnel session, related policy mechanism needs to decide how the adjustment to the tunnel reservation should be done to accommodate the end-to-end sessions mapped onto it. As discussed in [18], there could be multiple choices. In the first, the tunnel reservation is never adjusted, which makes the tunnel a rough equivalent of a fixed-capacity hardware link ("hard pipe"). In the second, the tunnel reservation is adjusted whenever a new end-to-end reservation arrives or an old one is torn down ("soft pipe"). Doing this will require the Texit to keep track of the resources allocated to the tunnel and the resources actually in use by end-to-end reservations separately. It is often appropriate to adopt a third choice, where we use some hysteresis in the adjustment of the tunnel reservation parameters. 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.

5.4. Tunnels with both Dynamic and Pre-configured Signaling Sessions

If a tunnel contains both dynamic and pre-configured tunnel sessions, it can be handled by corresponding mechanisms discussed above. The choice of mapping an end-to-end session to a specific type of tunnel session is up to policy control.

6. Processing Rules for Selected End-to-end QoS NSLP Messages

The following lists basic message processing rules for end-to-end QoS NSLP messages working in the sequential interaction mode with tunnel signaling. More details may be provided for this section in future version of this document.

Note that in case of aggregate tunnels, the actual tunnel session reservation, adjustment and termination are not (necessarily) determined by every end-to-end signaling messages, but by implementation specific algorithms instead.

6.1. End-to-end QUERY Message at Tentry

When an end-to-end QUERY message is received at Tentry, Tentry checks whether the end-to-end session is entitled to tunnel resources.

If the end-to-end session should be bound to a tunnel session yet to be created. Tentry creates a tunnel QUERY' message and sends it to Texit. Tentry also appends a tunnel BOUND_SESSION_ID object to the

end-to-end QUERY message. The tunnel BOUND_SESSION_ID object contains the session ID of the tunnel session. The end-to-end QUERY message is then encapsulated and sent out through the tunnel interface.

If the end-to-end session should be bound to an existing tunnel session (whether aggregate or individual), Tentry appends a tunnel BOUND_SESSION_ID object to the end-to-end tunnel QUERY message and sends it toward Texit through the tunnel interface.

6.2. End-to-end QUERY Message at Texit

When an end-to-end QUERY message containing a tunnel BOUND_SESSION_ID object is received, Texit creates a conditional reservation state for the end-to-end session (i.e., a state is created but the related outgoing signaling message, in this case the QUERY message, is held until further information is available). It also checks to see if a conditional reservation state for the associated tunnel session is available. If yes, it reads information from the tunnel session state and sends the end-to-end QUERY downstream. If the conditional reservation state for tunnel session is not yet available, it will be created upon receiving the tunnel QUERY', and then Texit should forward the end-to-end QUERY downstream with information from results of the tunnel QUERY'.

6.3. End-to-end RESERVE Message at Tentry

When a RESERVE message is received, in addition to normal processing for the request, the following tunnel related functionality is performed.

For sender-initiated RESERVE message,

If the RESERVE message is received with its T-bit set (RESERVE tear), Tentry removes the local state, then encapsulates the RESERVE message and tunnels it to Texit. If there is a tunnel session associated with this end-to-end session, Tentry also sends a tunnel RESERVE with T-bit set for that tunnel session.

If the end-to-end RESERVE message is a refresh for an existing end-to-end session and this session is associated with a tunnel session, the RESERVE message refreshes both two sessions. If the RESERVE message causes changes in resources reserved for the end-to-end session, depending on whether the tunnel signaling is sender initiated or receiver initiated, Tentry should create a new tunnel RESERVE' message or tunnel QUERY' message to start changing the tunnel reservation as well. At the same time, Tentry appends a tunnel BOUND_SESSION_ID object to the end-to-end RESERVE message and

sends it to Texit through the tunnel interface.

If the message is the first RESERVE message for an end-to-end session, Tentry determines whether the end-to-end session is entitled to tunnel resources based on policy control mechanisms outside the scope of this document. If not, no special tunnel related processing is needed. Otherwise, if this session should be bound to an existing tunnel session (whether aggregate or individual), Tentry creates the association between the end-to-end session and the tunnel session. Then it appends a tunnel BOUND_SESSION_ID object to the end-to-end RESERVE message and sends it through the tunnel interface (i.e. the message is encapsulated and tunneled to Texit as normal).

If the end-to-end session should be bound to a tunnel session yet to be created, Tentry assigns the tunnel flow ID, and constructs a tunnel RESERVE' or QUERY' message, depending on whether the tunnel signaling is sender initiated or receiver initiated. The QSPEC in this tunnel message may be different from the original QSPEC, taking into consideration the tunnel overhead of the encapsulation of data packets. Tentry then associates the tunnel session with the end-to-end session in the NSLP state and sends the tunnel message toward Texit to start reserving resources over the tunnel. At the same time, Tentry appends a tunnel BOUND_SESSION_ID object to the end-to-end RESERVE message and sends it through the tunnel interface.

For receiver-initiated RESERVE messages,

If the RESERVE message is received with its T-bit set (RESERVE tear), Tentry removes the local state and forwards the message upstream. If the tunnel signaling is sender initiated, Tentry also sends a tunnel RESERVE' message to teardown the tunnel session.

If the end-to-end RESERVE message contains a tunnel BOUND_SESSION_ID and is the first end-to-end RESERVE message, Tentry checks whether the tunnel session bound to the end-to-end session indicated by the RESERVE message already exists. If yes, Tentry records the association between the end-to-end and the tunnel session, reads information from the tunnel session to create the end-to-end RESERVE message to be forwarded upstream. If the state for the tunnel session is not available yet, Tentry should create state information for the tunnel session and indicate that a conditional reservation is pending. If tunnel signaling is sender initiated, Tentry also sends a tunnel RESERVE' message toward Texit to reserve tunnel resources. When the actual tunnel session status is known at Tentry (from a tunnel RESERVE' if tunnel signaling is receiver initiated or at tunnel RESPONSE' if tunnel signaling is sender initiated) and if at this time there is a pending reservation, Tentry should generate an end-to-end RESERVE message and forward it upstream.

If the end-to-end RESERVE message contains a tunnel BOUND_SESSION_ID and is a refresh, Texit refreshes the end-to-end session. If the RESERVE message causes changes in resources reserved for the end-to-end session and if tunnel signaling is sender initiated, Tentry sends a tunnel RESERVE' message to Texit to change the reservation. In any case, Texit checks the state information of the tunnel session. If it finds that the reservation has been updated inside the tunnel, Texit forwards the changed RESERVE message toward the sender. If the tunnel reservation update failed, Texit MUST send a RESPONSE with appropriate Error_Spec to the originator of the end-to-end RESERVE message.

6.4. End-to-end RESERVE Message at Texit

When Texit receives a RESERVE message, in addition to normal processing of the request, the Texit performs the following steps,

Sender-initiated RESERVE,

If the end-to-end RESERVE message is received with its T-bit set (RESERVE tear), Texit removes the local state, then forwards the RESERVE message downstream. If tunnel signaling is receiver-initiated, Texit also sends a tunnel RESERVE tear upstream toward Tentry to tear down the tunnel session.

If the end-to-end RESERVE message contains a tunnel BOUND_SESSION_ID and is the first end-to-end RESERVE message, Texit checks whether the state for the tunnel session indicated by the RESERVE message already exists. If yes, Texit records the association between the end-to-end and the tunnel session and reads information from the tunnel session to create the end-to-end RESERVE message to be forwarded downstream. If the state for the tunnel session is not available yet, Texit should create state information for the tunnel session and indicate that a conditional reservation is pending. When the actual tunnel RESERVE' arrives, the tunnel session state will be updated. If at this time there is a pending reservation, Texit will generate an end-to-end RESERVE message and forwards it downstream.

If the end-to-end RESERVE message contains a tunnel BOUND_SESSION_ID and is a refresh, Texit refreshes the end-to-end session. If the RESERVE message causes changes in resources reserved for the end-to-end session, Texit checks the state information of the tunnel session. If the reservation has been updated inside the tunnel, Texit forwards the RESERVE message toward the receiver. If the tunnel reservation update failed, Texit MUST send a RESPONSE with appropriate Error_Spec to the originator of the end-to-end RESERVE message.

Note that the processing rules for end-to-end RESERVE at Texit in end-to-end sender-initiated case is similar to those for end-to-end RESERVE at Tentry in end-to-end receiver-initiated case.

Receiver-initiated RESERVE,

If the RESERVE message is received with its T-bit set (RESERVE tear), Texit removes the local state, then forwards the RESERVE message upstream. If there is an individual tunnel session associated with this end-to-end session, Texit also sends a tunnel RESERVE' with T-bit set for that tunnel session.

Otherwise Texit checks to see if the end-to-end session is associated with a tunnel session. If only conditional reservation state is found and no actual reservation has been made, this RESERVE is the first end-to-end RESERVE message. Texit appends a tunnel BOUND_SESSION_ID object to this end-to-end RESERVE message and sends it toward Tentry through the tunnel interface. Meanwhile if tunnel signaling is receiver initiated Texit sends tunnel RESERVE' message toward Tentry to reserve tunnel resources.

If the end-to-end session is bound to a tunnel session and the RESERVE message is a refresh, it refreshes both the end-to-end session and tunnel session. If the RESERVE message causes changes in resources reserved for the end-to-end session and if tunnel signaling is receiver initiated, Texit may create a new tunnel RESERVE' message to change the tunnel reservation as well. Meanwhile, the end-to-end RESERVE is appended with the tunnel BOUND_SESSION_ID object and sent to Tentry through the reverse path.

6.5. Special Processing Rules for Tunnels with Aggregate Sessions

In situations where the end-to-end session is bound to aggregate tunnel sessions, the handling is similar to that of [18].

If the associated tunnel session is a "hard pipe" session, arrival of a new end-to-end reservation or adjustment of an existing end-to-end session may cause the overall resources needed in the tunnel session to exceed its capacity, this case is treated as admission control failure same as that of a tunnel reservation failure. Tentry should create a RESPONSE message with appropriate Error_Spec and send it to the originator of the RESERVE message.

If the associated tunnel session is a "soft pipe" session, arrival of a new end-to-end reservation or adjustment/deletion of existing sessions may cause the tunnel session to be modified. It is recommended that some hysteresis is enforced in the adjustment of the tunnel reservation parameters. This requires tunnel endpoint to keep

track of both the allocated tunnel session resources and the resources actually used by end-to-end sessions bound to that tunnel session.

7. Other Considerations

7.1. Other Types of NSLP

This document discusses QoS NSLP. It will be good if the scheme in this document could work with other NSLPs as well. Since NSIS-tunnel operation involves specific NSLP itself and different NSLPs have different message exchange semantics, the NSIS-tunnel specification would not be the same for all NSLPs. However the basic aspects behind NSIS-tunnel operation are indeed similar. NATFW NSLP is the only other main NSLP currently developed by the NSIS working group. The most important signaling operation in NATFW NSLP is CREATE. Assuming Tentry is a NATFW NSLP, the tunnel-handling for CREATE operation will be very similar to the sender-initiated QoS reservation case. There are also a number of reverse directional operations in NATFW NSLP, e.g., RESERVE_EXTERNAL_ADDRESS, UCREATE. It is not very clear whether tunnel will cause problems with these messages in general. But they are likely easier to be dealt with than the receiver-initiated reservation case in QoS NSLP. This topic will be discussed in future version of this document if necessary.

7.2. IPSEC Flows

If the tunnel supports IPSEC (especially ESP in Tunnel-Mode with or without AH), it may use the flow label, TC field, or IPSEC SPI along with the tunnel source and destination address, as discussed in Section 3.1 to form the tunnel Flow ID. All these are standard NSIS MRI fields that should be matched by the NSIS packet classifier. We may also define virtual destination ports as in [19] to provide further flow demultiplexing capability at the destination side if necessary.

7.3. NSIS-Tunnel and Mobility

The NSIS-tunnel operation needs to interact with IP mobility in an efficient way. In places where pre-configured tunnel sessions are available, the process is relatively straightforward. For dynamic individual signaling tunnel sessions, one way to improve tunnel NSIS-mobility efficiency is to reuse the session ID of the tunnel session when tunnel flow ID changes during mobility, as illustrated below.

With a mobile IP tunnel, one tunnel endpoint is the Home Agent (HA), and the other endpoint is the Mobile Node (MN) if collocated Care-of-

Address (CoA) is used, or the Foreign Agent (FA) if FA CoA is used. When MN is a receiver, Tentry is the HA and Texit is the MN or FA. In case of a mobility event, handoff tunnel signaling messages will start from HA, which may use the same session ID for the new tunnel session. When MN is a sender and collocated CoA is used, Tentry is the MN and Texit is the HA. Handoff tunnel signaling is started at the MN. It may also use the session ID of the previous tunnel session for the new tunnel session. When MN is a sender and FA CoA is used, the situation is complicated, because Tentry has changed from the old FA to the new FA. The new FA does not have the session ID of the previous tunnel session.

When mobile IP is working on a bi-directional tunneling mode, NSIS-tunnel operation with mobility may be further improved by localizing the handoff tunnel signaling process under the HA (i.e., without going through the path between HA and CN).

8. Security Considerations

This draft does not draw new security threats. Security considerations for NSIS NTLP and QoS NSLP are discussed in [2] and [3] respectively. General threats for NSIS can be found in [21].

9. Appendix

9.1. Summary of RSVP Operation Over IP Tunnels

RFC 2746 [18] provides an example scheme for RSVP operation over IP tunnels. The scheme needs to be supported by both the Tentry and Texit. To address the tunnel signaling visibility problem, separate tunnel signaling sessions are performed for end-to-end sessions. A binding between the tunnel sessions and the end-to-end sessions is established. Both the Tentry and Texit must agree on the binding 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 endpoints can be correctly transformed into errors reported by the tunnel endpoints to the end-to-end RSVP session. To address the tunnel QoS data visibility problem, a UDP header is inserted to all QoS data packets following the tunnel IP header. The additional UDP header provides source and destination ports that allow intermediate tunnel nodes to use standard RSVP filterspec handling and demultiplex different tunnel RSVP sessions.

The RFC 2746 scheme also mentions that in the case where the IP-in-IP tunnel supports IPSEC (especially ESP in tunnel-mode with or without

AH), the tunnel session uses the GPI SESSION and GPI SENDER_TEMPLATE, FILTER_SPEC as defined in [19] 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.

9.2. Various Design Alternatives

9.2.1. End-to-end and Tunnel Signaling Interaction Model

The contents of original end-to-end singling messages are not directly examined by tunnel intermediate nodes. To carry out tunnel signaling we choose to maintain a separate tunnel session for the end-to-end end session by generating separate signaling messages for the tunnel signaling session. Another possibility is to stack tunnel specific objects on top of the original end-to-end message and make these messages visible to tunnel intermediate nodes so they may serve both the end-to-end session and tunnel session. This turns out to be difficult because the actual tunnel signaling messages differ from the end-to-end signaling message both in GIST layer and NSLP layer information, such as MRI, PACKET CLASSIFIER and QSPEC. Although QSPEC can be stacked in an NSLP message, there doesn't seem to be a handy way to stack MRI and the PACKET CLASSIFIER in the NSLP layer. In addition, the stacking method only applies to individual signaling tunnels.

The separate end-to-end tunnel session signaling model adopted in this document handles both individual and aggregate signaling tunnels in a consistent way. Its major drawback is the racing condition we mentioned in Section 4.2. However, this can be readily handled with the introducing of a flag indicating whether the flow is willing to tolerate "tunnel reservation uncertainty".

To support tunnel signaling it is natural that at least one of the tunnel endpoints will need to understand the NSIS-tunnel operation. We see that Tentry always needs to be NSIS-Tunnel aware because it at least needs to encapsulate packets into special tunnel flow IDs. Texit needs to be NSIS-tunnel aware if the tunnel reservation is receiver initiated. When the tunnel reservation is sender-initiated, it is possible that Texit is NSIS-Tunnel unaware and the tunnel signaling still works. However, the condition is that no special packet decapsulation is needed (e.g. when UDP insertion is used for tunnel flow ID). Considering that most of the time we might have a bi-directional tunnel and also for more general applicability, we assumed both tunnel endpoints to be NSIS-Tunnel aware in this document.

9.2.2. Packet Classification over the Tunnel

Packet classification over the tunnel may be done in either of the two ways: first, retaining the end-to-end packet classification rules; second, using tunnel specific classification rules. In the first approach, tunnel packet classification is not tied with tunnel MRI. This is a useful property especially in handling tunnel mobility - as mobility occurs, the tunnel MRI changes, but the packet classification rule does not change. Therefore, the common path after a handoff does not need to be updated about the packet classification, resulting in a better handoff performance. The main problem with this approach is that most existing routers do not support inspection of inner IP headers in an IP tunnel, where the tunnel independent packet classification fields usually reside. Therefore this document chooses the second approach which does not pose special requirements on intermediate tunnel nodes.

9.2.3. Tunnel Binding Methods

In this document, the end-to-end session and tunnel session use different session IDs and they are associated with each other using the BOUND_SESSION_ID object. This choice is obvious for aggregate signaling tunnels because in that case the original end-to-end session and the corresponding aggregate tunnel session require independent control.

Sessions in individual signaling tunnels are created and deleted along with the related end-to-end session. So association between the end-to-end session and the corresponding individual tunnel session has another choice: the two sessions may share the same session ID. Instead of sending a BOUND_SESSION_ID object, it may be possible to define a BOUND_FLOW_ID object, to bind the flow ID of the end-to-end session to the flow ID of the tunnel session at the tunnel endpoints. However, since flow ID is usually derived from MRI, if a NAT is present in the tunnel, this BOUND_FLOW_ID object will have to be modified in the middle, which makes the process fairly complicated. Furthermore, it is not desired to have different session association mechanisms for aggregate signaling tunnels and individual signaling tunnels. Therefore, we decide to use the same tunnel BOUND_SESSION_ID mechanism in individual signaling tunnels. Note that, in this case the mobility handling inside the tunnel can still be optimized in certain situations, as discussed in Section 7.3.

9.2.4. Tunnel Binding Indication

In this document we used the existing BOUND_SESSION_ID object with a tunnel Binding_code to indicate the reason of binding. Two other options considered are:

1. Define a designated "tunnel object" to be included when the tunnel binding needs to be conveyed.
2. Define a "tunnel bit" in corresponding NSLP message headers.

These options are not chosen because they either need to create entirely new object, or need to change basic message headers. They are also not generic solutions that can cover other binding causes.

9.2.5. Carrying the Tunnel Binding Object

There are basically three ways to carry the binding object between Tentry and Texit, using (a) end-to-end signaling messages (b) tunnel signaling messages. (c) both end-to-end and tunnel signaling messages.

In option (a) only tunnel endpoints sees the tunnel binding information. While in option (b), every intermediate node sees the binding information. Since there will be no state for the end-to-end session in the tunnel intermediate nodes, they will all generate a message containing an "INFO_SPEC" object indicating no bound session found according to [3], which is not acceptable. Option (c) has a good point that if both end-to-end and tunnel signaling messages have tunnel binding information, the racing condition will be resolved faster. However it suffers the same problem as in (b). Therefore the choice in this document is option (a).

9.3. Change History

9.3.1. Changes in Version -02

1. Rearranged section names to emphasize the difference between dynamically created tunnel sessions and pre-configured tunnel sessions.
2. Added implementation considerations section about how to deal with the race condition in the separate session model, and allowed the dynamically created tunnel session to be an aggregate session.
3. Added operation examples on the two scenarios where e2e and tunnel session uses different signaling initiation modes.
4. Removed the illustration of binding_code for tunnel BOUND_SESSION_ID object since it has been added to NSLP specification.
5. Clarified that tunnel capability discovery is at NSLP layer.
6. Updated some of the message processing rules.
7. Updated some parts of the appendix.

9.3.2. Changes in Version -01

1. Added message processing rules.
2. Put some of the backgrounds and alternative design choices to appendix.
3. Proposed the binding_code for tunnel BOUND_SESSION_ID object.

10. Acknowledgements

11. References

11.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Schulzrinne, H. and R. Hancock, "GIST: General Internet Signaling Transport", draft-ietf-nsis-ntlp-09 (work in progress), February 2006.

[3] Manner, J., "NSLP for Quality-of-Service signalling", draft-ietf-nsis-qos-nslp-09 (work in progress), February 2006.

11.2. Informative References

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

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

[6] Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, August 2000.

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

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

[9] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[10] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[11] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6

Specification", RFC 2473, December 1998.

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

[13] Hancock, R., "Next Steps in Signaling: Framework", draft-ietf-nsis-fw-07 (work in progress), December 2004.

[14] Ash, J., "QoS-NSLP QSPEC Template", draft-ietf-nsis-qspec-08 (work in progress), December 2005.

[15] Stiemerling, M., "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)", draft-ietf-nsis-nslp-natfw-09 (work in progress), February 2006.

[16] Lee, S., "Applicability Statement of NSIS Protocols in Mobile Environments", draft-ietf-nsis-applicability-mobility-signaling-03 (work in progress), October 2005.

[17] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, March 2000.

[18] Terzis, A., Krawczyk, J., Wroclawski, J., and L. Zhang, "RSVP Operation Over IP Tunnels", RFC 2746, January 2000.

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

[20] Rajahalme, J., Conta, A., Carpenter, B., and S. Deering, "IPv6 Flow Label Specification", RFC 3697, March 2004.

[21] Tschofenig, H. and D. Kroeselberg, "Security Threats for Next Steps in Signaling (NSIS)", RFC 4081, June 2005.

Authors' Addresses

Charles Shen
Columbia University
Department of Computer Science
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA

Phone: +1 212 854 5599
Email: charles@cs.columbia.edu

Henning Schulzrinne
Columbia University
Department of Computer Science
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA

Phone: +1 212 939 7004
Email: schulzrinne@cs.columbia.edu

Sung-Hyuck Lee
SAMSUNG Advanced Institute of Technology
San 14-1, Nongseo-ri, Giheung-eup
Yongin-si, Gyeonggi-do 449-712
KOREA

Phone: +82 31 280 9552
Email: starsu.lee@samsung.com

Jong Ho Bang
SAMSUNG Advanced Institute of Technology
San 14-1, Nongseo-ri, Giheung-eup
Yongin-si, Gyeonggi-do 449-712
KOREA

Phone: +82 31 280 9585
Email: jh0278.bang@samsung.com

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.