IETF Next Steps in Signaling
Internet-Draft
Intended status: Informational
Expires: September 3, 2008

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

March 2, 2008

NSIS Operation Over IP Tunnels
draft-ietf-nsis-tunnel-04.txt

Status of this Memo

Copyright Notice

Abstract

This draft presents an NSIS operation over IP tunnel scheme using QoS NSLP as the NSIS signaling application. Both sender-initiated and receiver-initiated NSIS signaling modes are discussed. The scheme creates individual or aggregate tunnel sessions for end-to-end sessions traversing the tunnel. Packets belonging to qualified end-

to-end sessions are mapped to corresponding tunnel sessions and assigned special flow IDs to be distinguished from the rest of the tunnel traffic. Tunnel endpoints keep the association of the end-to-end and tunnel session mapping, so that adjustment in one session can 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

When IP tunnel mechanism is used to transfer signaling messages, e.g., NSIS messages, the signaling messages usually become hidden inside the tunnel and are not known to the tunnel intermediate nodes. In other words, the IP tunnel behaves as a logical link that does not support signaling in the end-to-end path. If true end-to-end signaling support is desired, there needs to be a scheme to enable signaling at the tunnel segment of the end-to-end signaling path. This draft describes such a scheme for NSIS operation over IP tunnels. We assume QoS NSLP as the NSIS signaling application.

2.1. IP Tunneling Mechanisms and Tunnel Signaling Capability

There are a number of common IP tunneling mechanisms, such as Generic Routing Encapsulation (GRE) [4][15], Generic Routing Encapsulation over IPv4 Networks (GREIP4) [5] , IP Encapsulation within IP (IP4INIP4) [7], Minimal Encapsulation within IP (MINENC) [8], Generic Packet Tunneling in IPv6 Specification (IP6GEN) [11], IPv6 over IPv4 tunneling (IP6INIP4) [9], IPSEC tunneling mode [19][10]. These mechanisms can be differentiated according to the format of the tunnel encapsulation header. IP4INIP4, IP6INIP4 and IP6GENIP4 can be seen as normal IP in IP tunnel encapsulation because their tunnel encapsulation headers are in the form of a standard IP header. All GRE-related IP tunneling (GRE,GREIP4), MINENC and IPSEC tunneling mode can be seen as modified IP in IP tunnel encapsulation because the tunnel encapsulation header contains additional information fields besides a standard IP header. The additional information fields are the GRE header for GRE and GREIP4, the minimum encapsulation header for MINENC and the Encapsulation Security Payload (ESP) header for IPSEC tunneling mode.

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. A signaling-aware IP tunnel can participate in a signaling network in various ways. Prior work on RSVP operation over IP tunnles (RSVP-TUNNEL) [16] identifies two types of QoS-aware tunnels: a tunnel that can promise some overall level of resources but cannot allocate resources specifically to individual data flows, or a tunnel that can make reservations for individual end-to-end data flows. This

classification leads to two types of tunnel signaling sessions: individual tunnel signaling sessions that are created and torn down dynamically as end-to-end session come and go, and aggregate tunnel sessions that can either be fixed, or dynamically adjusted as the actually used session resources increase or decrease. Aggregate tunnel sessions are usually pre-configured but can also be dynamically created. A tunnel MAY contain only individual tunnel sessions or aggregate tunnel sessions or both.

2.2. NSIS Tunnel Operation Overview

This NSIP operation over IP tunnel scheme is designed to work with most, if not all, existing IP in IP tunneling mechanisms. The scheme requires the tunnel endpoints to support specific tunnel related functionalities. Such tunnel endpoints are called NSIS-tunnel capable endpoints. Tunnel intermediate nodes do not need to have special knowledge about this scheme. When tunnel endpoints are NSIS-tunnel capable, this scheme enables the proper signaling initiation and adjustment inside the tunnel to match the requests of the corresponding end-to-end session. In cases when tunnel session signaling status is uncertain or not successful, the end-to-end session will be notified about the existence of possible NSIS-unaware links in the end-to-end path.

The overall design of this NSIS operation over IP tunnel scheme is conceptually similar to RSVP-TUNNEL [16]. However, the details of the scheme address all the important differences of NSIS from RSVP. For example,

- 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.
- NSIS QoS NSLP allows both sender-initiated and receiver-initiated reservations; RSVP only supports receiver-initiated reservations.
- NSIS deals only with unicast; RSVP also supports multicast.
- 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, there are two main issues in a signaling operation over IP tunnel scheme. First, how packet classification is performed inside the tunnel. Second, 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. After a typical IP-in-IP tunnel encapsulation, packets from different flows appear as having the same flow ID which usually consists of the Tunnel Entry (Tentry) address and Tunnel Exit (Texit) address. Therefore, the flow ID for a signaled flow needs to contain further demultiplexing information to make it distinguishable from non-signaled flows, and also from other different signaled flows.

The special flow ID for signaled flows inside the tunnel needs to be carried in tunnel signaling messages, along with tunnel adjusted QoS parameters, to set up or modify the state information in tunnel intermediate nodes. This process creates separate tunnel signaling sessions between the tunnel endpoints. In most cases, it is necessary to maintain the state association between an 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. Flow Packet Classification over the Tunnel

A flow can be an individual flow or an aggregate flow. Flow ID formats that MAY be used to identify packets in individual tunnel flows are listed below.

o Selected fields from the base IP header portion of the tunnel encapsulation 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 [6], or the Traffic Class, also known as DiffServ Code Point (DSCP) field. Note that the DSCP field can also be used to represent an aggregate DiffServ flow. As long as individual flow classification is processed before aggregate flow classification, or a longest match kind of packet classifier is used, this individual tunnel flow demultiplexing with DSCP field can work. In the rare cases where these conditions cannot 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, using DSCP field as part of the tunnel flow ID can be applied to both IPv4 and IPv6 and is probably easier to deploy. The 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, this group of flow ID formats 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 base IP header portion of the tunnel encapsulation header, combined with fields from the addtional infromation in the tunnel encapsulation header. This applies to modified IP-in-IP encapsulation as we mentioned in Section 2.1. An example of the additional information field is the Security Prameter Index (SPI) field for IPSEC tunnels. Comparing with the flow ID formats in the first group, these flow ID formats might pose more requirements at the NSIS protocol side if the addition information field is unique to the specific tunnel mechanism and not already recognized in basic NSIS specification.

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

The above three groups of 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 a DSCP value. There are other options for aggregate flows. For example, When additional interfaces at tunnel endpoints are available, the IP address of an additional interface at Tentry plus the IP address of the Texit, MAY constitute an aggregate flow ID.

The decision of using a specific 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. As long as the flow ID format is supported, Tentry SHOULD encapsulate all incoming packets for the specific data flows according to the chosen flow ID format. Texit SHOULD be able to decapsulate the packets if any special tunnel flow encapsulation is performed at the Tentry.

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

Tunnel signaling messages contain tunnel specific parameters such as tunnel Message Routing Information (MRI) and tunnel adjusted QoS

parameters. But in general, the formats of tunnel signaling messages are the same as end-to-end signaling messages. Tunnel signaling is carried out according to the same signaling rules as for 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-tunnel 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), and etc. In most cases, we expect to have bi-directional tunnels, where both tunnel 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 tunnel handling as the reason for binding. In the rest of this document, we refer to a BOUND_SESSION_ID object with its tunnel binding_code set 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 and contains the session ID of the corresponding tunnel session. NSIS-tunnel aware endpoints that receive this tunnel BOUND_SESSION_ID object SHOULD perform tunnel related procedures and then remove it from any end-to-end signaling messages sent out of the tunnel.

4. Protocol Operation with Dynamically Created Tunnel Sessions

The operation details for NSIS signaling over IP tunnels are more complicated if the tunnel session needs to be dynamically created, comparing to the case when tunnel sessions are pre-configured. We discuss these two cases in this and the subsequent section, respectively. If a tunnel contains both dynamic and pre-configured tunnel sessions, it can be handled by the combination of the corresponding mechanism for each type of tunnel sessions. The choice of mapping an end-to-end session to a specific type of tunnel session is up to policy control.

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:

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

In the following we describe a typical NSIS end-to-end and tunnel signaling interaction process during the tunnel setup 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 tunnel reservation MUST be confirmed before the end-to-end reservation can proceed further outside the tunnel.

It SHOULD be noted that different flow QoS requirements and policy assumptions MAY cause the timing sequence of the messaging flow to be slightly different. This 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 can keep consistent. The exception is when the tunnel session is an aggregate session. In that 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, a 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 request 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 the 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' |          |
     |          |          |<=========+          |
     |          | RESERVE' |          |          |
```

```
     |          |<=========+          |          |
     |          | RESPONSE'|          |          |
     |          +=========>|          |          |
     |          |          | RESPONSE'|          |
     |          |          +=========>|          |
     |          |          |          | 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 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 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 sends out a tunnel RESERVE' message back to Tentry. When Tentry receives the tunnel RESERVE' message it learns the outcome of the tunnel reservation and sends a tunnel RESPONSE' message to Texit.

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. It learns the outcome of tunnel session reservation from the tunnel RESPONSE' message. Then 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.

4.1.4. 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 request 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 the 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 Specific Issues

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 interaction approaches can be grouped into 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 suitable for a flow that requires hard guarantee of tunnel reservation, it MAY not be the best choice for a flow that can tolerate some QoS uncertainty but wants to complete the signaling on the path as fast as possible. The parallel mode is clearly for the latter case.

Having two separate signaling sessions also causes a possible race condition. 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 can refer to information in 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 that does not yet exist, it MAY either wait until the tunnel session information is ready, or forward the end-to-end session signaling without waiting for the tunnel session. If the end-to-end signaling indeed proceeds in the absence of the tunnel session, the tunnel session MAY still be established after some delay. Since the tunnel signaling message does not contain its associated end-to-end session's session ID, it cannot immediately change the state of its associated end-to-end session. However, the next refresh of the corresponding end-to-end session will carry the tunnel binding information and thus will update the association of the end-to-end and the tunnel session state. If the period waiting for the end-to-end signaling 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 knowledge of tunnel session.

The decision on whether and how long to wait for the corresponding tunnel session information is implementation specific and controlled by the tunnel endpoints. This document only requires that if an NSIS-tunnel aware endpoint decides to go forward with the end-to-end signaling outside the tunnel with an uncertain tunnel session condition, it SHOULD indicate this in the corresponding end-to-end signaling messages. As far as QoS NSLP is concerned, this means the NON-QoSM Hop field [12] SHOULD be set to one. Note that in some

cases, the application using NSIS signaling MAY wish to indicate the preferred way of end-to-end and tunnel signaling interaction. For example, an application that can not tolerate any QoS uncertainty will prefer the sequential mode of operation; an application that has a looser QoS requirement MAY prefer the parallel mode of operation for faster signaling speed. Current NSIS specification does not contain fields to convey this preference. New objects or flags will need to be defined if this behavior is considered necessary.

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 MAY contain multiple end-to-end sessions, it is more efficient to keep aggregate tunnel sessions rather than individual tunnel sessions whenever possible. This will save the cost of setting up a new session and avoid the setup latency as well as the session establishment race conditions mentioned above. Therefore, when the tunnel endpoint creates a reservation for a tunnel session based on the individual end-to-end session, it is up to local policy 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 make use of this aggregate tunnel session. The tunnel endpoint will also need to determine how long to keep the tunnel session if no active end-to-end session is currently mapped to the aggregate tunnel session. 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 sessions are set up, they are treated the same as pre-configured tunnel sessions 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 reuse the same session ID for both sessions. This will require additional manipulation of the NSLP state at the tunnel endpoints, since the NSLP state is usually keyed based on the session ID.

5. Protocol Operation with Pre-configured Tunnel Sessions

This section discusses NSIS operation over tunnels that are pre-configured through management interface with one or more tunnel sessions. A pre-configured tunnel sessions MAY be mapped to one

session as an individual tunnel session but are usually mapped to multiple end-to-end sessions as an aggregate tunnel session.

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

5.2. Tunnel with Multiple Pre-configured Aggregate Sessions

If there are multiple pre-configured aggregate sessions over a tunnel set up, these sessions MUST be distinguished by their different aggregate tunnel flow IDs. In this case it is necessary to explicitly bind the end-to-end sessions with specific tunnel sessions. This binding is conveyed between tunnel endpoints by the tunnel BOUND_SESSION_ID object. Once the binding has been established, Tentry SHOULD encapsulate qualified data packets according to the associated aggregate tunnel flow ID. Intermediate nodes in the tunnel will then be able to filter these packets to receive reserved tunnel resources.

5.3. Adjustment of Pre-configured Tunnel Sessions

Adjustment of pre-configured tunnel sessions upon the change of its mapped end-to-end sessions is related is up to local policy mechanisms. RSVP-TUNNEL [16] described multiple choices to accomplish this. First, the tunnel reservation is never adjusted, which makes the tunnel a rough equivalent of a fixed-capacity hardware link ("hard pipe"). 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. The third approach adopts 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.

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

The following lists basic tunnel related message processing rules for selected end-to-end QoS NSLP messages working in the sequential interaction mode. They are provided as references for implementors to insure minimal interoperability.

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

6.3.1. 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.

6.3.2. Receiver-initiated RESERVE Message

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 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, 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

6.4.1. Sender-initiated RESERVE Message

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' or RESPONSE' message 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.

6.4.2. Receiver-initiated RESERVE Message

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 RSVP-TUNNEL [16].

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 INFO_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 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. Tunnel Signaling Capability Discovery

The NSIS-tunnel signaling operations described in this document assume both Tentry and Texit are NSIS-tunnel capable. If prior knowledge of the other endpoint's NSIS-tunnel capability is not available, we need a discovery mechanism to find that out. For this purpose, we define a new NODE_CHAR object.

The format of the NODE_CHAR object follows the general object definition in GIST [2]. It contains a fixed header giving the object Type and object Length, followed by the object Value as shown below.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |A|B|r|r|         Type          |r|r|r|r|        Length         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   //                             Value                           //
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type: NODE_CHAR

Length: Fixed (1 32-bit word)

Value:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |T|                            Reserved                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Value field currently contains a single 'T' bit, indicating the basic NSIS-tunnel scheme defined in this document. It is also possible to use multiple bits to define NSIS-tunnel capability in finer granularity. We have adopted the simplest approach by using only one bit. The remaining reserved bits can be used to signal other node characteristics in the future.

The bits marked 'A' and 'B' define the desired behavior for objects whose Type field is not recognized. If a node does not recognize the NODE_CHAR object, the desired behavior is "Ignore". That is, the object MUST be deleted and the rest of the message processed as usual. This can be satisfied by setting 'AB' to '01' according to GIST specification .

This NODE_CHAR object is included in a QUERY or RESERVE message by a tunnel endpoint who wishes to learn about the other endpoint's tunnel handling capability. The other endpoint that receives this object will know that the sending endpoint is NSIS-tunnel capable, and place the same object in a RESPONSE message to inform the sending endpoint of its own tunnel handling capability. The procedures for using NODE_CHAR object in the four dynamically created tunnel session scenarios are further detailed below.

If both end-to-end and tunnel sessions are sender-initiated (Section 4.1.1) and Tentry is NSIS-tunnel capable, the Tentry includes an RII object and a NODE_CHAR object with T bit set in the first end-to-end RESERVE message sent to Texit. When Texit receives this RESERVE message, if it supports NSIS tunneling, it learns that Tentry is NSIS-tunnel capable and includes the same object with T bit set in the RESPONSE message sent back to Tentry. Otherwise, Texit ignores the NODE_CHAR object. When Tentry receives the RESPONSE message, it learns whether Texit is NSIS-tunnel capable by examining the existence of the NODE_CHAR object and its T-bit. If both tunnel endpoints are NSIS-tunnel capable, the rest of the procedures will follow those defined in Section 4.1.1. Alternatively, Tentry MAY send out tunnel RESERVE message before the RESPONSE message confirming the NSIS-tunnel capability of Texit is received. If later it learns that the Texit is not NSIS-tunnel capable, it SHOULD send out teardown messages to cancel the tunnel session reservation that has already been made. This way the signaling process is faster when Texit is NSIS-tunnel capable, but it can lead to temporary waste of tunnel resources if Texit is not NSIS-tunnel capable.

If both end-to-end and tunnel sessions are receiver-initiated (Section 4.1.2) and Tentry is NSIS-tunnel capable, the Tentry includes an RII object and a NODE_CHAR object with T bit set in the first end-to-end QUERY message sent toward Texit. An NSIS-tunnel capable Texit learns from the NODE_CHAR object whether Tentry is

NSIS-tunnel capable. In reply to this end-to-end QUERY message, the NSIS-tunnel capable Texit includes a NODE_CHAR object with T bit set in its RESPONSE message to notify Tentry of its own tunnel capability. If both tunnel endpoints are NSIS-tunnel capable, the rest of the procedures will follow those defined in Section 4.1.2. Otherwise, Texit will not initiate tunnel session reservations.

If the end-to-end session is sender-initiated, the tunnel session is receiver-initiated (Section 4.1.3), and Tentry is NSIS-tunnel capable, the Tentry includes an RII object and a NODE_CHAR object with T bit set in the first end-to-end RESERVE message sent toward Texit. An NSIS-tunnel capable Texit learns from the NODE_CHAR object whether Tentry is NSIS-tunnel capable. In reply to this end-to-end QUERY message, the NSIS-tunnel capable Texit includes a NODE_CHAR object with T bit set in its RESPONSE message to notify Tentry of its own tunnel capability. If both tunnel endpoints are NSIS-tunnel capable, the rest of the procedures will follow those defined in Section 4.1.3. Otherwise, Texit will not initiate tunnel session reservations.

If the end-to-end session is receiver-initiated, the tunnel session is sender-initiated (Section 4.1.4), and Tentry is NSIS-tunnel capable, the operation is similar to the case where both sessions are receiver-initiated. The Tentry includes an RII object and a NODE_CHAR object with T bit set in the first end-to-end QUERY message sent toward Texit. An NSIS-tunnel capable Texit learns from the NODE_CHAR object whether Tentry is also NSIS-tunnel capable. In reply to this end-to-end QUERY message, the NSIS-tunnel capable Texit includes a NODE_CHAR object with T bit set in its RESPONSE message to notify Tentry of its own tunnel capability. If both tunnel endpoints are NSIS tunnel capable, the rest procedures follow those defined in Section 4.1.4. Otherwise, Tentry will not initiate further NSIS tunnel session reservations.

8. Other Considerations

8.1. Other Types of NSLP

This document discusses tunnel operation using QoS NSLP. It will be desirable to have the scheme 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 could indeed be similar for different types of NSLPs. For example, in the case of NATFW NSLP [13], the most important signaling operation is CREATE. Assuming Tentry is a NATFW NSLP, the tunnel handling for the CREATE operation

is expected to be very similar to the sender-initiated QoS reservation case. There are also a number of reverse directional operations in NATFW NSLP, such as RESERVE_EXTERNAL_ADDRESS and UCREATE. It is not very clear whether IP tunnel will cause problems with these messages in general. But they are likely easier to deal with than the receiver-initiated reservation case in QoS NSLP. This topic will be discussed in future version of this document if necessary.

8.2. IPSEC Flows

If the tunnel supports IPSEC (especially ESP in Tunnel-Mode with or without AH), it MAY use the flow label, DSCP 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 can be matched by the NSIS packet classifier. Virtual destination ports as in RSVP-IPSEC [17] MAY be defined for further flow demultiplexing capability at the destination side if necessary.

8.3. NSIS-tunnel Operation and Mobility

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 NSIS mobility efficiency in the tunnel is to reuse the session ID of the tunnel session when tunnel flow ID changes during mobility. This works as follows. 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 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. In this case the new FA does not have the session ID of the previous tunnel session.

When mobile IP is operating on a bi-directional tunneling mode, NSIS-tunnel operation with mobility MAY be further improved by localizing the handoff tunnel signaling process by bypassing the path between HA and CN.

General aspects of NSIS interaction with mobility are discussed in

[14].

9. 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 [18].

10. Appendix

10.1. Various Design Alternatives

10.1.1. 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 session by generating tunnel specific signaling messages. An alternative approach is to stack tunnel specific objects on top of the original end-to-end messages and make these messages visible to tunnel intermediate nodes. Thus, these new messages serve both the end-to-end session and tunnel session. This approach 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 race condition we mentioned in Section 4.2. However, we defined simple rules to solve this problem while maintaining interoperability.

This document defines the sequential and parallel modes of end-to-end and tunnel signaling interaction. There are a number of different aspects that can result in variations in carrying out the actual interaction. One aspect is the tunnel session initiation location. For example, it is possible to initiate the tunnel session from Texit, instead of Tentry as in the proposed scheme. A second aspect is the tunnel session initiation time point. For example, 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, instead of when Tentry

receives the first end-to-end QUERY message, as in the proposed scheme. The advantage of our adopted approach is that it will allow the first end-to-end QUERY message to also gather tunnel characteristics along with the rest of the end-to-end path. A third aspect is how the tunnel signaling messages are used. For example, in the case where end-to-end session is receiver initiated and tunnel session is sender initiated (Section 4.1.4), the first tunnel QUERY' message sent after receiving the end-to-end QUERY message by Tentry can be replaced by a tunnel RESERVE' message, if the application wants to trade temporary oversized or wasted (if the end-to-end reservation turns out to be unsatisfied) tunnel resource reservations for faster signaling setup delay. All these aspects are local optimization issues. We require any implementation to support the basic scheme defined in the main text of document to allow interoperability.

10.1.2. 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. Mobility changes the tunnel MRI, if at the same time the packet classification rule does not change, the common path after a handoff does not need to be updated about the packet classification, which results 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 adopts the second approach which does not pose special classification requirements on intermediate tunnel nodes.

10.1.3. Tunnel Binding Methods

In this document, the end-to-end session and its mapping 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 tunnels sessions 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 desirable 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 for both individual and aggregation tunnel sessions. Note that in this case the mobility handling inside the tunnel can still be optimized in certain situations as discussed in Section 8.3.

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 were considered.

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 requires the creation of an entirely new object, or the change of basic message headers. They are also not generic solutions that can cover other binding causes.

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 see the tunnel binding information. In option (b), every tunnel intermediate node sees the binding information. Since there will be no state for the end-to-end session in 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 desirable. Option (c) has an advantage 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 for carrying the tunnel binding object is option (a).

11. Acknowledgements

12. References

12.1. 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 Signalling Transport", draft-ietf-nsis-ntlp-15 (work in progress), February 2008.

[3] Manner, J., Karagiannis, G., and A. McDonald, "NSLP for Quality-of-Service Signaling", draft-ietf-nsis-qos-nslp-16 (work in progress), February 2008.

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

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

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

[9] Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.

[10] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

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

[12] Ash, G., Bader, A., Kappler, C., and D. Oran, "QoS NSLP QSPEC Template", draft-ietf-nsis-qspec-19 (work in progress), February 2008.

[13] Stiemerling, M., Tschofenig, H., Aoun, C., and E. Davies, "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)", draft-ietf-nsis-nslp-natfw-18 (work in progress), February 2008.

[14] Sanda, T., "Applicability Statement of NSIS Protocols in Mobile

Environments", draft-ietf-nsis-applicability-mobility-signaling-09 (work in progress), February 2008.

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

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

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

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

[19] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 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 3109
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

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).