Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 26, 2008

J. Manner
University of Helsinki
G. Karagiannis
University of Twente/Ericsson
A. McDonald
Siemens/Roke Manor Research

July 25, 2007

NSLP for Quality-of-Service Signaling
draft-ietf-nsis-qos-nslp-15.txt

Abstract

This specification describes the NSIS Signaling Layer Protocol (NSLP) for signaling QoS reservations in the Internet. It is in accordance with the framework and requirements developed in NSIS. Together with GIST, it provides functionality similar to RSVP and extends it. The QoS NSLP is independent of the underlying QoS specification or architecture and provides support for different reservation models. It is simplified by the elimination of support for multicast flows. This specification explains the overall protocol approach, design decisions made and provides examples. It specifies object, message formats and processing rules.

Table of Contents

1. Introduction

This document defines a Quality of Service (QoS) NSIS Signaling Layer Protocol (NSLP), henceforth referred to as the "QoS NSLP". This protocol establishes and maintains state at nodes along the path of a data flow for the purpose of providing some forwarding resources for that flow. It is intended to satisfy the QoS-related requirements of RFC 3726 [RFC3726]. This QoS NSLP is part of a larger suite of signaling protocols, whose structure is outlined in the NSIS framework [RFC4080]; this defines a common NSIS Transport Layer Protocol (NTLP). The abstract NTLP has been developed into a concrete protocol, GIST (General Internet Signaling Transport) [I-D.ietf-nsis-ntlp]. The QoS NSLP relies on GIST to carry out many aspects of signaling message delivery.

The design of the QoS NSLP is conceptually similar to RSVP, RFC 2205 [RFC2205], and uses soft-state peer-to-peer refresh messages as the primary state management mechanism (i.e., state installation/refresh is performed between pairs of adjacent NSLP nodes, rather than in an end-to-end fashion along the complete signaling path). The QoS NSLP extends the set of reservation mechanisms to meet the requirements of RFC 3726 [RFC3726], in particular support of sender or receiver-initiated reservations, as well as, a type of bi-directional reservation and support of reservations between arbitrary nodes, e.g., edge-to-edge, end-to-access, etc. On the other hand, there is currently no support for IP multicast.

A distinction is made between the operation of the signaling protocol and the information required for the operation of the Resource Management Function (RMF). This document describes the signaling protocol, whilst [I-D.ietf-nsis-qspec] describes the RMF-related information carried in the QSPEC (QoS Specification) object in QoS NSLP messages. This is similar to the decoupling between RSVP and the IntServ architecture, RFC 1633 [RFC1633]. The QSPEC carries information on resources available, resources required, traffic descriptions and other information required by the RMF.

This document is structured as follows. The overall protocol design is outlined in Section 3.1. The operation and use of the QoS NSLP is described in more detail in the rest of Section 3. Section 4 then clarifies the protocol by means of a number of examples. These sections should be read by people interested in the overall protocol capabilities. The functional specification in Section 5 contains more detailed object and message formats and processing rules and should be the basis for implementers. The subsequent sections describe IANA allocation issues, and security considerations.

2. Terminology

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 RFC 2119 [RFC2119].

The terminology defined by GIST [I-D.ietf-nsis-ntlp] applies to this draft.

In addition, the following terms are used:

QNE: an NSIS Entity (NE), which supports the QoS NSLP.

QNI: the first node in the sequence of QNEs that issues a reservation request for a session.

QNR: the last node in the sequence of QNEs that receives a reservation request for a session.

P-QNE: Proxy-QNE, a node set to reply to messages with the PROXY scope flag set.

Session: A session defines an association between a QNI and QNR related to a data flow. All QNEs on the path, including the QNI and QNR, use the same identifier to refer to the state stored for the association. The same QNI and QNR may have more than one session active at any one time.

Session Identification (SESSION_ID, SID): This is a cryptographically random and (probabilistically) globally unique identifier of the application layer session that is associated with a certain flow. Often there will only be one data flow for a given session, but in mobility/multihoming scenarios there may be more than one and they may be differently routed [RFC4080].

Source or message source: The one of two adjacent NSLP peers that is sending a signaling message (maybe the upstream or the downstream peer). Note that this is not necessarily the QNI.

QoS NSLP operation state: State used/kept by the QoS NSLP processing to handle messaging aspects.

QoS reservation state: State used/kept by Resource Management Function to describe reserved resources for a session.

Flow ID: This is essentially the Message Routing Information (MRI)

in GIST for path-coupled signaling.

Figure 1 shows the components that have a role in a QoS NSLP signaling session. The flow sender and receiver would in most cases be part of the QNI and QNR nodes. Yet, these may be separate nodes, too.

```
                      QoS NSLP nodes
   IP address         (QoS unaware NSIS nodes are          IP address
   = Flow              not shown)                          = Flow
   Source              |          |            |          Destination
   Address             |          |            |           Address
                       V          V            V
   +--------+  Data +------+      +------+       +------+     +--------+
   |  Flow  |-------|------|------|------|-------|------|---->|  Flow  |
   | Sender |  Flow |      |      |      |       |      |     |Receiver|
   +--------+       | QNI  |      | QNE  |       | QNR  |     +--------+
                    |      |      |      |       |      |
                    +------+      +------+       +------+
                           =====================>
                           <=====================
                                 Signaling
                                   Flow
```

Figure 1: Components of the QoS NSLP architecture

A glossary of terms and abbreviations used in this document can be found in Appendix B.

3. Protocol Overview

3.1. Overall Approach

This section presents a logical model for the operation of the QoS NSLP and associated provisioning mechanisms within a single node. The model is shown in Figure 2.

```
                                      +---------------+
                                      |     Local     |
                                      |Applications or|
                                      |Management (e.g|
                                      |for aggregates)|
                                      +---------------+
                                              ^
                                              V
                                              V
               +----------+             +----------+      +---------+
               | QoS NSLP |             | Resource |      | Policy  |
               |Processing|<<<<<<>>>>>>>|Management|<<<>>>| Control |
               +----------+             +----------+      +---------+
                 .  ^   |              *      ^
                 |  V   .            *        ^
               +----------+        *          ^
               |   NTLP   |       *           ^
               |Processing|       *           V
               +----------+       *           V
                 |      |         *           V
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                 .      .         *           V
                 |      |         *     .............................
                 .      .         *     .   Traffic Control         .
                 |      |         *     .                +---------+.
                 .      .         *     .                |Admission|.
                 |      |         *     .                | Control |.
         +----------+    +------------+ .                +---------+.
     <-.-|  Input   |    | Outgoing   |-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.->
         |  Packet  |    | Interface  |   .+----------+    +---------+.
     ===>|Processing|====| Selection  |===.|  Packet  |====| Packet  |.==>
         |          |    |(Forwarding)|   .|Classifier|     Scheduler|.
         +----------+    +------------+   .+----------+    +---------+.
                                          .............................
           <.-.-> = signaling flow
           =====> = data flow (sender --> receiver)
           <<<>>> = control and configuration operations
           ****** = routing table manipulation
```

Figure 2: QoS NSLP in a Node

This diagram shows an example implementation scenario where QoS conditioning is performed on the output interface. However, this does not limit the possible implementations. For example, in some cases traffic conditioning may be performed on the incoming interface, or it may be split over the input and output interfaces. Also, the interactions with the Policy Control component may be more complex, involving interaction with the Resource Management Function,

and the AAA infrastructure.

From the perspective of a single node, the request for QoS may result from a local application request, or from processing an incoming QoS NSLP message. The request from a local application includes not only user applications (e.g., multimedia applications) but also network

management (e.g. initiating a tunnel to handle an aggregate, or interworking with some other reservation protocol - such as RSVP) and the policy control module (e.g., for explicit teardown triggered by AAA). In this sense, the model does not distinguish between hosts and routers.

Incoming messages are captured during input packet processing and handled by GIST. Only messages related to QoS are passed to the QoS NSLP. GIST may also generate triggers to the QoS NSLP (e.g., indications that a route change has occurred). The QoS request is handled by the RMF, which coordinates the activities required to grant and configure the resource. It also handles policy-specific aspects of QoS signaling.

The grant processing involves two local decision modules, 'policy control' and 'admission control'. Policy control determines whether the user is authorized to make the reservation. Admission control determines whether the network of the node has sufficient available resources to supply the requested QoS. If both checks succeed, parameters are set in the packet classifier and in the link layer interface (e.g., in the packet scheduler) to obtain the desired QoS. Error notifications are passed back to the request originator. The resource management function may also manipulate the forwarding tables at this stage, to select (or at least pin) a route; this must be done before interface-dependent actions are carried out (including sending outgoing messages over any new route), and is in any case invisible to the operation of the protocol.

Policy control is expected to make use of the authentication infrastructure or the authentication protocols external to the node itself. Some discussion can be found in a separate document on authorization issues [qos-auth]. More generally, the processing of policy and resource management functions may be outsourced to an external node leaving only 'stubs' co-located with the NSLP node; this is not visible to the protocol operation. A more detailed discussion of authentication and authorization can be found in Section 3.1.3.

Admission control, packet scheduling, and any part of policy control beyond simple authorization have to be implemented using specific definitions for types and levels of QoS. A key assumption is made

that the QoS NSLP is independent of the QoS parameters (e.g., IntServ service elements). These are captured in a QoS Model and interpreted only by the resource management and associated functions, and are opaque to the QoS NSLP itself. QoS Models are discussed further in Section 3.1.2.

The final stage of processing for a resource request is to indicate to the QoS NSLP protocol processing that the required resources have been configured. The QoS NSLP may generate an acknowledgment message in one direction, and may forward the resource request in the other. Message routing is carried out by the GIST module. Note that while Figure 2 shows a unidirectional data flow, the signaling messages can pass in both directions through the node, depending on the particular message and orientation of the reservation.

3.1.1. Protocol Messages

The QoS NSLP uses four message types:

RESERVE: The RESERVE message is the only message that manipulates QoS NSLP reservation state. It is used to create, refresh, modify and remove such state. The result of a RESERVE message is the same whether a message is received once or many times.

QUERY: A QUERY message is used to request information about the data path without making a reservation. This functionality can be used to reservations or for support of certain QoS models. The information obtained from a QUERY may be used in the admission control process of a QNE (e.g., in case of measurement-based admission control). Note that a QUERY does not change existing reservation state.

RESPONSE: The RESPONSE message is used to provide information about the result of a previous QoS NSLP message. This includes explicit confirmation of the state manipulation signaled in the RESERVE message, the response to a QUERY message or an error code if the QNE or QNR is unable to provide the requested information or if the response is negative. The RESPONSE message does not cause any reservation state to be installed or modified.

NOTIFY: NOTIFY messages are used to convey information to a QNE. They differ from RESPONSE messages in that they are sent asynchronously and need not refer to any particular state or previously received message. The information conveyed by a NOTIFY message is typically related to error conditions. Examples would be notification to an upstream peer about state being torn down or to indicate when a reservation has been preempted.

QoS NSLP messages are sent peer-to-peer. This means that a QNE

considers its adjacent upstream or downstream peer to be the source of the each message.

Each protocol message has a common header which indicates the message type and contains various flag bits. Message formats are defined in Section 5.1.2. Message processing rules are defined in Section 5.4.

QoS NSLP messages contain three types of objects:

1. Control Information: Control information objects carry general information for the QoS NSLP processing, such as sequence numbers or whether a response is required.

2. QoS specifications (QSPECs): QSPEC objects describe the actual resources that are required and depend on the QoS model being used. Besides any resource description they may also contain other control information used by the RMF's processing.

3. Policy objects: Policy objects contain data used to authorize the reservation of resources.

Object formats are defined in Section 5.1.3. Object processing rules are defined in Section 5.3.

3.1.2. QoS Models and QoS Specifications

The QoS NSLP provides flexibility over the exact patterns of signaling messages that are exchanged. The decoupling of QoS NSLP and QSPEC allows the QoS NSLP to be ignorant about the ways in which traffic, resources, etc. are described, and it can treat the QSPEC as an opaque object. Various QoS models can be designed, and these do not affect the specification of the QoS NSLP protocol. Only the RMF specific to a given QoS model will need to interpret the QSPEC. The Resource Management Function (RMF) reserves resources for each flow.

An ongoing effort attempts to specify a QSPEC template [I-D.ietf-nsis-qspec]. The QSPEC template contains object formats for generally useful elements of the QoS description, which is designed to ensure interoperability when using the basic set of objects.

The QSPEC fulfills a similar purpose to the TSpec, RSpec and AdSpec objects used with RSVP and specified in RFC 2205 [RFC2205] and RFC 2210 [RFC2210]. At each QNE, the content of the QSPEC is interpreted by the Resource Management Function and the Policy Control Function for the purposes of traffic and policy control (including admission control and configuration of the packet classifier and scheduler).

The QoS NSLP does not mandate any particular behavior for the RMF, instead providing interoperability at the signaling protocol level whilst leaving the validation of RMF behavior to contracts external to the protocol itself. The RMF may make use of various elements from the QoS NSLP message, not only the QSPEC object.

Still, this specification assumes that resource sharing is possible between flows with the same SESSION_ID that originate from the same QNI or between flows with a different SESSION_ID that are related through the BOUND_SESSION_ID object. For flows with the same SESSION_ID, resource sharing is only applicable when the existing reservation is not just replaced (which is indicated by the REPLACE flag in the common header). We assume that the QoS model supports resource sharing between flows. A QoS Model may elect to implement a more general behavior of supporting relative operations on existing reservations, such as ADDING or SUBTRACTING a certain amount of resources from the current reservation. A QoS Model may also elect to allow resource sharing more generally, e.g., between all flows with the same DSCP.

The QSPEC carries a collection of objects that can describe QoS specifications in a number of different ways. A generic template is defined in [I-D.ietf-nsis-qspec]. A QSPEC describing the resources requested will usually contain objects which need to be understood by all implementations, and it can also be enhanced with additional objects specific to a QoS model to provide a more exact definition to the RMF, which may be better able to use its specific resource management mechanisms (which may, e.g., be link specific) as a result.

A QoS Model defines the behavior of the RMF, including inputs and outputs, and how QSPEC information is used to describe resources available, resources required, traffic descriptions, and control information required by the RMF. A QoS Model also describes the minimum set of parameters QNEs should use in the QSPEC when signaling about this QoS Model.

QoS Models may be local (private to one network), implementation/ vendor specific, or global (implementable by different networks and vendors). All QSPECs should follow the design of the QSPEC template [I-D.ietf-nsis-qspec].

The definition of a QoS model may also have implications on how local behavior should be implemented in the areas where the QoS NSLP gives freedom to implementers. For example, it may be useful to identify recommended behavior for how a RESERVE message that is forwarded relates to that received, or when additional signaling sessions should be started based on existing sessions, such as required for

aggregate reservations. In some cases, suggestions may be made on whether state that may optionally be retained should be held in particular scenarios. A QoS model may specify reservation preemption, e.g., an incoming resource request may cause removal of an earlier established reservation.

3.1.3. Policy Control

Getting access to network resources, e.g., network access in general or access to QoS, typically involves some kind of policy control. One example of this is authorization of the resource requester. Policy control for QoS NSLP resource reservation signaling is conceptually organized as illustrated below in Figure 3.

```
                               +-------------+
                               | Policy      |
                               | Decision    |
                               | Point (PDP) |
                               +------+------+
                                      |
                              /-\-----+-----/\
                          ////                \\\\
                        ||                        ||
                       |      Policy transport      |
                        ||                        ||
                          \\\\                ////
                              \-------+------/
                                      |
+-------------+ QoS signaling  +------+------+
|  Entity     |<==============>| QNE = Policy|<=========>
|  requesting | Data Flow      | Enforcement |
|  resource   |----------------|-Point (PEP)-|---------->
+-------------+                +-------------+
```

Figure 3: Policy control with the QoS NSLP signaling

From the QoS NSLP point of view, the policy control model is essentially a two-party model between neighboring QNEs. The actual policy decision may depend on the involvement of a third entity (the policy decision point, PDP), but this happens outside of the QoS NSLP protocol by means of existing policy infrastructure (COPS, Diameter, etc). The policy control model for the entire end-to-end chain of QNEs is therefore one of transitivity, where each of the QNEs exchanges policy information with its QoS NSLP policy peer.

The authorization of a resource request often depends on the identity of the entity making the request. Authentication may be required. The GIST channel security mechanisms provide one way of

authenticating the QoS NSLP peer which sent the request, and so may be used in making the authorization decision.

Additional information might also be provided in order to assist in making the authorization decision. This might include alternative methods of authenticating the request.

The QoS NSLP does not contain objects to carry authorization information. A separate work [nslp-auth] defines this functionality for the QoS NSLP and the NATFW NSLP.

It is generally assumed that policy enforcement is likely to concentrate on border nodes between administrative domains. This may mean that nodes within the domain are "Policy Ignorant Nodes" that perform no per-request authentication or authorization, relying on the border nodes to perform the enforcement. In such cases, the policy management between ingress and egress edge of a domain relies on the internal chain of trust between the nodes in the domain. If this is not acceptable, a separate signaling session can be set up between the ingress and egress edge nodes in order to exchange policy information.

3.2. Design Background

This section presents some of the key functionality behind the specification of the QoS NSLP.

3.2.1. Soft States

The NSIS protocol suite takes a soft-state approach to state management. This means that reservation state in QNEs must be periodically refreshed. The frequency with which state installation is refreshed is expressed in the REFRESH_PERIOD object. This object contains a value in milliseconds indicating how long the state that is signaled for remains valid. Maintaining the reservation beyond this lifetime can be done by sending a RESERVE message periodically.

3.2.2. Sender and Receiver Initiation

The QoS NSLP supports both sender-initiated and receiver-initiated reservations. For a sender-initiated reservation, RESERVE messages travel in the same direction as the data flow that is being signaled for (the QNI is at the side of the source of the data flow). For a receiver-initiated reservation, RESERVE messages travel in the opposite direction (the QNI is at the side of the receiver of the data flow).

Note: these definitions follow the definitions in Section 3.3.1. of

RFC 4080 [RFC4080]. The main issue is, which node is in charge of requesting and maintaining the resource reservation. In a receiver-initiated reservation, even though the sender sends the initial QUERY, the receiver is still in charge of making the actual resource request, and maintaining the reservation.

3.2.3. Protection Against Message Re-ordering and Duplication

RESERVE messages affect the installed reservation state. Unlike NOTIFY, QUERY and RESPONSE messages, the order in which RESERVE messages are received influences the eventual reservation state that will be stored at a QNE, that is, the most recent RESERVE message replaces the current reservation. Therefore, in order to protect against RESERVE message re-ordering or duplication, the QoS NSLP uses a Reservation Sequence Number (RSN). The RSN has local significance only, i.e., between a QNE and its downstream peers.

3.2.4. Explicit Confirmations

A QNE may require a confirmation that the end-to-end reservation is in place, or a reply to a query along the path. For such requests, it must be able to keep track of which request each response refers to. This is supported by including a Request Identification Information (RII) object in a QoS NSLP message.

3.2.5. Reduced Refreshes

For scalability, the QoS NSLP supports an abbreviated form of refresh RESERVE message. In this case, the refresh RESERVE references the reservation using the RSN and the SESSION_ID, and does not include the full reservation specification (including QSPEC). By default state refresh should be performed with reduced refreshes in order to save bytes during transmission. Stateless QNEs will require full refresh since they do not store the whole reservation information.

If the stateful QNE does not support reduced refreshes, or there is a mismatch between the local and received RSN, the stateful QNE must reply with an RESPONSE carrying an INFO_SPEC indicating the error. Furthermore, the QNE must stop sending reduced refreshes to this peer if the error indicates lacking support for this feature.

3.2.6. Summary Refreshes and Summary Tear

For limiting the number of individual messages, the QoS NSLP supports a summary refresh and summary tear messages. When sending a refreshing RESERVE for a certain (primary) session, a QNE may include a SESSION_ID_LIST object where the QNE indicates (secondary) sessions that are also refreshed. An RSN_LIST object must also be added. The

SESSION IDs and RSNs are stacked in the objects such that the index in both stacks refer to the same reservation state, i.e., the SESSION_ID and RSN at index i in both objects refers to the same session. If the receiving stateful QNE notices unknown SESSION IDs or a mismatch with RSNs for a session, it will reply back to the upstream stateful QNE with an error.

In order to tear down several sessions at once, a QNE may include SESSION_ID_LIST and RSN_LIST objects in a tearing reserve. The downstream stateful QNE must then also tear the other sessions indicated. The downstream stateful QNE must silently ignore any unknown SESSION IDs.

GIST provides a SII_HANDLE for every downstream session. The SII_HANDLE identifies a peer, and should be the same for all sessions whose downstream peer is the same. The QoS NSLP uses this information to decide whether summary refresh messages can be sent, or when a summary tear is possible.

3.2.7. Message Scoping

A QNE may use local policy when deciding whether to propagate a message or not. For example, the local policy can define/configure that a QNE is, for a particular session, a QNI and/or a QNR. The QoS NSLP also includes an explicit mechanism to restrict message propagation by means of a scoping mechanism.

For a RESERVE or a QUERY message, two scoping flags limit the part of the path on which state is installed on the downstream nodes that can respond. When the SCOPING flag is set to zero, it indicates that the scope is "whole path" (default). When set to one, the scope is "single hop". When the PROXY scope flag is set, the path is terminated at a pre-defined Proxy QNE (P-QNE). This is similar to the Localized RSVP [lrsvp].

The propagation of a RESPONSE message is limited by the RII object, which ensures that it is not forwarded back along the path further than the node that requested the RESPONSE.

3.2.8. Session Binding

Session binding is defined as the enforcement of a relation between different QoS NSLP sessions (i.e., signaling flows with different SESSION_ID (SID) as defined in GIST [I-D.ietf-nsis-ntlp]).

Session binding indicates a unidirectional dependency relation between two or more sessions by including a BOUND_SESSION_ID object. A session with SID_A (the binding session) can express its

unidirectional dependency relation to another session with SID_B (the bound session) by including a BOUND_SESSION_ID object containing SID_B in its messages.

The concept of session binding is used to indicate the unidirectional dependency relation between the end-to-end session and the aggregate session in case of aggregate reservations. In case of bidirectional reservations, it is used to express the unidirectional dependency relation between the sessions used for forward and reverse reservation. Typically, the dependency relation indicated by session binding is purely informative in nature and does not automatically trigger any implicit action in a QNE. A QNE may use the dependency relation information for local resource optimization or to explicitly tear down reservations that are no longer useful. However, by using an explicit binding code, see Section 5.1.3.4, it is possible to formalise this dependency relation, meaning that if the bound session (e.g., session with SID_B) is terminated also the binding session (e.g., the session with SID_A) must be terminated.

A message may include more than one BOUND_SESSION_ID object. This may happen, e.g., in certain aggregation and bi-directional reservation scenarios, where an end-to-end session has an unidirectional dependency relation with an aggregate session and at the same time it has an unidirectional dependency relation with another session used for the reverse path.

3.2.9. Message Binding

QoS NSLP supports binding of messages in order to allow for expressing dependencies between different messages. The message binding can indicate either a unidirectional or bidirectional dependency relation between two messages by including in one of the message the MSG_ID object ("binding message") and in the other message ("bound message") the BOUND_MSG_ID object. The unidirectional dependency means that only RESERVE messages are bound to each other whereas a bidirectional dependency means that there is also a dependency for the related RESPONSE messages. The message binding can be used to speed up signaling by starting two signaling exchanges simultaneously that are synchronized later by using message IDs. This can be used as an optimization technique for example in scenarios where aggregate reservations are used. Section 4.6 provides more details.

3.2.10. Layering

The QoS NSLP supports layered reservations. Layered reservations may occur when certain parts of the network (domains) implement one or more local QoS models, or when they locally apply specific transport

characteristics (e.g., GIST unreliable transfer mode instead of reliable transfer mode). They may also occur when several per-flow reservations are locally combined into an aggregate reservation.

3.2.10.1. Local QoS Models

A domain may have local policies regarding QoS model implementation, i.e., it may map incoming traffic to its own locally defined QoS models. The QSPEC allows this functionality, and the operation is transparent to the QoS NSLP. The use of local QoS models within a domain is performed in the RMF.

3.2.10.2. Local Control Plane Properties

The way signaling messages are handled is mainly determined by the parameters that are sent over GIST-NSLP API and by the domain internal configuration. A domain may have a policy to implement local transport behavior. It may, for instance, elect to use an unreliable transport locally in the domain while still keeping end-to-end reliability intact.

The QoS NSLP supports this situation by allowing two sessions to be set up for the same reservation. The local session has the desired local transport properties and is interpreted in internal QNEs. This solution poses two requirements: the end-to-end session must be able to bypass intermediate nodes and the egress QNE needs to bind both sessions together. Bypassing intermediate nodes is achieved with GIST. The local session and the end-to-end session are bound at the egress QNE by means of the BOUND_SESSION_ID object.

3.2.10.3. Aggregate Reservations

In some cases it is desirable to create reservations for an aggregate, rather than on a per-flow basis, in order to reduce the amount of reservation state needed, as well as, the processing load for signaling messages. Note that the QoS NSLP does not specify how reservations need to be combined in an aggregate or how end-to-end properties need to be computed but only provides signaling support for it.

The essential difference with the layering approaches described in Section 3.2.10.1 and Section 3.2.10.2 is that the aggregate reservation needs a MRI that describes all traffic carried in the aggregate (e.g., a DSCP in case of IntServ over DiffServ). The need for a different MRI mandates the use of two different sessions, similar to Section 3.2.10.3 and to the RSVP aggregation solution in RFC 3175 [RFC3175].

Edge QNEs of the aggregation domain that want to maintain some end-to-end properties may establish a peering relation by sending the end-to-end message transparently over the domain (using the intermediate node bypass capability described above). Updating the end-to-end properties in this message may require some knowledge of the aggregated session (e.g., for updating delay values). For this purpose, the end-to-end session contains a BOUND_SESSION_ID carrying the SESSION_ID of the aggregate session.

3.2.11. Support for Request Priorities

This specification acknowledges the fact that in some situations, some messages or some reservations may be more important than others and therefore foresees mechanisms to give these messages or reservations priority.

Priority of certain signaling messages over others may be required in mobile scenarios when a message loss during call set-up is less harmful than during handover. This situation only occurs when GIST or QoS NSLP processing is the congested part or scarce resource.

Priority of certain reservations over others may be required when QoS resources are oversubscribed. In that case, existing reservations may be preempted in order to make room for new higher-priority reservations. A typical approach to deal with priority and preemption is through the specification of a setup priority and holding priority for each reservation. The resource management function at each QNE then keeps track of the resource consumption at each priority level. Reservations are established when resources, at their setup priority level, are still available. They may cause preemption of reservations with a lower holding priority than their setup priority.

Support of reservation priority is a QSPEC parameter and therefore outside the scope of this specification. The GIST specification provides a mechanism to support a number of levels of message priority that can be requested over the NSLP-GIST API.

3.2.12. Rerouting

The QoS NSLP needs to adapt to route changes in the data path. This assumes the capability to detect rerouting events, create a QoS reservation on the new path and optionally tear down reservations on the old path.

From an NSLP perspective, rerouting detection can be performed in two ways. It can either come through NetworkNotification from GIST, or from information seen at the NSLP. In the latter case, the QoS NSLP

node is able to detect changes in its QoS NSLP peers by keeping track of a Source Identification Information (SII) handle that provides information similar in nature to the RSVP_HOP object described in RFC 2205 [RFC2205]. When a RESERVE message with an existing SESSION_ID and a different SII is received, the QNE knows its upstream or downstream peer has changed, for sender-oriented and receiver-oriented reservations, respectively.

Reservation on the new path happens when a RESERVE message arrives at the QNE beyond the point where the old and new paths diverge. If the QoS NSLP suspects that a reroute has occurred, then a full RESERVE message (including the QSPEC) would be sent. A refreshing RESERVE (with no QSPEC) will be identified as an error by a QNE on the new path which does not have the reservation installed (i.e. it was not on the old path) or which previously had a different previous-hop QNE. It will send back an error message which results in a full RESERVE message being sent. Rapid recovery at the NSLP layer therefore requires short refresh periods. Detection before the next RESERVE message arrives is only possible at the IP layer or through monitoring of GIST peering relations (e.g., by TTL counting the number of GIST hops between NSLP peers or the observing changes in the outgoing interface towards GIST peer). These mechanisms can provide implementation specific optimizations, and are outside the scope of this specification.

When the QoS NSLP is aware of the route change, it needs to set up the reservation on the new path. This is done by sending a new RESERVE message. If the next QNE is, in fact, unchanged then this will be used to refresh/update the existing reservation. Otherwise it will lead to the reservation being installed on the new path.

Note that the operation for a receiver-initiated reservation session differs a bit from the above description. If the routing changes in the middle of the session, the QNE that notices that its downstream path changed, the divergence point, must send a QUERY with the R-flag downstream. It will be processed as above, and at some point hits a QNE for which the path downstream towards the QNI remains (the convergence point). This node must then send a full RESERVE upstream to set up the reservation state along the new path. It should not send the QUERY further downstream, since this would have no real use. Similarly, when the QNE that sent the QUERY receives the RESERVE, it should not send the RESERVE further upstream.

After the reservation on the new path is set up, the branching node may want to tear down the reservation on the old path (sooner than would result from normal soft-state time-out). This functionality is supported by keeping track of the old SII-Handle provided over the GIST API. This handle can be used by the QoS NSLP to route messages

explicitly to the next node.

If the old path is downstream, the QNE can send a tearing RESERVE using the old SII-Handle. If the old path is upstream, the QNE can send a NOTIFY with the code for "Route Change". This is forwarded upstream until it hits a QNE that can issue a tearing RESERVE downstream. A separate document discusses in detail the effect of mobility on the QoS NSLP signaling [I-D.ietf-nsis-applicability-mobility-signaling].

A QNI or a branch node may wish to keep the reservation on the old branch. This could for instance be the case when a mobile node has experienced a mobility event and wishes to keep reservation to its old attachment point in case it moves back there. For this purpose, a REPLACE flag is provided in the QoS NSLP common header, which, when not set, indicates that the reservation on the old branch should be kept.

Note that keeping old reservations affects the resources available to other nodes. Thus, the operator of the access network must make the final decision on whether this behavior is allowed. Also, the QNEs in the access network may add this flag even if the mobile node has not used the flag initially.

The design of the QoS NSLP allows reservations to be installed at a subset of the nodes along a path. In particular, usage scenarios include cases where the data flow endpoints do not support the QoS NSLP.

3.2.12.1. Last Node Behavior

In the case where the data flow receiver does not support the QoS NSLP, some particular considerations must be given to node discovery and rerouting at the end of the signaling path.

There are three cases for the last node on the signaling path: 1) Last node is the data receiver 2) Last node is a configured proxy for the data receiver 3) Last node is not the data receiver and is not explicitly configured to act as a signaling proxy on behalf of the data receiver.

Cases (1) and (2) can be handled by the QoS NSLP itself during the initial path setup, since the QNE knows that it should terminate the signaling. Case (3) requires some assistance from GIST which provides messages across the API to indicate that no further QoS NSLP supporting GIST nodes are present downstream, and downstream route change probing needs to continue once the reservation is installed to detect any changes in this situation.

Two particular scenarios need to be considered in this third case. In the first, referred to as "Path Extension", rerouting occurs such that an additional QNE is inserted into the signaling path between the old last node and the data receiver, as shown in Figure 4.

```
        /-------\   Initial route
       /         v
           /-\
        /--|B|--\               +-+
       /   \-/   \              |x| = QoS NSLP aware
    +-+           /-\           +-+
----|A|           |D|
    +-+           \-/           /-\
       \   +-+   /              |x| = QoS NSLP unaware
        \--|C|--/               \-/
           +-+
       \         ^
        \-------/   Updated route
```

Figure 4: Path Extension

When rerouting occurs, the data path changes from A-B-D to A-C-D. Initially the signaling path ends at A. Despite initially being the last node, node A needs to continue to attempt to send messages downstream to probe for path changes, unless it has been explicitly configured as a signaling proxy for the data flow receiver. This is required so that the signaling path change is detected, and C will become the new last QNE.

In a second case, referred to as "Path Truncation", rerouting occurs such that the QNE that was the last node on the signaling path is no longer on the data path. This is shown in Figure 5.

```
        /-------\   Initial route
       /         v
           +-+
        /--|B|--\                +-+
       /   +-+   \               |x| = QoS NSLP aware
    +-+           /-\            +-+
----|A|           |D|
    +-+           \-/            /-\
       \   /-\   /               |x| = QoS NSLP unaware
        \--|C|--/                \-/
           \-/
       \         ^
        \-------/   Updated route
```

Figure 5: Path Truncation

When rerouting occurs, the data path again changes from A-B-D to A-C-D. The signaling path initially ends at B, but this node is not on the new path. In this case, the normal GIST path change detection procedures at A will detect the path change and notify the QoS NSLP. GIST will also notify the signaling application that no downstream GIST nodes supporting the QoS NSLP are present. Node A will take over as the last node on the signaling path.

3.2.12.2. Handling Spurious Route Change Notifications

The QoS NSLP is notified by GIST (with the NetworkNotification primitive) when GIST believes that a rerouting event may have occurred. In some cases, events that are detected as possible route changes will turn out not to be. The QoS NSLP will not always be able to detect this, even after receiving messages from the 'new' peer.

As part of the RecvMessage API primitive, GIST provides an SII-Handle which can be used by the NSLP to direct a signaling message to a particular peer. The current SII-Handle will change if the signaling peer changes. However, it is not guaranteed to remain the same after a rerouting event where the peer does not change. Therefore, the QoS NSLP mechanism for reservation maintenance after a route change includes robustness mechanisms to avoid accidentally tearing down a reservation in situations where the peer QNE has remained the same after a 'route change' notification from GIST.

A simple example that illustrates the problem is shown in Figure 6 below.

```
        (1)                         +-+
      /-----\                       |x| = QoS NSLP aware
    +-+     /-\ (3) +-+             +-+
----|A|     |B|-----|C|----
    +-+     \-/     +-+             /-\
      \-----/                       |x| = QoS NSLP unaware
        (2)                         \-/
```

Figure 6: Spurious reroute alerting

In this example the initial route A-B-C uses links (1) and (3). After link (1) fails, the path is rerouted using links (2) and (3). The set of QNEs along the path is unchanged (it is A-C in both cases, since B does not support the QoS NSLP).

When the outgoing interface at A has changes, GIST may signal across its API to the NSLP with a NetworkNotification. The QoS NSLP at A will then attempt to repair the path by installing the reservation on

the path (2),(3). In this case, however, the old and new paths are the same.

To install the new reservation A will send a RESERVE message, which GIST will transport to C (discovering the new next peer as appropriate). The RESERVE also requests a RESPONSE from the QNR. When this RESERVE message is received through the RecvMessage API call from GIST at the QoS NSLP at C, the SII-Handle will be unchanged from its previous communications from A.

A RESPONSE message will be sent by the QNR, and be forwarded from C to A. This confirms that the reservation was installed on the new path. The SII-Handle passed with the RecvMessage call from GIST to the QoS NSLP will be different to that seen previously, since the interface being used on A has changed.

At this point A can attempt to tear down the reservation on the old path. The RESERVE message with the TEAR flag set is sent down the old path by using the GIST explicit routing mechanism and specifying the SII-Handle relating to the 'old' peer QNE.

If RSNs were being incremented for each of these RESERVE and RESERVE-with-TEAR messages the reservation would be torn down at C and any QNEs further along the path. To avoid this the RSN is used in a special way. The RESERVE down the new path is sent with the new current RSN set to the old RSN plus 2. The RESERVE-with-TEAR down the old path is sent with an RSN set to the new current RSN minus 1. This is the peer from which it was receiving RESERVE messages.

3.2.13. Pre-emption

The QoS NSLP provides building blocks to implement pre-emption. This specification does not define how pre-emption should work, but only provides signaling mechanisms that can be used by QoS Models. For example, an INFO_SPEC object can be added to messages to indicate that the signaled session was pre-empted. A BOUND_SESSION_ID object can carry the Session ID of the flow that caused the pre-emption to happen for the signaled session. How these are used by QoS Models is out of scope of the QoS NSLP specification.

3.3. GIST Interactions

The QoS NSLP uses GIST for delivery of all its messages. Messages are passed from the NSLP to GIST via an API (defined in Appendix B of [I-D.ietf-nsis-ntlp]), which also specifies additional information, including an identifier for the signaling application (e.g., 'QoS NSLP'), session identifier, MRI, and an indication of the intended direction - towards data sender or receiver. On reception, GIST

provides the same information to the QoS NSLP. In addition to the NSLP message data itself, other meta-data (e.g. session identifier and MRI) can be transferred across this interface.

The QoS NSLP keeps message and reservation state per session. A session is identified by a Session Identifier (SESSION_ID). The SESSION_ID is the primary index for stored NSLP state and needs to be constant and unique (with a sufficiently high probability) along a path through the network. The QoS NSLP picks a value for Session-ID.

This value is subsequently used by GIST and the QoS NSLP to refer to this session.

Currently, the QoS NSLP specification considers mainly the path-coupled MRM. However, extensions may specify how other types of MRMs may be applied in combination with the QoS NSLP.

When GIST passes the QoS NSLP data to the NSLP for processing, it must also indicate the value of the 'D' (Direction) flag for that message in the MRI.

The QoS NSLP does not provide any method of interacting with firewalls or Network Address Translators (NATs). It assumes that a basic NAT traversal service is provided by GIST.

3.3.1. Support for Bypassing Intermediate Nodes

The QoS NSLP may want to restrict the handling of its messages to specific nodes. This functionality is needed to support layering (explained in Section 3.2.10), when only the edge QNEs of a domain process the message. This requires a mechanism at GIST level (which can be invoked by the QoS NSLP) to bypass intermediate nodes between the edges of the domain.

The intermediate nodes are bypassed using multiple levels of the router alert option. In that case, internal routers are configured to handle only certain levels of router alerts. This is accomplished by marking the signaling messages, i.e., modifying the QoS NSLP default NSLP-ID value to another NSLP-ID predefined value. The marking is accomplished by the ingress edge by modifying the QoS NSLP default NSLP-ID value to a NSLP-ID predefined value, see Section 6.6. The egress stops this marking process by reassigning the QoS NSLP default NSLP-ID value to the original RESERVE message. The exact operation of modifying the NSLP-ID must be specified in the relevant QoS model specification.

3.3.2. Support for Peer Change Identification

There are several circumstances where it is necessary for a QNE to identify the adjacent QNE peer, which is the source of a signaling application message; e.g., it may be to apply the policy that "state can only be modified by messages from the node that created it" or it might be that keeping track of peer identity is used as a (fallback) mechanism for rerouting detection at the NSLP layer.

This functionality is implemented in GIST service interface with SII-handle. As shown in the above example, we assume the SII- handling will support both own SII and peer SII.

Keeping track of the SII of a certain reservation also provides a means for the QoS NSLP to detect route changes. When a QNE receives a RESERVE referring to existing state but with a different SII, it knows that its upstream peer has changed. It can then use the old SII to initiate a teardown along the old section of the path. This functionality is supported in GIST service interface when the peer's SII which is stored on message reception is passed to GIST upon message transmission.

3.3.3. Support for Stateless Operation

Stateless or reduced state QoS NSLP operation makes the most sense when some nodes are able to operate in a stateless way at GIST level as well. Such nodes should not worry about keeping reverse state, message fragmentation and reassembly (at GIST), congestion control or security associations. A stateless or reduced state QNE will be able to inform the underlying GIST of this situation. GIST service interface supports this functionality with the Retain-State attribute in the MessageReceived primitive.

3.3.4. Priority of Signaling Messages

The QoS NSLP will generate messages with a range of performance requirements for GIST. These requirements may result from a prioritization at the QoS NSLP (Section 3.2.11) or from the responsiveness expected by certain applications supported by the QoS NSLP. GIST service interface supports this with the 'priority' transfer attribute.

3.3.5. Knowledge of Intermediate QoS NSLP Unaware Nodes

In some cases it is useful to know that there are routers along the path where QoS cannot be provided. The GIST service interface supports this by keeping track of IP-TTL and Original-TTL in the RecvMessage primitive. A difference between the two indicates the

number of QoS NSLP unaware nodes. In this case the QNE that detects this difference should set the "B" (BREAK) flag. If a QNE generates a QUERY, RESERVE or RESPONSE message, after receiving a QUERY or RESERVE message with a "Break" flag set, it can set the "B" (BREAK) flag in these messages. There are however, situations where the egress QNE in a local domain may have some other means to provide QoS [I-D.ietf-nsis-qspec]. For example, in an RMD-QOSM [I-D.ietf-nsis-rmd] (or RMD-QOSM like) aware local domain that uses either NTLP stateless nodes or NSIS unaware nodes the end to end RESERVE or QUERY message bypasses these NTLP stateless or NSIS unaware nodes. However, the reservation within the local domain can be signaled by the RMD-QOSM (or RMD-QOSM like QOSM). In such situations, the "B" (BREAK) flag in the end to end RESERVE or QUERY message should not be set by the edges of the local domain.

4. Examples of QoS NSLP Operation

The QoS NSLP can be used in a number of ways. The examples given here give an indication of some of the basic processing. However, they are not exhaustive and do not attempt to cover the details of the protocol processing.

4.1. Sender-initiated Reservation

```
QNI        QNE        QNE        QNR
 |          |          |          |
 | RESERVE  |          |          |
 +--------->|          |          |
 |          | RESERVE  |          |
 |          +--------->|          |
 |          |          | RESERVE  |
 |          |          +--------->|
 |          |          |          |
 |          |          | RESPONSE |
 |          |          |<---------+
 |          | RESPONSE |          |
 |          |<---------+          |
 | RESPONSE |          |          |
 |<---------+          |          |
 |          |          |          |
 |          |          |          |
```

Figure 7: Basic Sender Initiated Reservation

To make a new reservation, the QNI constructs a RESERVE message containing a QSPEC object, from its chosen QoS model, which describes the required QoS parameters.

The RESERVE message is passed to GIST which transports it to the next QNE. There it is delivered to the QoS NSLP processing which examines the message. Policy control and admission control decisions are made. The exact processing also takes into account the QoS model being used. The node performs appropriate actions (e.g., installing reservation) based on the QSPEC object in the message.

The QoS NSLP then generates a new RESERVE message (usually based on the one received). This is passed to GIST, which forwards it to the next QNE.

The same processing is performed at further QNEs along the path, up to the QNR. The determination that a node is the QNR may be made directly (e.g., that node is the destination for the data flow), or using GIST functionality to determine that there are no more QNEs between this node and the data flow destination.

Any node may include a request for a RESPONSE in its RESERVE messages. It does so by including a Request Identification Information (RII) object in the RESERVE message. If the message already includes an RII, an interested QNE must not add a new RII object nor replace the old RII object. Instead it needs to remember the RII value so that it can match a RESPONSE message belonging to the RESERVE. When it receives the RESPONSE, it forwards the RESPONSE upstream towards the RII originating node.

In this example, the RESPONSE message is forwarded peer-to-peer along the reverse of the path that the RESERVE message took (using GIST path state), and so is seen by all the QNEs on this segment of the path. It is only forwarded as far as the node which requested the RESPONSE originally.

The reservation can subsequently be refreshed by sending further RESERVE messages containing the complete reservation information, as for the initial reservation. The reservation can also be modified in the same way, by changing the QSPEC data to indicate a different set of resources to reserve.

The overhead required to perform refreshes can be reduced, in a similar way to that proposed for RSVP in RFC 2961 [RFC2961]. Once a RESPONSE message has been received indicating the successful installation of a reservation, subsequent refreshing RESERVE messages can simply refer to the existing reservation, rather than including the complete reservation specification.

4.2. Sending a Query

QUERY messages can be used to gather information from QNEs along the path. For example, they can be used to find out what resources are available before a reservation is made.

In order to perform a query along a path, the QNE constructs a QUERY message. This message includes a QSPEC containing the actual query to be performed at QNEs along the path. It also contains an RII object used to match the response back to the query, and an indicator of the query scope (next node, whole path, proxy). The QUERY message is passed to GIST to forward it along the path.

A QNE receiving a QUERY message should inspect it and create a new message, based on that received with the query objects modified as required. For example, the query may request information on whether a flow can be admitted, and so a node processing the query might record the available bandwidth. The new message is then passed to GIST for further forwarding (unless it knows it is the QNR, or is the limit for the scope in the QUERY).

At the QNR, a RESPONSE message must be generated if the QUERY message includes a Request Identification Information (RII) object. Various objects from the received QUERY message have to be copied into the RESPONSE message. It is then passed to GIST to be forwarded peer-to-peer back along the path.

Each QNE receiving the RESPONSE message should inspect the RII object to see if it 'belongs' to it (i.e., it was the one that originally created it). If it does not then it simply passes the message back to GIST to be forwarded upstream.

If there was an error in processing a RESERVE, instead of an RII, the RESPONSE may carry an RSN. Thus, a QNE must also be prepared to look for an RSN object if no RII was present, and act based on the error code set in the INFO_SPEC of the RESPONSE.

4.3. Basic Receiver-initiated Reservation

As described in the NSIS framework [RFC4080] in some signaling applications, a node at one end of the data flow takes responsibility for requesting special treatment - such as a resource reservation - from the network. Both ends then agree whether sender or receiver-initiated reservation is to be done. In case of a receiver initiated reservation, both ends agree whether a "One Pass With Advertising" (OPWA) [opwa95] model is being used. This negotiation can be accomplished using mechanisms that are outside the scope of NSIS.

To make a receiver-initiated reservation, the QNR constructs a QUERY message, which may contain a QSPEC object from its chosen QoS model (see Figure 8). The QUERY must have the RESERVE-INIT flag set. This QUERY message does not need to trigger a RESPONSE message and therefore, the QNI must not include the RII object (Section 5.4.2) in the QUERY message. The QUERY message may be used to gather information along the path, which is carried by the QSPEC object. An example of such information is the "One Pass With Advertising" (OPWA) [opwa95]. This QUERY message causes GIST reverse-path state to be installed.

```
 QNR        QNE        QNE        QNI
sender                          receiver
  |          |          |          |
  | QUERY    |          |          |
  +--------->|          |          |
  |          | QUERY    |          |
  |          +--------->|          |
  |          |          | QUERY    |
  |          |          +--------->|
  |          |          |          |
  |          |          | RESERVE  |
  |          |          |<---------+
  |          | RESERVE  |          |
  |          |<---------+          |
  | RESERVE  |          |          |
  |<---------+          |          |
  |          |          |          |
  | RESPONSE |          |          |
  +--------->|          |          |
  |          | RESPONSE |          |
  |          +--------->|          |
  |          |          | RESPONSE |
  |          |          +--------->|
  |          |          |          |
```

Figure 8: Basic Receiver Initiated Reservation

The QUERY message is transported by GIST to the next downstream QoS NSLP node. There it is delivered to the QoS NSLP processing which examines the message. The exact processing also takes into account the QoS model being used and may include gathering information on path characteristics that may be used to predict the end-to-end QoS.

The QNE generates a new QUERY message (usually based on the one received). This is passed to GIST, which forwards it to the next QNE. The same processing is performed at further QNEs along the path, up to the flow receiver. The receiver detects that this QUERY

message carries the RESERVE-INIT flag and by using the information contained in the received QUERY message, such as the QSPEC, constructs a RESERVE message.

The RESERVE is forwarded peer-to-peer along the reverse of the path that the QUERY message took (using GIST reverse path state). Similar to the sender-initiated approach, any node may include an RII in its RESERVE messages. The RESPONSE is sent back to confirm the resources are set up. The reservation can subsequently be refreshed with RESERVE messages in the upstream direction.

4.4. Bidirectional Reservations

The term "bidirectional reservation" refers to two different cases that are supported by this specification:

o Binding two sender-initiated reservations together, e.g., one sender-initiated reservation from QNE A to QNE B and another one from QNE B to QNE A.

o Binding a sender-initiated and a receiver-initiated reservation together, e.g., a sender-initiated reservation from QNE A towards QNE B, and a receiver-initiated reservation from QNE A towards QNE B for the data flow in the opposite direction (from QNE B to QNE A). This case is particularly useful when one end of the communication has all required information to set up both sessions.

Both ends have to agree on which bi-directional reservation type they need to use. This negotiation/agreement can be accomplished using mechanisms that are outside the scope of NSIS.

The scenario with two sender-initiated reservations is shown in Figure 9. Note that RESERVE messages for both directions may visit different QNEs along the path because of asymmetric routing. Both directions of the flows are bound by inserting the BOUND_SESSION_ID object at the QNI and QNR. RESPONSE messages are optional and not shown in the picture for simplicity.

```
      A          QNE        QNE        B
      |           |  FLOW-1  |          |
      |===============================>|
      |RESERVE-1 |           |          |
   QNI+--------->|RESERVE-1 |          |
      |           +-------------------->|QNR
      |           |           |          |
      |           |  FLOW-2  |          |
      |<===============================|
      |           |           |RESERVE-2 |
      |  RESERVE-2            |<---------+QNI
   QNR|<--------------------+          |
      |           |           |          |
```

Figure 9: Bi-directional reservation for sender+sender scenario

The scenario with a sender-initiated and a receiver-initiated reservation is shown in Figure 10. In this case, QNI B sends out two RESERVE messages, one for the sender-initiated and one for the receiver-initiated reservation. Note that the sequence of the two RESERVE messages may be interleaved.

```
      A          QNE        QNE        B
      |           |  FLOW-1  |          |
      |===============================>|
      |  QUERY-1 |           |          |
   QNI+--------->| QUERY-1  |          |
      |           +-------------------->|QNR
      |           |           |          |
      |           |RESERVE-1 |          |
      |RESERVE-1 +<--------------------|QNR
   QNI+<---------|           |          |
      |           |           |          |
      |           |  FLOW-2  |          |
      |<===============================|
      |           |           |RESERVE-2 |
      |RESERVE-2 |           |<---------+QNI
   QNR|<--------------------+          |
      |           |           |          |
```

Figure 10: Bi-directional reservation for sender+receiver scenario

4.5. Aggregate Reservations

In order to reduce signaling and per-flow state in the network, the reservations for a number of flows may be aggregated.

```
   QNI        QNE      QNE/QNI'     QNE'    QNR'/QNE      QNR
                     aggregator          deaggregator
    |          |          |          |          |          |
    | RESERVE  |          |          |          |          |
   +--------->|          |          |          |          |
    |          | RESERVE  |          |          |          |
    |          +--------->|          |          |          |
    |          |          | RESERVE  |          |          |
    |          |          +-------------------->|          |
    |          |          | RESERVE' |          |          |
    |          |          +=========>| RESERVE' |          |
    |          |          |          +=========>| RESERVE  |
    |          |          |          |          +--------->|
    |          |          |          | RESPONSE'|          |
    |          |          | RESPONSE'|<=========+          |
    |          |          |<=========+          |          |
    |          |          |          |          | RESPONSE |
    |          |          |          | RESPONSE |<---------+
    |          |          |<--------------------+          |
    |          | RESPONSE |          |          |          |
    |          |<---------+          |          |          |
    | RESPONSE |          |          |          |          |
    |<---------+          |          |          |          |
    |          |          |          |          |          |
    |          |          |          |          |          |
```

Figure 11: Sender Initiated Reservation with Aggregation

An end-to-end per-flow reservation is initiated with the messages shown in Figure 11 as "RESERVE".

At the aggregator a reservation for the aggregated flow is initiated (shown in Figure 11 as "RESERVE'"). This may use the same QoS model as the end-to-end reservation but has an MRI identifying the aggregated flow (e.g., tunnel) instead of for the individual flows.

This document does not specify how the QSPEC of the aggregate session can be derived from the QSPECs of the end-to-end sessions.

The messages used for the signaling of the individual reservation need to be marked such that the intermediate routers will not inspect them. In the QoS NSLP the following marking possibility is applied, see also RFC3175.

All routers use essentially the same algorithm for which messages they process, i.e. all messages at aggregation level 0. However, messages have their aggregation level incremented on entry to an aggregation region and decremented on exit. In this technique the

interior routers are not required to do any rewriting of the RAO values. However, the aggregating/deaggregating routers must be configured with which of their interfaces lie at which aggregation level, and also requires consistent message rewriting at these boundaries.

In particular, the Aggregator performs the marking by modifying the QoS NSLP default NSLP-ID value to a NSLP-ID predefined value, see Section 6.6. A RAO value is then uniquely derivable from each predefined NSLP-ID. However, the RAO does not have to have a one-to-one relation to a specific NSLP-ID.

```
           Aggregator                      Deaggregator

              +---+     +---+     +---+     +---+
              |QNI|-----|QNE|-----|QNE|-----|QNR|           aggregate
              +---+     +---+     +---+     +---+           reservation

 +---+     +---+     .....     .....     +---+     +---+
 |QNI|-----|QNE|-----.   .-----.   .-----|QNE|-----|QNR|  end-to-end
 +---+     +---+     .....     .....     +---+     +---+  reservation
```

Figure 12: Reservation aggregation

The deaggregator acts as the QNR for the aggregate reservation. Session binding information carried in the RESERVE message enables the deaggregator to associate the end-to-end and aggregate reservations with one another (using the BOUND_SESSION_ID).

The key difference between this example and the one shown in Section 4.5 is that the flow identifier for the aggregate is expected to be different to that for the end-to-end reservation. The aggregate reservation can be updated independently of the per-flow end-to-end reservations.

4.6. Message Binding

Section 4.5 sketches the interaction of an aggregated end-to-end flow and an aggregate. For this scenario, and probably others, it is useful to have a method for synchronizing signaling message exchanges of different sessions. This can be used to speed up signaling, because some message exchanges can be started simultaneously and can be processed in parallel until further processing of a message from one particular session depends on another message from a different session. For instance, in Figure 11 there is a case where inclusion of a new reservation requires to increase the capacity of the encompassing aggregate first. So the RESERVE (bound message) for the

individual flow arriving at the deaggregator should wait until the RESERVE' (binding message) for the aggregate arrived successfully (otherwise the individual flow could not be included into the existing aggregate and cannot be admitted). Another alternative would be to increase the aggregate first and then to reserve resources for a set of aggregated individual flows. In this case the binding and synchronization between the (RESERVE and RESERVE') messages is not needed.

A message binding may be used (depending an the aggregators policy) as follows: a QNE (aggregator QNI' in Figure 14) generates randomly a 128-bit MSG_ID (same rules apply as for generating a SESSION_ID) and includes it as BOUND_MSG_ID object into the bound signaling message (RESERVE (1) in Figure 14) that should wait for the arrival of a related binding signaling message (RESERVE' (3) in Figure 14) that carries the associated MSG_ID object. The BOUND_SESSION_ID should also be set accordingly. Only one MSG_ID or BOUND_MSG_ID object per message is allowed. If the dependency relation between the two messages is bidirectional then the Message_Binding_Type flag is SET (value is 1). Otherwise, the Message_Binding_Type flag is UNSET. In most cases an RII object must be included in order to get a corresponding RESPONSE back.

The receiving QNE enqueues (probably after some pre-processing) this message for the corresponding session. It also starts a MsgIDWait timer in order to discard the message in case the related "triggering" message (RESERVE' in Figure 15) does not arrive. The timeout period for this time SHOULD be set to the default retransmission timeout period (QOSNSLP_REQUEST_RETRY). In case a retransmitted RESERVE message arrives before the timeout it will simply override the waiting message (i.e. the latter is discarded and the new message is now waiting with the MsgIDWait timer being reset). At the same time, the "triggering" message including a MSG_ID object, carrying the same value as the BOUND_MSG_ID object is sent by the same initiating QNE (QNI' in Figure 13). The intermediate QNE' sees the MSG_ID object, but can determine that it is not the endpoint for the session (QNR') and therefore simply forwards the message after normal processing. The receiving QNE (QNR') as endpoint for the aggregate session (i.e., deaggregator) interprets the MSG_ID object and looks for a corresponding waiting message with a BOUND_MSG_ID of the same value whose waiting condition is satisfied now. Depending on successful processing of the RESERVE' (3), processing of the waiting RESERVE will be resumed and the MsgIDWait timer will be stopped as soon as the related RESERVE' arrived.

```
      QNI        QNE      QNE/QNI'     QNE'    QNR'/QNE      QNR
                        aggregator           deaggregator
       |          |          |          |          |          |
       | RESERVE  |          |          |          |          |
       +--------->|          |          |          |          |
       |          | RESERVE  |          |          |          |
       |          +--------->|          |          |          |
       |          |          | RESERVE  |          |          |
       |          |          |   (1)    |          |          |
       |          |          +-------------------->|          |
       |          |          | RESERVE' |          |          |
       |          |          |   (2)    |          |          |
       |          |          +=========>| RESERVE' |          |
       |          |          |          |   (3)    |          |
       |          |          |          +=========>| RESERVE  |
       |          |          |          |          |   (4)    |
       |          |          |          |          +--------->|
       |          |          |          | RESPONSE'|          |
       |          |          | RESPONSE'|<=========+          |
       |          |          |<=========+          |          |
       |          |          |          |          | RESPONSE |
       |          |          |          | RESPONSE |<---------+
       |          |          |<--------------------+          |
       |          | RESPONSE |          |          |          |
       |          |<---------+          |          |          |
       | RESPONSE |          |          |          |          |
       |<---------+          |          |          |          |
       |          |          |          |          |          |
       |          |          |          |          |          |


   (1):     RESERVE:  SESSION_ID=F, BOUND_MSG_ID=x, BOUND_SESSION_ID=A
   (2)+(3): RESERVE': SESSION_ID=A, MSG_ID=x
   (4):     RESERVE:  SESSION_ID=F  (MSG_ID object was removed)
```

Figure 13: Example for using message binding

Several further cases have to be considered in this context:

o "Triggering message" (3) arrives before waiting (bound) message (1): In this case the processing of the triggering message depends on the value of the Message_Binding_Type flag. If Message_Binding_Type is UNSET (value is 0) then the triggering message can be processed normally, but the MSG_ID and the result (success or failure) should be saved for the waiting message. Thus the RESPONSE' can be sent by the QNR' immediately. If the waiting message (1) finally arrives at the QNR', it can be detected that the waiting condition was already satisfied, because

the triggering message already arrived earlier. If Message_Binding_Type is SET (value is 1) then the triggering message interprets the MSG_ID object and looks for the corresponding waiting message with a BOUND_MSG_ID of the same value, which in this case has not yet arrived. It then starts a MsgIDWait timer in order to discard the message in case the related message (RESERVE (1) in Figure 14) does not arrive. Depending on successful processing of the RESERVE (1), processing of the waiting RESERVE' will be resumed, the MsgIDWait timer will be stopped as soon as the related RESERVE arrived and the RESPONSE' can be sent by the QNR' towards the QNI'.

- o The "triggering message" (3) does not arrive at all: this may be the case due to message loss (which will cause a retransmission by the QNI' if the RII object is included) or due to a reservation failure at an intermediate node (QNE' in the example). The MsgIDWait timeout will then simply discard the waiting message at QNR'. In this case the QNR' MAY send a RESPONSE message towards the QNI informing that the synchronisation of the two messages has failed.
- o Retransmissions should use the same MSG_ID, because usually only one message of the two related messages is retransmitted. As mentioned above: retransmissions will only occur if the RII object is set in the RESERVE. If a retransmitted message with a MSG_ID arrives while a bound message with the same MSG_ID is still waiting, the retransmitted message will replace the bound message.

For a receiving node there are conceptually two lists indexed by message IDs. One list contains the IDs and results of triggering messages (those carrying a MSG_ID object), the other list contains the IDs and message contents of the bound waiting messages (those who carried a BOUND_MSG_ID). The former list is used when a triggering message arrives before the bound message. The latter list is used when a bound message arrives before a triggering message.

4.7. Reduced State or Stateless Interior Nodes

This example uses a different QoS model within a domain, in conjunction with GIST and NSLP functionality which allows the interior nodes to avoid storing GIST and QoS NSLP state. As a result the interior nodes only store the QSPEC-related reservation state, or even no state at all. This allows the QoS model to use a form of "reduced-state" operation, where reservation states with a coarser granularity (e.g., per-class) are used, or a "stateless" operation where no QoS NSLP state is needed (or created).

The key difference between this example and the use of different QoS models in Section 4.5 is that the transport characteristics for the reservation, i.e., GIST can be used in a different way for the edge-

to-edge and hop-by-hop sessions. The reduced state reservation can be updated independently of the per-flow end-to-end reservations.

4.7.1. Sender-initiated Reservation

The QNI initiates a RESERVE message (see Fig. 14). At the QNEs on the edges of the stateless or reduced-state region the processing is different and the nodes support two QoS models. At the ingress the original RESERVE message is forwarded but ignored by the stateless or reduced-state nodes. This is accomplished by marking this message, i.e., modifying the QoS NSLP default NSLP-ID value to another NSLP-ID predefined value (see Section 4.6). The marking must be accomplished by the ingress by modifying the QoS_NSLP default NSLP-ID value to a NSLP-ID predefined value. The egress must reassign the QoS NSLP default NSLP-ID value to the original end-to-end RESERVE message. An example of such operation is given in [I-D.ietf-nsis-rmd].

The egress node is the next QoS NSLP hop for the end-to-end RESERVE message. Reliable GIST transfer mode can be used between the ingress and egress without requiring GIST state in the interior. At the egress node the RESERVE message is then forwarded normally.

At the ingress a second RESERVE' message is also built (Fig. 14). This makes use of a QoS model suitable for a reduced state or stateless form of operation (such as the RMD per hop reservation). Since the original RESERVE and the RESERVE' messages are addressed identically, the RESERVE' message also arrives at the same egress QNE that was also traversed by the RESERVE message. Message binding is used to synchronize the messages.

When processed by interior (stateless) nodes the QoS NSLP processing exercises its options to not keep state wherever possible, so that no per flow QoS NSLP state is stored. Some state, e.g., per class, for the QSPEC related data may be held at these interior nodes. The QoS NSLP also requests that GIST use different transport characteristics (e.g., sending of messages in unreliable GIST transfer mode). It also requests the local GIST processing not to retain messaging association state or reverse message routing state.

Nodes, such as those in the interior of the stateless or reduced-state domain, that do not retain reservation state cannot send back RESPONSE messages (and so cannot use the refresh reduction extension).

At the egress node the RESERVE' message is interpreted in conjunction with the reservation state from the end-to-end RESERVE message (using information carried in the message to correlate the signaling flows). The RESERVE message is only forwarded further if the processing of

the RESERVE' message was successful at all nodes in the local domain, otherwise the end-to-end reservation is regarded as having failed to be installed. Note that the egress should use a timer, with a preconfigured value, that can be used to synchronise the arrival of both messages, i.e., the end-to-end RESERVE message and the local RESERVE' message.

```
           QNE             QNE             QNE            QNE
         ingress         interior        interior        egress
      GIST stateful  GIST stateless  GIST stateless  GIST stateful
             |               |               |              |
     RESERVE |               |               |              |
    -------->| RESERVE       |               |              |
             +--------------------------------------------->|
             | RESERVE'      |               |              |
             +-------------->|               |              |
             |               | RESERVE'      |              |
             |               +-------------->|              |
             |               |               | RESERVE'     |
             |               |               +------------->|
             |               |               |  RESPONSE'   |
             |<---------------------------------------------+
             |               |               |              | RESERVE
             |               |               |              +--------->
             |               |               |              | RESPONSE
             |               |               |              |<---------
             |               |               |     RESPONSE |
             |<---------------------------------------------+
     RESPONSE|               |               |              |
    <--------|               |               |              |
```

Figure 14: Sender-initiated reservation with Reduced State Interior Nodes

4.7.2. Receiver-initiated Reservation

Since NSLP neighbor relationships are not maintained in the reduced-state region, only sender-initiated signaling can be supported within the reduced state region. If a receiver-initiated reservation over a stateless or reduced state domain is required this can be implemented as shown in Figure 15.

```
           QNE             QNE             QNE
         ingress         interior         egress
     GIST stateful  GIST stateless  GIST stateful
            |               |               |
    QUERY   |               |               |
   -------->| QUERY         |               |
            +------------------------------>|
            |               |               | QUERY
            |               |               +-------->
            |               |               | RESERVE
            |               |               |<--------
            |               |       RESERVE |
            |<------------------------------+
            | RESERVE'      | RESERVE'      |
            |-------------->|-------------->|
            |               |     RESPONSE' |
            |<------------------------------+
    RESERVE |               |               |
   <--------|               |               |
```

Figure 15: Receiver-initiated reservation with Reduced State Interior Nodes

The RESERVE message that is received by the egress QNE of the stateless domain is sent transparently to the ingress QNE (known as the source of the QUERY message). When the RESERVE message reaches the ingress, the ingress QNE needs to send a sender- initiated RESERVE' over the stateless domain. The ingress QNE needs to wait for a RESPONSE'. If the RESPONSE' notifies that the reservation was accomplished successfully then the ingress QNE sends a RESERVE message further upstream.

4.8. Proxy Mode

Besides the sender- and receiver-initiated reservations, the QoS NSLP includes a functionality we refer to as Proxy Mode. Here a QNE is set by administrator assignment to work as a proxy QNE (P-QNE) for a certain region, e.g., for an administrative domain. A node initiating the signaling may set the PROXY scope flag to indicate that the signaling is meant to be confined within the area controlled by the proxy, e.g., the local access network.

The Proxy Mode has two uses. First it allows to confine the QoS NSLP signaling to a pre-defined section of the path. Secondly, it allows a node to make reservations for an incoming data flow.

For outgoing data flows and sender-initiated reservations, the end host is the QNI, and sends a RESERVE with the PROXY scope flag set.

The P-QNE is the QNR, it will receive the RESERVE, notice the PROXY scope flag is set and reply with a RESPONSE (if requested). This operation is the same as illustrated in Figure 7. The receiver-oriented reservation for outgoing flows works the same way as in Figure 8, the P-QNE is the QNI.

For incoming data flows, the end host is the QNI, and it sends a RESERVE towards the data sender with the PROXY scope flag set. Here the end host sets the MRI so that it indicates the end host as the receiver of the data, and sets the D-flag.

GIST is able to send messages towards the data sender if there is existing message routing state or it is able to use the Upstream Q-mode Encapsulation. In some cases GIST will be unable to determine the appropriate next hop for the message, and so will indicate a failure to deliver it (by sending an error message). This may occur, for example, if GIST attempts to determine an upstream next hop and there are multiple possible inbound routes that could be used.

Bi-directional reservations, as discussed in Section 4.4. The P-QNE will be the QNR or QNI for reservations.

If the PROXY scope flag is set in an incoming QoS NSLP message, the QNE must set the same flag in all QoS NSLP messages it sends that are related to this session.

5. QoS NSLP Functional Specification

5.1. QoS NSLP Message and Object Formats

A QoS NSLP message consists of a common header, followed by a body consisting of a variable number of variable-length, typed "objects". The common header and other objects are encapsulated together in a GIST NSLP-Data object. The following subsections define the formats of the common header and each of the QoS NSLP message types. In the message formats, the common header is denoted as COMMON_HEADER.

For each QoS NSLP message type, there is a set of rules for the permissible choice of object types. These rules are specified using the Augmented Backus-Naur Form (ABNF) specified in RFC 4234 [RFC4234]. The ABNF implies an order for the objects in a message. However, in many (but not all) cases, object order makes no logical difference. An implementation SHOULD create messages with the objects in the order shown here, but MUST accept the objects in any order.

5.1.1. Common Header

All GIST NSLP-Data objects for the QoS NSLP MUST contain this common header as the first 32 bits of the object (this is not the same as the GIST Common Header).

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Message Type  | Message Flags |      Generic Flags            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields in the common header are as follows:

Msg Type: 8 bits

1 = RESERVE

2 = QUERY

3 = RESPONSE

4 = NOTIFY

Message-specific flags: 8 bits

These flags are defined as part of the specfication of individual messages, and, thus, are different with each message type.

Generic flags: 16 bits

Generic flags have the same meaning for all message types. There exist currently three generic flags, the (next hop) Scoping flag (S), the Proxy scope flag (P) and the Acknowledgement Requested (A).

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Reserved      |B|A|P|S|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SCOPING (S) - when set, indicates that the message is scoped and should not travel down the entire path but only as far as the next QNE (scope="next hop"). By default, this flag is not set (default scope="whole path").

PROXY (P) - when set, indicates that the message is scoped, and should not travel down the entire path but only as far as the P-QNE. By default, this flag is not set.

ACK-REQ (A) - when set, indicates that the message should be acknowledged by the receiving peer. The flag is only used between stateful peers, and only used with RESERVE and QUERY messages. Currently, the flag is only used with refresh messages. By default the flag is not set.

BREAK (B) - when set, indicates that there are routers along the path where QoS cannot be provided.

The set of appropriate flags depends on the particular message being processed. Any bit not defined as a flag for a particular message MUST be set to zero on sending and MUST be ignored on receiving.

The ACK-REQ flag is useful when a QNE wants to make sure the messages received by the downstream QNE are truly processed by the QoS NSLP, not just delivered by GIST. This is useful for faster dead peer diagnostics on the NSLP layer. This liveliness test can only be used with refresh RESERVE messages. The ACK-REQ-flag must not be set for RESERVE messages that already include an RII object, since a confirmation has already been requested from the QNR. Reliable transmission of messages between two QoS NSLP peer should be handled by GIST, not the NSLP by itself.

5.1.2. Message Formats

5.1.2.1. RESERVE

The format of a RESERVE message is as follows:

```
RESERVE = COMMON_HEADER
          RSN [ RII ] [ REFRESH_PERIOD ] [ *BOUND_SESSION_ID ]
          [ SESSION_ID_LIST [ RSN_LIST ] ]
          [ MSG_ID / BOUND_MSG_ID ] [ INFO_SPEC ]
          [ [ PACKET_CLASSIFIER ] QSPEC ]
```

The RSN is the only mandatory object and MUST always be present in all cases. A QSPEC MUST be included in the initial RESERVE sent towards the QNR. A PACKET_CLASSIFIER MAY be provided. If the PACKET_CLASSIFIER is not provided, then the full set of information provided in the GIST MRI for the session should be used for packet classification purposes.

Subsequent RESERVE messages meant as reduced refreshes, where no QSPEC is provided, MUST NOT include a PACKET_CLASSIFIER either.

There are no requirements on transmission order, although the above order is recommended.

Two message-specific flags are defined for use in the common header with the RESERVE message. These are:

```
+-+-+-+-+-+-+-+-+
|Reserved   |T|R|
+-+-+-+-+-+-+-+-+
```

TEAR (T) - when set, indicates that reservation state and QoS NSLP operation state should be torn down. The former is indicated to the RMF. Depending on the QoS model, the tear message may include a QSPEC to further specify state removal, e.g., for an aggregation, the QSPEC may specify the amount of resources removed from the aggregate.

REPLACE (R) - when set the flag has two uses. First, it indicates that a RESERVE with different MRI (but same SID) replaces an existing one, so the old one MAY be torn down immediately. This is the default situation. This flag may be unset to indicate a desire from an upstream node to keep an existing reservation on an old branch in place. Second, this flag is also used to indicate whether the reserved resources on the old branch should be torn down or not when a data path change happens. In this case, the MRI is the same and only the route path changes.

If the REFRESH_PERIOD is not present, a default value of 30 seconds is assumed.

If the session of this message is bound to another session, then the RESERVE message SHOULD include the SESSION_ID of that other session in a BOUND_SESSION_ID object. In the situation of aggregated tunnels, the aggregated session MAY not include the SESSION_ID of its bound sessions in BOUND_SESSION_ID(s).

The negotiation of whether to perform sender or receiver-initiated signaling is done outside the QoS NSLP. Yet, in theory, it is possible that a "reservation collision" may occur if the sender believes that a sender-initiated reservation should be performed for a flow, whilst the other end believes that it should be starting a receiver- initiated reservation. If different session identifiers are used then this error condition is transparent to the QoS NSLP though it may result in an error from the RMF, otherwise the removal of the duplicate reservation is left to the QNIs/QNRs for the two sessions.

If a reservation is already installed and a RESERVE message is received with the same session identifier from the other direction (i.e., going upstream where the reservation was installed by a downstream RESERVE message, or vice versa) then an error indicating "RESERVE received from wrong direction" MUST be sent in a RESPONSE

message to the signaling message source for this second RESERVE.

A refresh right along the path can be forced by requesting a RESPONSE from the far end (i.e., by including an RII object in the RESERVE message). Without this, a refresh RESERVE would not trigger RESERVE messages to be sent further along the path, as each hop has its own refresh timer.

A QNE may ask for confirmation of tear operation by including an RII object. Retransmissions SHOULD be disabled. A QNE sending a tearing RESERVE with an RII included MAY ask GIST to use reliable transport. When the QNE sends out a tearing RESERVE, it MUST NOT send refresh messages anymore.

If the routing path changed due to mobility and the mobile node's IP address changed, and it sent a Mobile IP binding update, the resulting refresh is a new RESERVE. This RESERVE includes a new MRI and will be propagated end-to-end; there is no need to force end-to-end forwarding by including an RII.

Note: It is possible for a host to use this mechanism to constantly force the QNEs on the path to send refreshing RESERVE messages. It may, therefore, be appropriate for QNEs to perform rate limiting on the refresh messages that they send.

5.1.2.2. QUERY

The format of a QUERY message is as follows:

```
   QUERY = COMMON_HEADER
           [ RII ][ *BOUND_SESSION_ID ]
           [ PACKET_CLASSIFIER ] [ INFO_SPEC ] QSPEC
```

QUERY messages MUST always include a QSPEC. QUERY messages MAY include a PACKET_CLASSIFIER when the message is used to trigger a receiver-initiated reservation. If a PACKET_CLASSIFIER is not included then the full GIST MRI should be used for packet classification purposes in the subsequent RESERVE. A QUERY message MAY contain a second QSPEC object.

A QUERY message for requesting information about network resources MUST contain an RII object to match an incoming RESPONSE to the QUERY.

The QSPEC object describes what is being queried for and may contain objects that gather information along the data path. There are no requirements on transmission order, although the above order is recommended.

One message-specific flags are defined for use in the common header with the QUERY message. This is:

```
+-+-+-+-+-+-+-+-+
|Reserved     |R|
+-+-+-+-+-+-+-+-+
```

RESERVE-INIT (R) - when this is set, the QUERY is meant as a trigger for the recipient to make a resource reservation by sending a RESERVE.

If the session of this message is bound to another session, then the RESERVE message SHOULD include the SESSION_ID of that other session in a BOUND_SESSION_ID object. In the situation of aggregated tunnels, the aggregated session MAY not include the SESSION_ID of its bound sessions in BOUND_SESSION_ID(s).

5.1.2.3. RESPONSE

The format of a RESPONSE message is as follows:

```
RESPONSE = COMMON_HEADER
           [ RII / RSN ] INFO_SPEC [SESSION_ID_LIST [ RSN_LIST ] ]
           [ QSPEC ]
```

A RESPONSE message MUST contain an INFO_SPEC object which indicates the success of a reservation installation or an error condition. Depending on the value of the INFO_SPEC, the RESPONSE MAY also contain a QSPEC object. The value of an RII or an RSN object was provided by some previous QNE. There are no requirement on transmission order, although the above order is recommended.

No message-specific flags are defined for use in the common header with the RESPONSE message.

5.1.2.4. NOTIFY

The format of a NOTIFY message is as follows:

```
NOTIFY = COMMON_HEADER
         INFO_SPEC [ QSPEC ]
```

A NOTIFY message MUST contain an INFO_SPEC object indicating the reason for the notification. Depending on the INFO_SPEC value, it MAY contain a QSPEC object providing additional information.

No message-specific flags are defined for use with the NOTIFY message.

5.1.3. Object Formats

The QoS NSLP uses a Type-Length-Value (TLV) object format similar to that used by GIST. Every object consists of one or more 32-bit words with a one-word header. For convenience the standard object header is shown here:

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

The value for the Type field comes from the shared NSLP object type space, the various objects are presented in subsequent sections. The Length field is given in units of 32 bit words and measures the length of the Value component of the TLV object (i.e., it does not include the standard header).

The bits marked 'A' and 'B' are flags used to signal the desired treatment for objects whose treatment has not been defined in the protocol specification (i.e., whose Type field is unknown at the receiver). The following four categories of object have been identified, and are described here.

AB=00 ("Mandatory"): If the object is not understood, the entire message containing it MUST be rejected, and an error message sent back.

AB=01 ("Ignore"): If the object is not understood, it MUST be deleted and the rest of the message processed as usual.

AB=10 ("Forward"): If the object is not understood, it MUST be retained unchanged in any message forwarded as a result of message processing, but not stored locally.

AB=11 ("Refresh"): If the object is not understood, it should be incorporated into the locally stored QoS NSLP signaling application operational state for this flow/session, forwarded in any resulting message, and also used in any refresh or repair message which is generated locally. The contents of this object does not need to be interpreted, and should only be stored as bytes on the QNE.

The remaining bits marked 'r' are reserved. The extensibility flags AB are similar to those used in the GIST specification. All objects defined in this specification MUST be understood by all QNEs, thus, they MUST have the AB-bits set to "00". A QoS NSLP implementation must recognize objects of the following types: RII, RSN,

REFRESH_PERIOD, BOUND_SESSION_ID, INFO_SPEC, and QSPEC.

The object header is followed by the Value field, which varies for different objects. The format of the Value field for currently defined objects is specified below.

The object diagrams here use '//' to indicate a variable sized field.

5.1.3.1. Request Identification Information (RII)

Type: 0x01

Length: Fixed - 1 32-bit word

Value: An identifier which MUST be (probabilistically) unique within the context of a SESSION_ID, and SHOULD be different every time a RESPONSE is desired. Used by a QNE to match back a RESPONSE to a request in a RESERVE or QUERY message.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Response Identification Information (RII)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.1.3.2. Reservation Sequence Number (RSN)

Type: 0x02

Length: Fixed - 2 32-bit words

Value: An incrementing sequence number that indicates the order in which state modifying actions are performed by a QNE, and an epoch identifier to allow the identification of peer restarts. The RSN has local significance only, i.e., between a QNE and its downstream stateful peers. The RSN is not reset when the downstream peer changes.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Reservation Sequence Number (RSN)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Epoch Identifier                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.1.3.3. Refresh Period (REFRESH_PERIOD)

Type: 0x03

Length: Fixed - 1 32-bit word

Value: The refresh timeout period R used to generate this message; in milliseconds.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Refresh Period (R)                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.1.3.4. Bound Session ID (BOUND_SESSION_ID)

Type: 0x04

Length: Fixed - 5 32-bit words

Value: contains an 8-bit Binding_Code that indicates the nature of binding. The rest specifies the SESSION_ID (as specified in GIST [I-D.ietf-nsis-ntlp]) of the session that MUST be bound to the session associated with the message carrying this object.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  RESERVED                     |  Binding Code |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                          Session ID                           +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Currently defined Binding Codes are:

o 0x01 - Tunnel and end-to-end sessions

o 0x02 - Bi-directional sessions

o 0x03 - Aggregate sessions

o 0x04 - Dependent sessions (binding session is alive only if the other session is also alive)

o 0x05 - Indicated session caused pre-emption

More binding codes maybe defined based on the above four atomic binding actions. Note a message may include more than one BOUND_SESSION_ID object. This may be needed in case one needs to define more specifically the reason for binding, or if the session must on depend on more than one other session (with possibly different reasons). Note that a session with e.g., SID_A (the binding session) can express its unidirectional dependency relation to another session with e.g., SID_B (the bound session) by including a BOUND_SESSION_ID object containing SID_B in its messages.

5.1.3.5. Packet Classifier (PACKET_CLASSIFIER)

Type: 0x05

Length: Variable

Value: Contains a variable length MRM-specific data

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//          Method-specific classifier data (variable)         //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

At this stage, the QoS NSLP only uses the path-coupled routing MRM. The method-specific classifier data is two bytes long and consists of a set of flags:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|X|Y|P|T|F|S|A|B|                      Reserved                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The flags are:

X - Source Address and Prefix

Y - Destination Address and Prefix

P - Protocol

T - DiffServ Code Point

F - Flow Label

S - SPI

A - Source Port

B - Destination Port

The flags indicate which fields from the MRI MUST be used by the packet classifier. This allows a subset of the information in the MRI to be used for identifying the set of packets which are part of the reservation. Flags MUST only be set if the data is present in the MRI (i.e., where there is a corresponding flag in the GIST MRI, the flag can only be set if the corresponding GIST MRI flag is set). It should be noted that some flags in the PACKET_CLASSIFIER (X and Y) relate to data that is always present in the MRI, but are optional to use for QoS NSLP packet classification. The appropriate set of flags set may depend, to some extent, on the QoS model being used.

As mentioned earlier in this section, the QoS NSLP is currently only defined for use with the Path-Coupled Message Routing Mechanism (MRM) in GIST. Future work may extend the QoS NSLP to additional routing mechanisms. Such MRMs must include sufficient information in the MRI to allow the subset of packets for which QoS is to be provided to be identified. When QoS NSLP is extended to support a new MRM, appropriate method-specific classifier data for the PACKET_CLASSIFIER object MUST be defined.

5.1.3.6. Information Object (INFO_SPEC) and Error Codes

Type: 0x06

Length: Variable

Value: Contains a 16-bit error code, a 4-bit error class, a 4-bit error source identifier type, and an 8-bit error source identifier length (in 32-bit words), an error source identifier and optionally variable length error-specific information.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Error Code           |E-Class|ESI Typ|   ESI-Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                   Error Source Identifier                   //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//             Optional error-specific information             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class Field:

The four E-Class bits of the object indicate the error severity class. The currently defined severity classes are:

o 0x01 - Informational

o 0x02 - Success

o 0x03 - Protocol Error

o 0x04 - Transient Failure

o 0x05 - Permanent Failure

o 0x06 - QoS Model Error

Error field:

Within each error severity class a number of error values are defined.

o Informational:

* 0x01 - Unknown BOUND_SESSION_ID: the message refers to an unknown SESSION_ID in its BOUND_SESSION_ID object.

* 0x02 - Route Change: possible route change occurred on downstream path.

* 0x03 - Reduced refreshes not supported, full QSPEC required.

* 0x04 - Congestion situation: Possible congestion situation occurred on downstream path.

* 0x05 - Unknown SESSION ID in SESSION_ID_LIST

* 0x06 - Mismatching RSN in RSN LIST

o Success:

* 0x01 - Reservation successful

* 0x02 - Tear down successful

* 0x03 - Acknowledgement

* 0x04 - Refresh successful

o Protocol Error:

* 0x01 - Illegal message type: the type given in the Message Type field of the common header is unknown.

* 0x02 - Wrong message length: the length given for the message does not match the length of the message data.

* 0x03 - Bad flags value: an undefined flag or combination of flags was set in the generic flags

* 0x04 - Bad flags value: an undefined flag or combination of flags was set in the message-specific flags

* 0x05 - Mandatory object missing: an object required in a message of this type was missing.

* 0x06 - Illegal object present: an object was present which must not be used in a message of this type.

* 0x07 - Unknown object present: an object of an unknown type was present in the message.

* 0x08 - Wrong object length: the length given for the object did not match the length of the object data present.

* 0x09 - RESERVE received from wrong direction.

* 0x0a - Unknown object field value: a field in an object had an unknown value.

* 0x0b - Duplicate object present.

* 0x0c - Malformed QSPEC.

* 0x0d - Unknown MRI.

* 0x0e - Erroneous value in the TLV object's value field.

* 0x0f - Incompatible QSPEC

o Transient Failure:

* 0x01 - No GIST reverse-path forwarding state

* 0x02 - No path state for RESERVE, when doing a receiver- oriented reservation

* 0x03 - RII conflict

* 0x04 - Full QSPEC required

* 0x05 - Mismatch synchronization between end-to-end RESERVE and intra-domain RESERVE

* 0x06 - Reservation preempted

* 0x07 - Reservation failure

* 0x08 - Path truncated - Next peer dead

o Permanent Failure:

* 0x01 - Internal or system error

* 0x02 - Authorization failure

o QoS Model Error:

This error class can be used by QoS Models to add error codes specific to the QoS Model being used. All these errors and events are created outside the QoS NSLP itself. The error codes in this class are defined in QoS model specifications. Note that this error class may also include codes that are not purely errors, but rather some non-fatal information.

Error Source Identifier

The Error Source Identifier is for diagnostic purposes and its inclusion is OPTIONAL. It is suggested that implementations use this for the IP address, host name or other identifier of the QNE generating the INFO_SPEC to aid diagnostic activities. A QNE SHOULD NOT be used in any other purpose other than error logging or presenting to the user as part of any diagnostic information. A QNE SHOULD NOT attempt to send a message to that address.

If no Error Source Identifier is included, the Error Source Identifier Type field must be zero.

Currently three Error Source Identifiers have been defined: IPv4, IPv6 and FQDN.

Error Source Identifier: IPv4

Error Source Identifier Type: 0x01

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       32-bit IPv4 address                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Error Source Identifier: IPv6

Error Source Identifier Type: 0x02

```
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

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       128-bit IPv6 address                    +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Error Source Identifier: FQDN name in UTF-8

Error Source Identifier Type: 0x03

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                            FQDN Name                         //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If the length of the FQDN name is not a multiple of 32-bits, the field is padded with zero octets to the next 32-bit boundary.

If a QNE encounters protocol errors, it MAY include additional information, mainly for diagnostic purposes. Additional information MAY be included if the type of an object is erroneous, or a field has an erroneous value.

If the type of an object is erroneous, the following optional error-specific information may be included at the end of the INFO_SPEC.

Object Type Info:

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |         Object Type           |            Reserved           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This object provides information about the type of object which caused the error.

If a field in an object had an incorrect value, the following optional error-specific information may be added at the end of the INFO_SPEC.

Object Value Info:

```
    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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | Rsvd  |  Real Object Length    |            Offset             |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    //                            Object                            //
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Real Object Length: Since the length in the original TLV header may be inaccurate, this field provides the actual length of the object (including the TLV Header) included in the error message.

Offset: The byte in the object at which the QNE found the error. When this byte is set to "0", the complete object is included.

Object: The invalid TLV object (including the TLV Header).

This object carries information about a TLV object which was found to be invalid in the original message. An error message may contain more than one Object Value Info object.

5.1.3.7. SESSION ID List (SESSION_ID_LIST)

Type: 0x07

Length: Variable

Value: A list of 128-bit SESSION IDs used in summary refresh and summary tear messages. All SESSION IDs are concatenated together.

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                                                               +
    |                                                               |
    +                          Session ID 1                         +
    |                                                               |
    +                                                               +
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    :                                                               :
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                                                               +
    |                                                               |
    +                          Session ID n                         +
    |                                                               |
    +                                                               +
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.1.3.8. Reservation Sequence Number (RSN) List (RSN_LIST)

Type: 0x08

Length: Variable

Value: A list of 32-bit Reservation Sequence Number (RSN) values. All RSN are concatenated together.

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                       Epoch Identifier                        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |             Reservation Sequence Number 1 (RSN1)              |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    :                                                               :
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |             Reservation Sequence Number n (RSNn)              |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.1.3.9. Message ID (MSG_ID)

Type: 0x09

Length: Fixed - 5 32-bit words

Value: contains an 1-bit Message_Binding_Type (D) that indicates the dependency relation of a message binding. The rest specifies a 128 bit randomly generated value that "uniquely" identifies this particular message.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   RESERVED                                  |D|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                           Message ID                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Message Binding Codes are:

* 0 - Unidirectional binding dependency

* 1 - Bi-directional binding dependency

5.1.3.10. Bound Message ID (BOUND_MSG_ID)

Type: 0x0A

Length: Fixed - 5 32-bit words

Value: contains an 1-bit Message_Binding_Type (D) that indicates the dependency relation of a message binding. The rest specifies a 128 bit randomly generated value that refers to a Message ID in another message.

```
       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
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                  RESERVED                                   |D|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                                                               |
      +                                                               +
      |                                                               |
      +                        Bound Message ID                       +
      |                                                               |
      +                                                               +
      |                                                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Message Binding Codes are:

* 0 - Unidirectional binding dependency

* 1 - Bi-directional binding dependency

5.1.3.11. QoS Specification (QSPEC)

Type: 0x0B

Length: Variable

Value: Variable length QSPEC (QoS specification) information, which is QoS Model dependent.

The contents and encoding rules for this object are specified in other documents. See [I-D.ietf-nsis-qspec].

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   //                         QSPEC Data                          //
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.2. General Processing Rules

This section provides the general processing rules used by QoS-NSLP. The triggers communicated between RM/QOSM and QoS-NSLP functionalities are given in Appenices A.1, A.2 and A.3.

5.2.1. State Manipulation

The processing of a message and its component objects involves manipulating the QoS NSLP and reservation state of a QNE.

For each flow, a QNE stores (RMF-related) reservation state which depends on the QoS model / QSPEC used and QoS NSLP operation state which includes non-persistent state (e.g., the API parameters while a QNE is processing a message) and persistent state which is kept as long as the session is active.

The persistent QoS NSLP state is conceptually organized in a table with the following structure. The primary key (index) for the table is the SESSION_ID:

SESSION_ID

A 128-bit identifier.

The state information for a given key includes:

Flow ID

Based on GIST MRI. Several entries are possible in case of mobility events.

SII-Handle for each upstream and downstream peer

The SII-Handle is a local identifier generated by GIST and passed over the API. It is a handle that allows to refer to a particular GIST next hop. See SII-Handle in [I-D.ietf-nsis-ntlp] for more information.

RSN from the upstream peer

The RSN is a 32 bit counter.

The latest local RSN

A 32 bit counter.

List of RII for outstanding responses with processing information.

The RII is a 32 bit number.

State lifetime

The state lifetime indicates how long the state that is being

signaled for remains valid.

List of bound sessions

A list of BOUND_SESSION_ID 128-bit identifiers for each session bound to this state.

Scope of the signaling

If the Proxy scope is used, a flag is needed to identify all signaling of this session as being scoped.

Adding the state requirements of all these items gives an upper bound on the state to be kept by a QNE. The need to keep state depends on the desired functionality at the NSLP layer.

5.2.2. Message Forwarding

QoS NSLP messages are sent peer-to-peer along the path. The QoS NSLP does not have the concept of a message being sent directly to the end of the path. Instead, messages are received by a QNE, which may then send another message (which may be identical to the received message, or contain some subset of objects from it) to continue in the same direction (i.e., towards QNI or QNR) as the message received.

The decision on whether to generate a message to forward may be affected by the value of the SCOPING or PROXY flags, or by the presence of an RII object.

5.2.3. Standard Message Processing Rules

If a mandatory object is missing from a message then the receiving QNE MUST NOT propagate the message any further. It MUST construct a RESPONSE message indicating the error condition and send it back to the peer QNE that sent the message.

If a message contains an object of an unrecognised type, then the behavior depends on the AB extensibility flags.

If the Proxy scope flag was set in an incoming QoS NSLP message, the QNE must set the same flag in all QoS NSLP messages it sends that are related to this session.

5.2.4. Retransmissions

Retransmissions may happen end-to-end, e.g., between QNI and QNR (using an RII object), or peer-to-peer, between two adjacent QNEs. When a QNE transmits a RESERVE with an RII object set, it waits for a

RESPONSE from the responding QNE. QoS NSLP messages for which a response is requested by including an RII object, but fail to elicit a response are retransmitted. Similarly, a QNE may include the ACK-REQ-flag to request confirmation of a refresh message reception from its immediate peer. The retransmitted message should be exactly the same as the original message, e.g., the RSN is not modified with each retransmission.

The initial retransmission occurs after a QOSNSLP_REQUEST_RETRY wait period. Retransmissions MUST be made with exponentially increasing wait intervals (doubling the wait each time). QoS NSLP messages SHOULD be retransmitted until either a RESPONSE (which might be an error) has been obtained, or until QOSNSLP_RETRY_MAX seconds after the initial transmission. In the latter case, a failure SHOULD be indicated to the signaling application. The default values for the above-mentioned timers are:

QOSNSLP_REQUEST_RETRY: 2 seconds Wait interval before initial retransmit of the message

QOSNSLP_RETRY_MAX: 30 seconds Give up retrying to send the message

Retransmissions SHOULD be disabled for tear messages.

5.2.5. Rerouting

5.2.5.1. Last Node Behavior

As discussed in Section 3.2.12 some care needs to be taken to handle cases where the last node on the path may change.

A node that is the last node on the path, but not the data receiver (or an explicitly configured proxy for it), MUST continue to attempt to send messages downstream to probe for path changes. This must be done in order to handle the "Path Extension" case described in Section 3.2.12.1.

A node on the path, that was not previously the last node, MUST take over as the last node on the signaling path if GIST path change detection identifies that there are no further downstream nodes on the path. This must be done in order to handle the "Path Truncation" case described in Section 3.2.12.1.

5.2.5.2. Avoiding Mistaken Teardown

In order to handle the spurious route change problem described in Section 3.2.12.2, the RSN must be used in a particular way when maintaining the reservation after a route change is believed to have

occurred.

We assume that the current RSN (RSN[current]) is initially RSN0.

When a route change is believed to have occurred, the QNE SHOULD send a RESERVE message, including the full QSPEC. This must contain an RSN which is RSN[current] = RSN0 + 2. It SHOULD include an RII, to request a response from the QNR. An SII-Handle MUST NOT be specified when passing this message over the API to GIST, so that it is correctly routed to the new peer QNE.

When the QNE receives the RESPONSE message that relates to the RESERVE message sent down the new path, it SHOULD send a RESERVE message with the TEAR flag sent down the old path. To do so, it MUST request GIST to use its explicit routing mechanism and the QoS NSLP MUST supply an SII-Handle relating to the old peer QNE. When sending this RESERVE message it MUST contain an RSN which is RSN[current] - 1. (RSN[current] remains unchanged).

If the RESPONSE received after sending the RESERVE down the new path contains the code "Refresh successful" in the INFO_SPEC, then the QNE MAY elect not to send the tearing RESERVE, since this indicates that the path is unchanged.

5.2.5.3. Upstream Route Change Notification

GIST may notify the QoS NSLP that a possible upstream route change has occurred over the GIST API. On receiving such a notification, the QoS NSLP SHOULD send a NOTIFY message with Informational code 0x02 for signaling sessions associated with the identified MRI. If this is sent, it MUST be sent to the old peer using the GIST explicit routing mechanism through the use of the SII-Handle.

On receiving such a NOTIFY message, the QoS NSLP SHOULD use the InvalidateRoutingState API call to inform GIST that routing state may be out of date. The QoS NSLP SHOULD send a NOTIFY message upstream. The NOTIFY message should be propagated back to the QNI or QNR.

5.2.5.4. Route Change Oscillation

In some circumstances a route change may occur, but the path then falls back to the original route.

After a route change the routers on the old path will continue to refresh the reservation until soft state times out, or an explicit TEAR is received.

After detecting an upstream route change a QNE SHOULD consider the

new upstream peer as current and not fall back to the old upstream peer unless:

- it stops receiving refreshes from the old upstream peer for at least the soft state timeout period and then starts receiving messages from the old upstream peer again

- or, it stops receiving refreshes from the new upstream peer for at least the soft state timeout period.

GIST routing state keeps track of the latest upstream peer it has seen, and so may spuriously indicate route changes occur when the old upstream peer refreshes its routing state until the state at that node is explicitly torn down or times out.

5.3. Object Processing

This section presents processing rules for individual QoS NSLP objects.

5.3.1. Reservation Sequence Number (RSN)

A QNE's own RSN is a sequence number which applies to a particular signaling session (i.e., with a particular SESSION_ID). It MUST be incremented for each new RESERVE message where the reservation for the session changes. The RSN is manipulated using the serial number arithmetic rules from [RFC1982], which also defines wrapping rules and the meaning of 'equals', 'less than' and 'greater than' for comparing sequence numbers in a circular sequence space.

The RSN starts at zero. It is stored as part of the per-session state and it carries on incrementing (i.e., it is not reset to zero) when a downstream peer change occurs. (Note that section 5.2.5.2 provides some particular rules for use when a downstream peer changes.)

The RSN object also contains an Epoch Identifier, which provides a method for determining when a peer has restarted (e.g., due to node reboot or software restart). The exact method for providing this value is implementation defined. Options include storing a serial number which is incremented on each restart, picking a random value on each restart or using the restart time.

On receiving a RESERVE message a QNE examines the Epoch Identifier to determine if the peer sending the message has restarted. If the Epoch Identifier is different to that stored for the reservation then the RESERVE message MUST be treated as an updated reservation (even if the RSN is less than the current stored value), and the stored RSN

and Epoch Identifier MUST be updated to the new values.

When receiving a RESERVE message a QNE uses the RSN given in the message to determine whether the state being requested is different to that already stored. If the RSN is equal to that stored for the current reservation the current state MUST be refreshed. If the RSN is greater than the current stored value, the current reservation MUST be modified appropriately as specified in the QSPEC (provided that admission control and policy control succeed), and the stored RSN value updated to that for the new reservation. If the RSN is greater than the current stored value and the RESERVE was a reduced refresh, the QNE SHOULD send upstream a transient error message "Full QSPEC required". If the RSN is less than the current value, then it indicates an out-of-order message and the RESERVE message MUST be discarded.

If the QNE does not store per-session state (and so does not keep any previous RSN values) then it MAY ignore the value of the RSN. It MUST also copy the same RSN into the RESERVE message (if any) it sends as a consequence of receiving this one.

5.3.2. Request Identification Information (RII)

A QNE sending QUERY or RESERVE messages may require a response to be sent. It does so by including a Request Identification Information (RII) object. When creating an RII object the QNE MUST select the value for the RII such that it is probabilistically unique within the given session. A RII object is typically set by the QNI.

A number of choices are available when implementing this. Possibilities might include using a random value, or a node identifier together with a counter. If the value collides with one selected by another QNE for a different QUERY then RESPONSE messages may be incorrectly terminated, and may not be passed back to the node that requested them.

The node that created the RII object MUST remember the value used in the RII to match back any RESPONSE it will receive. The node SHOULD use a timer to identify situations where it has taken too long to receive the expected RESPONSE. If the timer expires without receiving a RESPONSE it MAY perform a retransmission as discussed in Section 5.2.4. In this case this QNE MUST not generate any RESPONSE or NOTIFY message to notify this error.

If an intermediate QNE wants to receive a response for an outgoing message, but the message already included an RII when it arrived, the QNE MUST NOT add a new RII object nor replace the old RII object, but MUST simply remember this RII to match a later RESPONSE message.

When it receives the RESPONSE, it forwards the RESPONSE upstream towards the RII originating node. Note that only the node that originally created the RII can set up a retransmission timer. Thus, if an intermediate QNE decides to use the RII already contained in the message, it MUST NOT set up a retransmission timer, but rely on the retransmission timer set up by the QNE that inserted the RII.

When receiving a message containing an RII object the node MUST send a RESPONSE if

o The SCOPING flag is set ('next hop' scope),

o The PROXY scope flag is set and the QNE is the P-QNE, or

o This QNE is the last one on the path for the given session.

and the QNE keeps per-session state for the given session.

In the rare event that the QNE wants to request a response for a message that already included an RII, and this RII value conflicts with an existing RII value on the QNE, the node should interrupt the processing the message, and send an error message upstream to indicate an RII collision, and request a retry with a new RII value.

5.3.3. BOUND_SESSION_ID

As shown in the examples in Section 4, the QoS NSLP can relate multiple sessions together. It does this by including the SESSION_ID from one session in a BOUND_SESSION_ID object in messages in another session.

When receiving a message with a BOUND_SESSION_ID object, a QNE MUST copy the BOUND_SESSION_ID object into all messages it sends for the same session. A QNE that stores per-session state MUST store the value of the BOUND_SESSION_ID.

The BOUND_SESSION_ID is only indicative in nature. However, a QNE implementation may use BOUND_SESSION_ID information to optimize resource allocation, e.g., for bidirectional reservations. When receiving a tear down message (e.g., a RESERVE message with tear down semantic) for an aggregate reservation, it may use this information to initiate a tear down for end-to-end sessions bound to the aggregate. A QoS NSLP implementation MUST be ready to process more than one BOUND_SESSION_ID object within a single message.

5.3.4. REFRESH_PERIOD

Refresh timer management values are carried by the REFRESH_PERIOD object which has local significance only. At the expiration of a "refresh timeout" period, each QNE independently examines its state and sends a refreshing RESERVE message to the next QNE peer where it is absorbed. This peer-to-peer refreshing (as opposed to the QNI initiating a refresh which travels all the way to the QNR) allows QNEs to choose refresh intervals as appropriate for their environment. For example, it is conceivable that refreshing intervals in the backbone, where reservations are relatively stable, are much larger than in an access network. The "refresh timeout" is calculated within the QNE and is not part of the protocol; however, it must be chosen to be compatible with the reservation lifetime as expressed by the REFRESH_PERIOD, and an assessment of the reliability of message delivery.

The details of timer management and timer changes (slew handling and so on) are identical to the ones specified in Section 3.7 of RFC 2205 [RFC2205].

There are two time parameters relevant to each QoS NSLP state in a node: the refresh period R between generation of successive refreshes for the state by the neighbor node, and the local state's lifetime L. Each RESERVE message may contain a REFRESH_PERIOD object specifying the R value that was used to generate this (refresh) message. This R value is then used to determine the value for L when the state is received and stored. The values for R and L may vary from peer to peer.

5.3.5. INFO_SPEC

The INFO_SPEC object is carried by the RESPONSE and NOTIFY messages and it is used to report a successful, an unsuccessful, or an error situation. In case of an error situation the error messages SHOULD be generated even if no RII object is included in the RESERVE or in the QUERY messages. Note that when the TEAR flag is set in the RESERVE message an error situation SHOULD NOT trigger the generation of a RESPONSE message.

Six classes of INFO_SPEC objects are identified and specified in Section 5.1.3.6. The message processing rules for each class are defined below.

A RESPONSE message MUST carry INFO_SPEC objects towards the QNI. The RESPONSE message MUST be forwarded unconditionally up to the QNI. The actions that SHOULD be undertaken by the QNI that receives the INFO_SPEC object are specified by the local policy of the QoS model

supported by this QNE. The default action is that the QNI that receives the INFO_SPEC object SHOULD not trigger any other QoS NSLP procedure.

The Informational INFO_SPEC class MUST be generated by a by a stateful QoS NSLP QNE when an Informational error class is caught. The Informational INFO-SPEC object MUST be carried by a RESPONSE or a NOTIFY message.

In case of an unidirectional reservation, the Success INFO_SPEC class MUST be generated by a stateful QoS NSLP QNR when a RESERVE message is received and the reservation state installation or refresh succeeded. In case of a bi-directional reservation the INFO-SPEC object SHOULD be generated by a stateful QoS NSLP QNE when a RESERVE message is received and the reservation state installation or refresh succeeded. The Success INFO-SPEC object MUST be carried by a RESPONSE or a NOTIFY message.

In case of an unidirectional reservation, the Protocol Error INFO_SPEC class MUST be generated by a stateful QoS NSLP QNE when a RESERVE or QUERY message is received by the QNE and a protocol error is caught. In case of a bi-directional reservation, the Protocol Error INFO_SPEC class SHOULD be generated by a stateful QoS NSLP QNE when a RESERVE or QUERY message is received by the QNE and a protocol error is caught. A RESPONSE message MUST carry this object, which MUST be forwarded unconditionally towards the upstream QNE that generated the RESERVE or QUERY message that triggered the generation of this INFO_SPEC object. The default action for a stateless QoS NSLP QNE that detects such an error is that none of the QoS NSLP objects SHOULD be processed and the RESERVE or QUERY message SHOULD be forwarded downstream.

In case of an unidirectional reservation, the Transient Failure INFO_SPEC class MUST be generated by a stateful QoS NSLP QNE when a RESERVE or QUERY message is received by the QNE and one Transient failure error code is caught, or when an event happens that causes a transient error. In case of a bi-directional reservation, the Transient Failure INFO_SPEC class SHOULD be generated by a stateful QoS NSLP QNE when a RESERVE or QUERY message is received by the QNE and one Transient failure error code is caught.

A RESPONSE message MUST carry this object, which MUST be forwarded unconditionally towards the upstream QNE that generated the RESERVE or QUERY message that triggered the generation of this INFO_SPEC object. The transient RMF-related error MAY also be carried by a NOTIFY message. The default action is that the QNE that receives this INFO_SPEC object SHOULD re-trigger the retransmission of the RESERVE or QUERY message that triggered the generation of the

INFO_SPEC object. The default action for a stateless QoS NSLP QNE that detects such an error is that none of the QoS NSLP objects SHOULD be processed and the RESERVE or QUERY message SHOULD be forwarded downstream.

In case of an unidirectional reservation, the Permanent Failure INFO_SPEC class MUST be generated by a stateful QoS NSLP QNE when a RESERVE or QUERY message is received by a QNE and an internal or system error occured, or authorization failed. In case of a bi-directional reservation, the Permanent Failure INFO_SPEC class SHOULD be generated by a stateful QoS NSLP QNE when a RESERVE or QUERY message is received by a QNE and an internal or system error occured, or authorization failed. A RESPONSE message MUST carry this object, which MUST be forwarded unconditionally towards the upstream QNE that generated the RESERVE or QUERY message that triggered this protocol error. The permanent RMF-related, the internal or system errors MAY also be carried by a NOTIFY message. The default action for a stateless QoS NSLP QNE that detects such an error is that none of the QoS NSLP objects SHOULD be processed and the RESERVE or QUERY message SHOULD be forwarded downstream.

The QoS-specific error class may be used when errors outside the QoS NSLP itself occur that are related to the particular QoS Model being used. The processing rules of these errors are not specified in this document.

5.3.6. SESSION_ID_LIST

A SESSION_ID_LIST is carried in RESERVE messages. It is used in two cases, to refresh or two tear down the indicated sessions. A SESSION_ID_LIST carries information about sessions that should be refreshed or torn down, in addition to the main (primary) session indicated in the RESERVE.

If the primary SESSION_ID is not understood, the SESSION_ID_LIST object MUST NOT be processed.

When a stateful QNE goes through the SESSION_ID_LIST, if it finds one or more unknown SESSION ID values, it SHOULD construct an informational RESPONSE message back to the upstream stateful QNE with error code for unknown SESSION ID in SESSION_ID_LIST, and include all uknown SESSION IDs in a SESSION_ID_LIST.

If the RESERVE is a tear, for each session in the SESSION_ID_LIST, the stateful QNE MUST inform the RMF that the reservation is no longer required. RSN values MUST also be interpreted in order to distinguish whether the tear down is valid, or whether it is refering to an old state, and, thus, should be silently discarded.

If the RESERVE is a refresh, the stateful QNE MUST also process the RSN_LIST object as detailed in the next section.

If the RESERVE is a tear, for each session in the SESSION_ID_LIST, the QNE MUST inform the RMF that the reservation is no longer required. RSN values MUST be interpreted.

Note that a stateless QNE can not support summary or single reduced refreshes, and always needs full single refreshes.

5.3.7. RSN_LIST

An RSN_LIST MUST be carried in RESERVE messages when a QNE wants to perform a refresh or tear-down of several sessions with a single NSLP message. The RSN_LIST object MUST be populated with RSN values of the same sessions and in the same order as indicated in the SESSION_ID_LIST. Thus, entries on both objects at locationg i refer to the same session.

If the primary session and RSN reference in the RESERVE were not understood, the stateful QNE MUST NOT process the RSN_LIST. Instead an error RESPONSE SHOULD be sent back to the upstream stateful QNE.

On receiving an RSN_LIST object, the stateful QNE should check whether the number of items in the SESSION_ID_LIST and RSN_LIST objects match. If there is a mismatch, the stateful QNE SHOULD send back a protocol error indicating a bad value in the object.

While matching the RSN_LIST values to the SESSION_ID_LIST values, if one or more RSN values in the RSN_LIST are not in synch with the local values, the stateful QNE SHOULD construct an informational RESPONSE message with an error code for RSN mismatch in RSN_LIST. The stateful QNE MUST include the erroneous SESSION ID and RSN values in SESSION_ID_LIST and RSN_LIST objects in the RESPONSE.

If no errors were found in processing the RSN_LIST, the stateful QNE refreshes the reservation states of all sessions, both the primary single session indicated in the refresh, and all sessions in the SESSION_ID_LIST.

For each successfully processed session in the RESERVE, the stateful QNE performs a refresh of the reservation state. Thus, even if some sessions were not in synch, the remaining sessions in the SESSION_ID_LIST and RSN_LIST are refreshed.

5.3.8. QSPEC

The contents of the QSPEC depends on the QoS model being used. A template for QSPEC objects can be found in [I-D.ietf-nsis-qspec].

Upon reception, the complete QSPEC is passed to the Resource Management Function (RMF), along with other information from the message necessary for the RMF processing. A QNE may also receive an INFO_SPEC that includes a partial or full QSPEC. This will also be passed to the RMF.

5.4. Message Processing Rules

This section provides rules for message processing. Not all possible error situations are considered. A general rule for dealing with erroneous messages is that a node should evaluate the situation before deciding how to react. There are two ways to react to erroneous messages:

a) Silently drop the message, or

b) Drop the message, and reply with an error code to the sender.

The default behavior, in order to protect the QNE from a possible DoS attack, is to silently drop the message. However, if the QNE is able to authenticate the sender, e.g., through GIST, the QNE may send a proper error message back to the neighbor QNE in order to let it know that there is an inconsistency in the states of adjacent QNEs.

5.4.1. RESERVE Messages

The RESERVE message is used to manipulate QoS reservation state in QNEs. A RESERVE message may create, refresh, modify or remove such state. A QNE sending a RESERVE MAY require a response to be sent by including a Request Identification Information (RII) object, see Section 5.3.2.

RESERVE messages MUST only be sent towards the QNR. A QNE that receives a RESERVE message checks the message format. In case of malformed messages, the QNE MAY send a RESPONSE message with the appropriate INFO_SPEC.

Before performing any state changing actions a QNE MUST determine whether the request is authorized. The way to do this check depends on the authorization model being used.

When the RESERVE is authorized, a QNE checks the COMMON_HEADER flags. If the TEAR flag is set, the message is a tearing RESERVE which

indicates complete QoS NSLP state removal (as opposed to a reservation of zero resources). On receiving such a RESERVE message the QNE MUST inform the RMF that the reservation is no longer required. The RSN value MUST be processed. After this, there are two modes of operation:

1. If the tearing RESERVE did not include an RII, i.e., the QNI did not want a confirmation, the QNE SHOULD remove the QoS NSLP state. It MAY signal to GIST (over the API) that reverse path state for this reservation is no longer required. Any errors in processing the tearing RESERVE SHOULD NOT be sent back towards the QNI since the upstream QNEs will already have removed their session states, thus, they are unable to do anything to the error.

2. If an RII was included, the stateful QNE SHOULD still keep the NSLP operational state until a RESPONSE for the tear going towards the QNI is received. This operational state SHOULD be kept for one refresh interval, after which the NSLP operational state for the session is removed. Depending on the QoS model, the tear message MAY include a QSPEC to further specify state removal. If the QoS model requires a QSPEC, and none is provided, the QNE SHOULD reply with an error message, and SHOULD NOT remove the reservation.

If the tearing RESERVE includes a QSPEC, but none is required by the QoS model, the QNE MAY silently discard the QSPEC and proceed as if it did not exist in the message. In general, a QoS NSLP implementation should carefully consider, when an error message should be sent, and when not. If the tearing RESERVE did not include an RII, then the upstream QNE has removed the RMF and NSLP states, and will not be able to do anything to the error. If an RII was included, the upstream QNE may still have the NSLP operational state, but no RMF state.

If a QNE receives a tearing RESERVE for a session it still has the operational state, but the RMF state was removed, the QNE SHOULD accept the message and forward it downstream as if all is well.

If the tearing RESERVE includes a SESSION_ID_LIST, the stateful QNE MUST process the object as described earlier in this document, and for each identified session, indicate to the RMF that the reservation is no longer required.

If a QNE receives a refreshing RESERVE for a session it still has the operational state, but the RMF state was removed, the QNE MUST silently drop the message and not forward it downstream.

As discussed in Section 5.2.5.2, to avoid incorrect removal of state after a rerouting event, a node receiving a RESERVE message with the

TEAR flag set which does not come from the current peer QNE, identified by its SII, MUST be ignored and MUST NOT be forwarded.

If the QNE has reservations which are bound and dependent to this session (they contain the SESSION_ID of this session in their BOUND_SESSION_ID object and use Binding Code: 0x04), it MUST send a NOTIFY message for each of the reservations with an appropriate INFO_SPEC. If the QNE has reservations which are bound, but which they are not dependent to this session (the Binding Code in the BOUND_SESSION_ID object has one of the values: 0x01, 0x02, 0x03), it MAY send a NOTIFY message for each of the reservations with an appropriate INFO_SPEC. The QNE MAY elect to send RESERVE messages with the TEAR flag set for these reservations.

The default behavior of a QNE that receives a RESERVE with a SESSION_ID for which it already has state installed but with a different flow ID is to replace the existing reservation (and tear down the reservation on the old branch if the RESERVE is received with a different SII).

In some cases, this may not be the desired behavior. In that case, the QNI or a QNE MAY set the REPLACE flag in the common header to zero to indicate that the new session does not replace the existing one.

A QNE that receives a RESERVE with the REPLACE flag set to zero but with the same SII, will indicate REPLACE=0 to the RMF (where it will be used for the resource handling). Furthermore, if the QNE maintains a QoS NSLP state then it will also add the new flow ID in the QoS NSLP state. If the SII is different, this means that the QNE is a merge point. In that case, in addition to the operations specified above, the value REPLACE=0 is also indicating that a tearing RESERVE SHOULD NOT be sent on the old branch.

When a QNE receives a RESERVE message with an unknown SESSION_ID and this message contains no QSPEC because it was meant as a refresh then the node MUST send a RESPONSE message with an INFO_SPEC that indicates a missing QSPEC to the upstream peer ("Full QSPEC required"). The upstream peer SHOULD send a complete RESERVE (i.e., one containing a QSPEC) on the new path (new SII).

At a QNE, resource handling is performed by the RMF. For sessions with the REPLACE flag set to zero, we assume that the QoS model includes directions to deal with resource sharing. This may include, adding the reservations, or taking the maximum of the two or more complex mathematical operations.

This resource handling mechanism in the QoS Model is also applicable

to sessions with different SESSION_ID but related through the BOUND_SESSION_ID object. Session replacement is not an issue here, but the QoS Model may specify whether to let the sessions that are bound together share resources on common links or not.

Finally, it is possible that a RESERVE is received with no QSPEC at all. This is the case of a reduced refresh. In this case, rather than sending a refreshing RESERVE with the full QSPEC, only the SESSION_ID and the RSN are sent to refresh the reservation. Note that this mechanism just reduces the message size (and probably eases processing). One RESERVE per session is still needed. Such a reduced refresh may further include a SESSION_ID_LIST and RSN_LIST, which indicate further sessions to be refreshed along the primary session. The processing of these objects were described earlier in this document.

If the REPLACE flag is set, the QNE SHOULD update the reservation state according to the QSPEC contained in the message (if the QSPEC is missing the QNE SHOULD indicate this error by replying with a RESPONSE containing the corresponding INFO_SPEC "Full QSPEC required"). It MUST update the lifetime of the reservation. If the REPLACE flag is not set, a QNE SHOULD NOT remove the old reservation state if the SII which is passed by GIST over the API is different than the SII that was stored for this reservation. The QNE MAY elect to keep sending refreshing RESERVE messages.

If a stateful QoS NSLP QNE receives a RESERVE message with the BREAK flag set then the BREAK flag of new generated messages (e.g., RESERVE or RESPONSE) MUST be set. When a stateful QoS NSLP QNE receives a RESERVE message with the BREAK flag not set then the IP-TTL and Original-TTL values in GIST RecvMessage primitive MUST be monitored. If they differ then the BREAK flag of new generated messages (e.g., RESERVE or RESPONSE) SHOULD be set. In situations where a QNE or a domain is able to provide QoS using other means, see Section 3.3.5, then the BREAK flag MUST not be set.

If the RESERVE message included an RII, and any of the following are true, the QNE MUST send a RESPONSE message:

o If the QNE is configured, for a particular session, to be a QNR,

o The SCOPING flag is set,

o The Proxy scope flag is set and the QNE is a P-QNE, or

o The QNE is the last QNE on the path to the destination.

When a QNE receives a RESERVE message, its processing may involve

sending out another RESERVE message.

If a QNE has received a RESPONSE mandating the use of full refreshes from its downstream peer for a session, the QNE MUST continue to use full refresh messages.

If the session of this message is bound to another session, then the RESERVE message SHOULD include the SESSION_ID of that other session in a BOUND_SESSION_ID object. In the situation of aggregated tunnels, the aggregated session MAY not include the SESSION_ID of its bound sessions in BOUND_SESSION_ID(s).

In case of receiver-initiated reservations, the RESERVE message must follow the same path that has been followed by the QUERY message. Therefore, GIST is informed, over the QoS NSLP/GIST API, to pass the message upstream, i.e., by setting GIST "D" flag, see GIST [I-D.ietf-nsis-ntlp].

The QNE MUST create a new RESERVE and send it to its next peer, when:

- A new resource set up was done,

- A new resource set up was not done, but the QOSM still defines that a RESERVE must be propagated,

- The RESERVE is a refresh and includes new MRI, or

- If the RESERVE-INIT flag is included in an arrived QUERY.

If the QNE sent out a refresh RESERVE with the ACK-REQ-flag set, and did not receive a RESPONSE from its immediate stateful peer within the retransmission period of QOSNSLP_RETRY_MAX, the QNE SHOULD send a NOTIFY to its immediate upstream sateful peer and indicate "Path truncated - Next peer dead" in the INFO_SPEC. The ACK-REQ-flag SHOULD NOT be added to a RESERVE that already include an RII object, since a confirmation from the QNR has already been requested.

Finally, if a received RESERVE requested acknowledgement through the ACK-REQ-flag in the COMMON HEADER flags and the processing of the message was successul, the stateful QNE SHOULD send back a RESPONSE with an INFO_SPEC carrying the acknowledgement success code. The QNE MAY include the ACK-REQ-flag in the next refresh message it will send for the session. The use of the ACK-REQ-flag for diagnostics purposes is a policy issue, i.e., using an acknowledged refresh message as a hint to further probe the end-to-end path can be used simply as a hint to check that the end-to-end path is still intact.

5.4.2. QUERY Messages

A QUERY message is used to request information about the data path without making a reservation. This functionality can be used to 'probe' the network for path characteristics or for support of certain QoS models, or for initiating a receiver-initiated reservation.

A QNE sending a QUERY indicates a request for a response by including a Request Identification Information (RII) object, see Section 5.3.2. A request to initiate a receiver-initiated reservation is done through the RESERVE-INIT flag, see Section 5.1.2.2.

When a QNE receives a QUERY message the QSPEC is passed to the RMF for processing. The RMF may return a modified QSPEC that is used in any QUERY or RESPONSE message sent out as a result of the QUERY processing.

When processing a QUERY message, a QNE checks whether the RESERVE-INIT flag is set. If the flag is set, the QUERY is used to install reverse path state. In this case, if the QNE is not the QNI, it creates a new QUERY message to send downstream. If the QUERY contained a QSPEC, it MUST be passed to the RMF where it may be modified by the QoS Model specific QUERY processing. If the QNE is the QNI, the QNE creates a RESERVE message, which contains a QSPEC received from the RMF and which may be based on the received QSPEC. If this node was not expecting to perform a receiver-initiated reservation then an error MUST be sent back along the path.

If an RII object is present, and if the QNE is the QNR, the SCOPING flag is set or the PROXY scope flag is set and the QNE is a P-QNE, the QNE MUST generate a RESPONSE message and pass it back along the reverse of the path used by the QUERY.

In other cases, the QNE MUST generate a QUERY message which is then forwarded further along the path using the same MRI, Session ID and Direction as provided when the QUERY was received over the GIST API.

The QSPEC to be used is that provided by the RMF as described previously. When generating a QUERY to send out to pass the query further along the path, the QNE MUST copy the RII object (if present) unchanged into the new QUERY message. A QNE that is also interested in the response to the query keeps track of the RII to identify the RESPONSE when it passes through it.

Note that QUERY messages with the RESERVE-INIT flag set MUST be answered by the QNI. This feature may be used, e.g., following handovers, to set up new path state in GIST, and request the other

party to send a RESERVE back on this new GIST path.

If a stateful QoS NSLP QNE receives a QUERY message with the RESERVE-INIT flag and BREAK flag set then the BREAK flag of new generated messages (e.g., QUERY, RESERVE or RESPONSE) MUST be set. When a stateful QoS NSLP QNE receives a QUERY message with the the RESERVE-INIT flag set and BREAK flag not set then then the IP-TTL and Original-TTL values in GIST RecvMessage primitive MUST be monitored. If they differ then the BREAK flag of new generated messages (e.g., QUERY, RESERVE or RESPONSE) SHOULD be set. In situations where a QNE or a domain is able to provide QoS using other means, see Section 3.3.5, then the BREAK flag MUST not be set.

Finally, if a received QUERY requested acknowledgement through the ACK-REQ-flag in the COMMON HEADER flags and the processing of the message was successul, the stateful QNE SHOULD send back a RESPONSE with an INFO_SPEC carrying the acknowledgement success code.

5.4.3. RESPONSE Messages

The RESPONSE message is used to provide information about the result of a previous QoS NSLP message, e.g., confirmation of a reservation or information resulting from a QUERY. The RESPONSE message does not cause any state to be installed, but may cause state(s) to be modified, e.g., if the RESPONSE contains information about an error.

A RESPONSE message MUST be sent when the QNR processes a RESERVE or QUERY message containing an RII object or if the QNE receives a scoped RESERVE or a scoped QUERY. In this case, the RESPONSE message MUST contain the RII object copied from the RESERVE or the QUERY. Also, if there is an error in processing a received RESERVE, a RESPONSE is sent indicating the nature of the error. In this case, the RII and RSN, if available, MUST be included in the RESPONSE.

On receipt of a RESPONSE message containing an RII object, the stateful QoS NSLP QNE MUST attempt to match it to the outstanding response requests for that signaling session. If the match succeeds, then the RESPONSE MUST NOT be forwarded further along the path if it contains an INFO_SPEC class informational or success. If the QNE did not insert this RII itself, if must forward the RESPONSE to the next peer. Thus, for RESPONSES indicating success, forwarding should only stop if the QNE inserted the RII by itself, If the RESPONSE carries an INFO_SPEC indicating an error, forwarding SHOULD continue upstream towards the QNI by using RSNs as described in the next paragraph.

On receipt of a RESPONSE message containing an RSN object, a stateful QoS NSLP QNE MUST compare the RSN to that of the appropriate signaling session. If the match succeeds then the INFO_SPEC MUST be

processed. If the INFO_SPEC object is used to notify errors then the node MUST use the stored upstream peer RSN value, associated with the same session, and forward the RESPONSE message further along the path towards the QNI.

If the INFO_SPEC is not used to notify error situations, see above, then if the RESPONSE message carries an RSN, the message MUST NOT be forwarded further along the path.

If there is no match for RSN, the message SHOULD be silently dropped.

On receipt of a RESPONSE message containing neither an RII nor an RSN object, the RESPONSE MUST NOT be forwarded further along the path.

In the typical case RESPONSE messages do not change the states installed in intermediate QNEs. However, depending on the QoS model, there may be situations where states are affected, e.g.,

- if the RESPONSE includes an INFO_SPEC describing an error situation resulting in reservations to be removed, or

- the QoS model allows a QSPEC to define [min,max] limits on the resources requested, and downstream QNEs gave less resources than their upstream nodes, which means that the upstream nodes may release a part of the resource reservation.

If a stateful QoS NSLP QNE receives a RESPONSE message with the BREAK flag set then the BREAK flag of new generated message (e.g., RESPONSE) MUST be set.

5.4.4. NOTIFY Messages

NOTIFY messages are used to convey information to a QNE asynchronously. NOTIFY messages do not cause any state to be installed. The decision to remove state depends on the QoS model. The exact operation depends on the QoS model. A NOTIFY message does not directly cause other messages to be sent. NOTIFY messages are sent asynchronously, rather than in response to other messages. They may be sent in either direction (upstream or downstream).

A special case of synchronous NOTIFY is when the upstream QNE asked to use reduced refresh by setting the appropriate flag in the RESERVE. The QNE receiving such a RESERVE MUST reply with a NOTIFY and a proper INFO_SPEC code whether the QNE agrees to use reduced refresh between the upstream QNE.

The Transient error code 0x07 "Reservation preempted" is sent to the QNI whose resources were preempted. The NOTIFY message carries

information to the QNI that one QNE no longer has a reservation for the session. It is up to the QNI to decide what to do based on the QoS Model being used. The QNI would normally tear down the preempted reservation by sending a RESERVE with the TEAR flag set using the SII of the preempted reservation. However, the QNI can follow other procedures as specified in its QoS Model. More discussion on preemption can be found in the QSPEC Template [I-D.ietf-nsis-qspec] and the individual QoS Model specifications.

6. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the QoS NSLP, in accordance with BCP 26 RFC 2434 [RFC2434].

The QoS NSLP requires IANA to create a number of new registries: - QoS NSLP Message Types - QoS NSLP Binding Codes - QoS NSLP Error Classes and Error Codes

It also requires registration of new values in a number of registries:

- NSLP Object Types - GIST NSLP-ID - Router Alert Option Values (IPv4 and IPv6)

6.1. QoS NSLP Message Type

The QoS NSLP Message Type is an 8 bit value. This specification defines four QoS NSLP message types, which form the initial contents of this registry: RESERVE (0x01), QUERY (0x02), RESPONSE (0x03) and NOTIFY (0x04).

The value 0 is reserved. Values 1-239 are to be allocated by Standards Action. Values 240 to 255 are for Experimental/Private Use.

When a new message type is defined, any message flags used with it must also be defined.

6.2. NSLP Message Objects

[Delete this part if already done by another NSLP:

A new registry is to be created for NSLP Message Objects. This is a 12-bit field (giving values from 0 to 4095). This registry is shared between a number of NSLPs. Allocation policies are as follows: 0-1023: Standards Action 1024-1999: Specification Required 2000-2047:

Private/Experimental Use 2048-4095: Reserved

When a new object is defined, the extensbility bits (A/B) must also be defined.]

This document defines eleven new NSLP objects. These are described in Section 5.1.3: RII (0x01), RSN (0x02), REFRESH_PERIOD (0x03), BOUND_SESSION_ID (0x04), PACKET_CLASSIFIER (0x05), INFO_SPEC (0x06), SESSION ID LIST (0x07), RSN LIST (0x08), MSG_ID (0x09), BOUND_MSG_ID (0x0A), and QSPEC (0x0B).

Values are to be assigned from the Standards Action required section of the NSLP Object Type registry.

6.3. QoS NSLP Binding Codes

A new registry is to be created for the 8-bit Binding Codes used in the BOUND_SESSION_ID object. The initial values for this registry are listed in Section 5.1.3.4.

Value 0 is reserved. Values 1 to 127 are to be assigned based on a policy of Specification Required. Values 128 to 159 are for Exerimental/Private Use. Other values are Reserved.

6.4. QoS NSLP Error Classes and Error Codes

In addition Error Classes and Error Codes for the INFO_SPEC object are defined. These are described in Section 5.1.3.6.

The Error Class is 4-bits in length. The initial values are: 0: Reserved 1: Informational 2: Success 3: Protocol Error 4: Transient Failure 5: Permanent Failure 6: QoS Model Error 7-15: Reserved

The Error Code is 16 bits in length. Each Error Codes are assigned within a particular Error Class. This requires the creation of a registry for Error Codes in each Error Class. The error code 0 in each class is Reserved.

Policies for the error code registries are as follows: 0-8191: Standards Action 8192-12287: Specification Required 12288-16383: Experimental/Private Use 16384-65536: Reserved

The initial assignments for the Error Code registries are given in section 5.1.3.6.

6.5. QoS NSLP Error Source Identifiers

Section 5.1.3.4 defines Error Source Identifiers, the type of which is identified by a 4 bit value. The value 0 is reserved, all other values are assigned on a basis of Specification Required, except for 14 and 15 which are for Experimental/Private Use.

Initial assignments are given in section 5.1.3.4.

6.6. NSLP IDs and Router Alert Option Values

This specification defines an NSLP for use with GIST. Furthermore it specifies that a number of NSLP-ID values are used for the support of bypassing intermediary nodes. Consequently, new identifiers must be assigned for them from the GIST NSLP identifier registry. The QoS NSLP requires that 32 NSLP-ID values be assigned, corresponding to QoS NSLP Aggregation Levels 0 to 31.

The GIST specification also requires that NSLP-IDs be associated with specific Router Alert Option (RAO) values (although multiple NSLP-IDs may be associated with the same value). For the purposes of the QoS NSLP, each of its NSLP-ID values should be associated with a different RAO value. This requires that a block of 32 new IPv4 RAO values and a block of 32 new IPv6 RAO values be assigned, corresponding to QoS NSLP Aggregation Levels 0 to 31.

7. Security Considerations

The security requirement for the QoS NSLP is to protect the signaling exchange for establishing QoS reservations against identified security threats. For the signaling problem as a whole, these threats have been outlined in NSIS threats [RFC4081]; the NSIS framework [RFC4080] assigns a subset of the responsibility to GIST and the remaining threats need to be addressed by NSLPs. The main issues to be handled can be summarized as:

Authorization:

The QoS NSLP must assure that the network is protected against theft-of-service by offering mechanisms to authorize the QoS reservation requester. A user requesting a QoS reservation might want proper resource accounting and protection against spoofing and other security vulnerabilities which lead to denial of service and financial loss. In many cases authorization is based on the authenticated identity. The authorization solution must provide guarantees that replay attacks are either not possible or limited to a certain extent. Authorization can also be based on traits which

enables the user to remain anonymous. Support for user identity confidentiality can be accomplished.

Message Protection:

Signaling message content should be protected against modification, replay, injection and eavesdropping while in transit. Authorization information, such as authorization tokens, need protection. This type of protection at the NSLP layer is necessary to protect messages between NSLP nodes.

Rate Limitation:

QNEs should perform rate limiting on the refresh messages that they send. An attacker could send erroneous messages on purpose, forcing the QNE to constantly reply with an error message. Authentication mechanisms would help in figuring out if error situations should be reported to the sender, or silently ignored. If the sender is authenticated, the QNE should reply promptly.

Prevention of Denial of Service Attacks:

GIST and QoS NSLP nodes have finite resources (state storage, processing power, bandwidth). The protocol mechanisms s in this document try to minimize exhaustion attacks against these resources when performing authentication and authorization for QoS resources.

To some extent the QoS NSLP relies on the security mechanisms provided by GIST which by itself relies on existing authentication and key exchange protocols. Some signaling messages cannot be protected by GIST and hence should be used with care by the QoS NSLP. An API must ensure that the QoS NSLP implementation is aware of the underlying security mechanisms and must be able to indicate which degree of security is provided between two GIST peers. If a level of security protection for QoS NSLP messages is required which goes beyond the security offered by GIST or underlying security mechanisms, additional security mechanisms described in this document must be used. The different usage environments and the different scenarios where NSIS is used make it very difficult to make general statements without reducing its flexibility.

7.1. Trust Relationship Model

This specification is based on a model which requires trust between neighboring NSLP nodes to establish a chain-of-trust along the QoS signaling path. The model is simple to deploy, was used in previous QoS authorization environments (such as RSVP) and seems to provide sufficiently strong security properties. We refer to this model as

the New Jersey Turnpike.

On the New Jersey Turnpike, motorists pick up a ticket at a toll booth when entering the highway. At the highway exit the ticket is presented and payment is made at the toll booth for the distance driven. For QoS signaling in the Internet this procedure is roughly similar. In most cases the data sender is charged for transmitted data traffic where charging is provided only between neighboring entities.

```
   +------------------+  +------------------+  +------------------+
   |          Network |  |          Network |  |          Network |
   |             X    |  |             Y    |  |             Z    |
   |                  |  |                  |  |                  |
   |              ----------->        ----------->                |
   |                  |  |                  |  |                  |
   |                  |  |                  |  |                  |
   +--------^---------+  +------------------+  +-------+----------+
            |                                          .
            |                                          .
            |                                          v
         +--+---+  Data                   Data      +--+---+
         | Node |  ==============================>  | Node |
         |  A   |  Sender                Receiver   |  B   |
         +------+                                   +------+

     Legend:

     ----> Peering relationship which allows neighboring
           networks/entities to charge each other for the
           QoS reservation and data traffic

     ====> Data flow

     .... Communication to the end host
```

Figure 42: New Jersey Turnpike Model

The model shown in Figure 42 uses peer-to-peer relationships between different administrative domains as a basis for accounting and charging. As mentioned above, based on the peering relationship a chain-of-trust is established. There are several issues which come to mind when considering this type of model:

o The model allows authorization on a request basis or on a per-session basis. Authorization mechanisms are elaborated in Section 4.9. The duration for which the QoS authorization is valid needs to

be controlled. Combining the interval with the soft-state interval is possible. Notifications from the networks also seem to be viable approach.

o The price for a QoS reservation needs to be determined somehow and communicated to the charged entity and to the network where the charged entity is attached. Protocols providing Advice of Charge functionality are out of scope.

o This architecture is simple enough to allow a scalable solution (ignoring reverse charging, multicast issues and price distribution).

Charging the data sender as performed in the model simplifies security handling by demanding only peer-to-peer security protection. Node A would perform authentication and key establishment. The established security association (together with the session key) would allow the user to protect QoS signaling messages. The identity used during the authentication and key establishment phase would be used by Network X (see Figure 42) to perform the so-called policy-based admission control procedure. In our context this user identifier would be used to establish the necessary infrastructure to provide authorization and charging. Signaling messages later exchanged between the different networks are then also subject to authentication and authorization. The authenticated entity thereby is, however, the neighboring network and not the end host.

The New Jersey Turnpike model is attractive because of its simplicity. S. Schenker et. al. [shenker] discuss various accounting implications and introduced the edge pricing model. The edge pricing model shows similarity to the model described in this section with the exception that mobility and the security implications itself are not addressed.

7.2. Authorization Model Examples

Various authorization models can be used in conjunction with the QoS NSLP.

7.2.1. Authorization for the Two Party Approach

The two party approach (Figure 43) is conceptually the simplest authorization model.

```
+-------------+  QoS request     +--------------+
|  Entity     |----------------->| Entity       |
|  requesting |                  | authorizing  |
|  resource   |granted / rejected| resource     |
|             |<-----------------| request      |
+-------------+                  +--------------+
          ^                            ^
          +............................+
                  compensation
```

Figure 43: Two party approach

In this example the authorization decision only involves the two entities, or makes use of previous authorization using an out-of-band mechanism to avoid the need for active participation of an external entity during the NSIS protocol execution.

This type of model may be applicable, e.g., between two neighboring networks (inter-domain signaling) where a long-term contract (or other out-of-band mechanisms) exists to manage charging and provides sufficient information to authorize individual requests.

7.2.2. Token-based Three Party Approach

An alternative approach makes use of tokens, such as those described in RFC 3520 [RFC3520] and RFC 3521 [RFC3521] or used as part of the Open Settlement Protocol [osp]. Authorization tokens are used to associate two different signaling protocols runs (e.g., SIP and NSIS) and their authorization decision with each other. The latter is a form of assertion or trait. As an example, with the authorization token mechanism, some form of authorization is provided by the SIP proxy, which acts as the resource authorizing entity in Figure 44. If the request is authorized, then the SIP signaling returns an authorization token which can be included in the QoS signaling protocol messages to refer to the previous authorization decision. The tokens themselves may take a number of different forms, some of which may require the entity performing the QoS reservation to query external state.

```
     Authorization
     Token Request   +--------------+
     +-------------->| Entity  C    | financial settlement
     |               | authorizing  | <..................+
     |               | resource     |                    .
     |        +------+ request      |                    .
     |        |      +--------------+                    .
     |        |                                          .
     |        |Authorization                             .
     |        |Token                                     .
     |        |                                          .
     |        |                                          .
     |        |                                          .
     |        |      QoS request                         .
   +-------------+ + Authz. Token   +--------------+     .
   |  Entity     |----------------->| Entity B     |     .
   |  requesting |                  | performing   |     .
   |  resource   |granted / rejected| QoS          |  <..+
   |      A      |<-----------------| reservation  |
   +-------------+                  +--------------+
```

Figure 44: Token based three party approach

For the digital money type of systems (e.g., OSP tokens), the token represents a limited amount of credit. So, new tokens must be sent with later refresh messages once the credit is exhausted.

7.2.3. Generic Three Party Approach

Another method is for the node performing the QoS reservation to delegate the authorization decision to a third party, as illustrated in Figure 45. The authorization decision may be performed on a per-request basis, periodically, or on a per-session basis.

```
                                   +--------------+
                                   | Entity C     |
                                   | authorizing  |
                                   | resource     |
                                   | request      |
                                   +-----------+--+
                                      ^        |
                                  QoS |        | QoS
                                 authz|        |authz
                                  req.|        | res.
                      QoS             |        v
   +-------------+    request      +--+-----------+
   |  Entity     |---------------->| Entity B     |
   |  requesting |                 | performing   |
   |  resource   |granted / rejected| QoS         |
   |      A      |<----------------| reservation  |
   +-------------+                 +--------------+
```

Figure 45: Three party approach

7.3. Computing the Authorization Decision

Whenever an authorization decision has to be made then there is the question which information serves as an input to the authorizing entity. The following information items have been mentioned in the past for computing the authorization decision (in addition to the authenticated identity):

Price

QoS objects

Policy rules

Policy rules include attributes like time of day, subscription to certain services, membership, etc. into consideration when computing an authorization decision.

The policies used to make the authorization are outside the scope of this document and implementation/deployment specific.

8. Acknowledgments

The authors would like to thank Eleanor Hepworth, Ruediger Geib, Roland Bless, Nemeth Krisztian, Markus Ott, Mayi Zoumaro-Djayoon, Martijn Swanink, and Ruud Klaver for their useful comments. Roland, especially, has done deep reviews of the document, making sure the

protocol is well defined. Bob Braden provided helpful comments and guidance which were gratefully received.

9. Contributors

This draft combines work from three individual drafts. The following authors from these drafts also contributed to this document: Robert Hancock (Siemens/Roke Manor Research), Hannes Tschofenig and Cornelia Kappler (Siemens AG), Lars Westberg and Attila Bader (Ericsson) and Maarten Buechli (Dante) and Eric Waegeman (Alcatel). In addition, Roland Bless has contributed considerable amounts of text all along the writing of this specification.

Sven Van den Bosch was the first editor of the draft. Since version 06 of the draft, Jukka Manner has taken the editorship. Yacine El Mghazli (Alcatel) contributed text on AAA. Charles Shen and Henning Schulzrinne suggested the use of the reason field in the BOUND_SESSION_ID.

10. References

10.1. Normative References

[I-D.ietf-nsis-ntlp]
Schulzrinne, H. and R. Hancock, "GIST: General Internet Signalling Transport", draft-ietf-nsis-ntlp-14 (work in progress), July 2007.

[I-D.ietf-nsis-qspec]
Ash, J., "QoS NSLP QSPEC Template", draft-ietf-nsis-qspec-17 (work in progress), July 2007.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[I-D.ietf-nsis-applicability-mobility-signaling]
Lee, S., "Applicability Statement of NSIS Protocols in Mobile Environments", draft-ietf-nsis-applicability-mobility-signaling-07 (work in progress), July 2007.

[I-D.ietf-nsis-rmd]
Bader, A., "RMD-QOSM - The Resource Management in Diffserv QOS Model", draft-ietf-nsis-rmd-10 (work in progress), June 2007.

[RFC1633] Braden, B., Clark, D., and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.

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

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

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2961] Berger, L., Gan, D., Swallow, G., Pan, P., Tommasi, F., and S. Molendini, "RSVP Refresh Overhead Reduction Extensions", RFC 2961, April 2001.

[RFC3175] Baker, F., Iturralde, C., Le Faucheur, F., and B. Davie, "Aggregation of RSVP for IPv4 and IPv6 Reservations", RFC 3175, September 2001.

[RFC3520] Hamer, L-N., Gage, B., Kosinski, B., and H. Shieh, "Session Authorization Policy Element", RFC 3520, April 2003.

[RFC3521] Hamer, L-N., Gage, B., and H. Shieh, "Framework for Session Set-up with Media Authorization", RFC 3521, April 2003.

[RFC3726] Brunner, M., "Requirements for Signaling Protocols", RFC 3726, April 2004.

[RFC4080] Hancock, R., Karagiannis, G., Loughney, J., and S. Van den Bosch, "Next Steps in Signaling (NSIS): Framework", RFC 4080, June 2005.

[RFC4081] Tschofenig, H. and D. Kroeselberg, "Security Threats for Next Steps in Signaling (NSIS)", RFC 4081, June 2005.

[RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[lrsvp] Manner, J. and K. Raatikainen, "Localized QoS Management for Multimedia Applications in Wireless Access Networks. IASTED, IMSA, August, 2004, pp. 193-200.".

[nslp-auth] Manne, J., Stiemerling, M., and H. Tschofenig, "Authorization for NSIS Signaling Layer Protocols, Work in Progress.".

[opwa95] Breslau, L., "Two Issues in Reservation Establishment, Proc. ACM SIGCOMM '95 , Cambridge , MA , August 1995.".

[osp] ETSI, ""Telecommunications and Internet protocol harmonization over networks (tiphon); open settlement protocol (osp) for inter- domain pricing, authorization, and usage exchange", Technical Specification 101 321, version 2.1.0.".

[qos-auth] Tschofenig, H., "QoS NSLP Authorization Issues. Work in Progress.".

[shenker] Shenker, S. and et al., ""Pricing in computer networks: Reshaping the research agenda", Proc. of TPRC 1995, 1995.".

Appendix A. Appendix A. Abstract NSLP-RMF API

This appendix provides an abstract API between the QoS NSLP and the RMF. It should not be taken as a strict rule of implementors, but rather help clarify the interface between the NSLP and RMF.

A.1. Triggers from QOS-NSLP towards RMF

The QoS-NSLP triggers the RMF/QOSM functionality by using the sendrmf() primitive:

```
int sendrmf(sid, nslp_req_type, qspec, authorization_info,
NSLP_objects, filter, features_in, GIST_API_triggers,
incoming_interface, outgoing_interface)
```

- o sid: SESSION_ID - The NSIS session identifier
- o nslp_req_type: indicates type of request:
  - * RESERVE
  - * QUERY

   * RESPONSE
   * NOTIFY
- qspec: the QSPEC object, if present
- authorization_info: the AUTHO_SESSION object, if present
- NSLP_objects: data structure that contains a list with received QoS-NSLP objects. This list can be used by e.g., Local application, Management, Policy control:
   * RII
   * RSN
   * BOUND_SESSION_ID list
   * REFRESH_PERIOD
   * SESSION_ID_LIST
   * RSN_LIST
   * INFO_SPEC
   * MSG_ID
   * BOUND_MSG_ID
- filter: the information for packet filtering, based on the MRI and the PACKET_CLASSIFIER object.
- features_in: it represents the flags included in the common header of the received QOS-NSLP message, but also additional
- triggers:
   * BREAK
   * REQUEST REDUCED REFRESHES
   * RESERVE-INIT
   * TEAR
   * REPLACE
   * ACK-REQ
   * PROXY
   * SCOPING
   * synchronization_required: this attribute is set (see e.g., Section 4.6 and 4.7.1) when the QoS-NSLP functionality supported by a QNE Egress receives a non tearing RESERVE message that includes a MSG_ID or a BOUND_MSG_ID object and the BINDING_CODE value of the BOUND_SESSION_ID object is equal to one of the following values:
      + Tunnel and end-to-end sessions
      + Aggregate sessions
   * GIST_API_triggers: it represents the attributes that are provided by GIST to QoS-NSLP via the GIST API:
      + NSLPID
      + Routing-State-Check
      + SII-Handle
      + Transfer-Attributes
      + GIST-Hop-Count
      + IP-TTL
      + IP-Distance

o incoming_interface: the ID of the incoming interface. Used only when the QNE reserves resources on incoming interface. Default is 0 (no reservations on incoming interface)
o outgoing_interface: the ID of the outgoing interface. Used only when the QNE reserves resources on outgoing interface. Default is 0 (no reservations on outgoing interface)

A.2. Triggers from RMF/QOSM towards QOS-NSLP

The RMF triggers the QoS-NSLP functionality using the "recvrmf()" and "config()" primitives to perform either all or a subset of the features listed below.

The recvrmf() primitive represents either a response to a request that has been sent via the API by the QoS-NSLP or an asynchronous notification. Note that when the RMF/QOSM receives a request via the API from the QoS-NSLP function, then one or more than one "recvrmf()" response primitives can be sent via the API towards QoS-NSLP. In this way the QOS-NSLP can generate one, or more than one, QoS-NSLP messages that can be for example, used in the situation that the arrival of one end-to-end RESERVE triggers the generation of two (or more) RESERVE messages, an end-to-end RESERVE message and one (or more) intra-domain (local) RESERVE message.

The config() primitive is used to configure certain features, such as QNE type, statefullness, bypassing support.

Note that the selection of the subset of triggers is controlled by the QoS Model.

```
int recvrmf(sid, nslp_resp_type, qspec, authorization_info, status,
NSLP_objects, filter, features_out, GIST_API_triggers
incoming_interface, outgoing_interface)
```

o sid: SESSION_ID - The NSIS session identifier
o nslp_resp_type: indicates type of response:
  * RESERVE
  * QUERY
  * RESPONSE
  * NOTIFY
o qspec: the QSPEC object, if present
o authorization_info: the AUTHO_SESSION object, if present
o status: boolean that notifies the status of the reservation and can be used by QOS-NSLP to include in the INFO_SPEC object:
  * RESERVATION_SUCCESSFUL
  * TEAR_DOWN_SUCCESSFUL

   * NO RESOURCES
   * RESERVATION_FAILURE
   * RESERVATION_PREEMPTED: reservation was pre-empted
   * AUTHORIZATION_FAILED: authorizing the request failed
   * MALFORMED_QSPEC: request failed due to malformed qspec
   * SYNCHRONISATION_FAILED: Mismatch synchronization between an end-to-end RESERVE and an intra-domain RESERVE (see Section 4.6 and 4.7.1)
   * CONGESTION_SITUATION: Possible congestion situation occurred on downstream path
   * QOS Model Error

- o NSLP_objects: data structure that contains a list with QoS-NSLP objects that can be used by QoS-NSLP, when the QNE is a QNI, QNR, QNI_Ingress, QNR_Ingress, QNI_Egress, QNR_Egress:
   * RII
   * RSN
   * BOUND_SESSION_ID list
   * REFRESH_PERIOD
   * SESSION_ID_LIST
   * RSN_LIST
   * MSG_ID
   * BOUND_MSG_ID
- o filter: it represents the MRM related PACKET CLASSIFIER
- o features_out: it represents among others the flags that can be used by the QOS-NSLP for new generated QoS-NSLP messages:
   * BREAK
   * REQUEST REDUCED REFRESHES
   * RESERVE-INIT
   * TEAR
   * REPLACE
   * ACK-REQ
   * PROXY
   * SCOPING
   * BYPASSING: when the outgoing message should be bypassed then it includes the required bypassing level. Otherwise it is empty. It can be set only by QNI_Ingress, QNR_Ingress, QNI_Egress, QNR_Egress. It can be unset only by QNI_Ingress, QNR_Ingress, QNI_Egress, QNR_Egress.
   * BINDING () when BINDING is required then it includes a BOUND_SESSION_ID list. Otherwise it is empty. It can only be requested by the following QNE types: QNI, QNR, QNI_Ingress, QNR_Ingress, QNI_Egress, QNR_Egress.
   * NEW_SID - it requests to generate a new session with a new SESSION_ID. If the QoS-NSLP generates a new SESSION_ID then the QoS-NSLP has to return the value of this new SESSION_ID to the RMF/QOSM. It can be requested by a QNI, QNR, QNI_Ingress, QNI_Egress, QNR_Ingress, QNR_Egress.

  * NEW_RSN - it requests to generate a new RSN. If the QoS-NSLP generates a new RSN then the QoS-NSLP has to return the value of this new RSN to the RMF/QOSM.
  * NEW_RII - it requests to generate a new RII. If the QoS-NSLP generates a new RII then the QoS-NSLP has to return the value of this new RII to the RMF/QOSM.
- GIST_API_triggers: it represents the attributes that are provided to GIST via QoS-NSLP via the GIST API
  * NSLPID
  * SII-Handle
  * Transfer-Attributes
  * GIST-Hop-Count
  * IP-TTL
  * ROUTING-STATE-CHECK (if set it requires from GIST to create a routing state)
- incoming_interface: the ID of the incoming interface. Used only when the QNE reserves resources on incoming interface. Default is 0 (no reservations on incoming interface)
- outgoing_interface: the ID of the outgoing interface. Used only when the QNE reserves resources on outgoing interface. Default is 0 (no reservations on outgoing interface)

A.3. Configuration interface

The config() function is meant for configuring per-session settings, from the RMF towards the NSLP.

```
int config(sid, qne_type, state_type, bypassing_type)
```

- sid: SESSION_ID - The NSIS session identifier
- qne_type: it defines the type of a QNE
  * QNI
  * QNI_Ingress: the QNE is a QNI and an Ingress QNE
  * QNE: the QNE is not a QNI or QNR
  * QNE_Interior: the QNE is an Interior QNE, but it is not a QNI or QNR
  * QNI_Egress: the QNE is a QNI and an Egress QNE
  * QNR
  * QNR_Ingress: the QNE is a QNR and an Ingress QNE
  * QNR_Egress: the QNE is a QNR and an Egress QNE
- state_type: it defines if the QNE keeps QoS-NSLP operational states
  * STATEFULL
  * STATELESS
- bypassing_type: it defines if a QNE bypasses end-to-end messages or not

Appendix B. Appendix B. Glossary

AAA: Authentication, Authorization and Accounting

EAP: Extensible Authentication Protocol

MRI: Message Routing Information (see [I-D.ietf-nsis-ntlp])

NAT: Network Address Translator

NSLP: NSIS Signaling Layer Protocol (see [RFC4080])

NTLP: NSIS Transport Layer Protocol (see [RFC4080])

OPWA: One Pass With Advertising

OSP: Open Settlement Protocol

PIN: Policy Ignorant Node

QNE: an NSIS Entity (NE), which supports the QoS NSLP (see Section 2)

QNI: the first node in the sequence of QNEs that issues a reservation request for a session (see Section 2)

QNR: the last node in the sequence of QNEs that receives a reservation request for a session (see Section 2)

QSPEC: Quality of Service Specification

RII: Request Identification Information

RMD: Resource Management for DiffServ

RMF: Resource Management Function

RSN: Reservation Sequence Number

RSVP: Resource Reservation Protocol (see [RFC2205])

SII: Source Identification Information

SIP: Session Initiation Protocol

SLA: Service Level Agreement

Authors' Addresses

Jukka Manner
University of Helsinki
P.O. Box 68
University of Helsinki FIN-00014 University of Helsinki
Finland

Email: jmanner@cs.helsinki.fi

Georgios Karagiannis
University of Twente/Ericsson
P.O. Box 217
Enschede 7500 AE
The Netherlands

Email: karagian@cs.utwente.nl

Andrew McDonald
Siemens/Roke Manor Research
Roke Manor Research Ltd.
Romsey, Hants S051 0ZN
UK

Email: andrew.mcdonals@roke.co.uk

Full Copyright Statement

Intellectual Property

Acknowledgment

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