Next Steps in Signalling
Internet-Draft
Expires: April, 2006

J. Manner (ed.)
University of Helsinki
G. Karagiannis
University of Twente/Ericsson
A. McDonald
Siemens/Roke Manor Research
S. Van den Bosch
Alcatel
October 2005

NSLP for Quality-of-Service signalling
<draft-ietf-nsis-qos-nslp-08.txt>

Status of this Memo

Abstract

This draft describes an NSIS Signalling Layer Protocol (NSLP) for signalling 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 draft explains the overall protocol approach, design decisions made and provides examples. It specifies object and message formats and processing rules.

Table of Contents

1. Introduction

1.1. Scope and background

This document defines a Quality of Service (QoS) NSIS Signalling 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 signalling protocols, whose structure is outlined in the NSIS

framework [RFC4080]; this defines a common NSIS Transport Layer Protocol (NTLP) which QoS NSLP uses to carry out many aspects of signalling message delivery. A specification of the NTLP, GIST [I-D.ietf-nsis-ntlp] is done in another document.

The design of 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 signalling path). Although there is no backwards compatibility at the level of protocol messages, interworking with RSVP at a signalling application gateway would be possible in some circumstances. 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 no explicit support for IP multicast.

A distinction is made between the operation of the signalling protocol and the information required for the operation of the Resource Management Function (RMF). This document describes the signalling 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 approach to protocol design is outlined in Section 3.1. The operation and use of QoS NSLP is then clarified by means of a number of examples in Section 4. These sections should be read by readers interested in the

protocol capabilities. The functional specification Section 5 contains more detailed object and message formats and processing rules and should be the basis for implementers. The subsequent sections describe extensibility (IANA), requirements on GIST API and security considerations.

1.2. Model of operation

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 1.

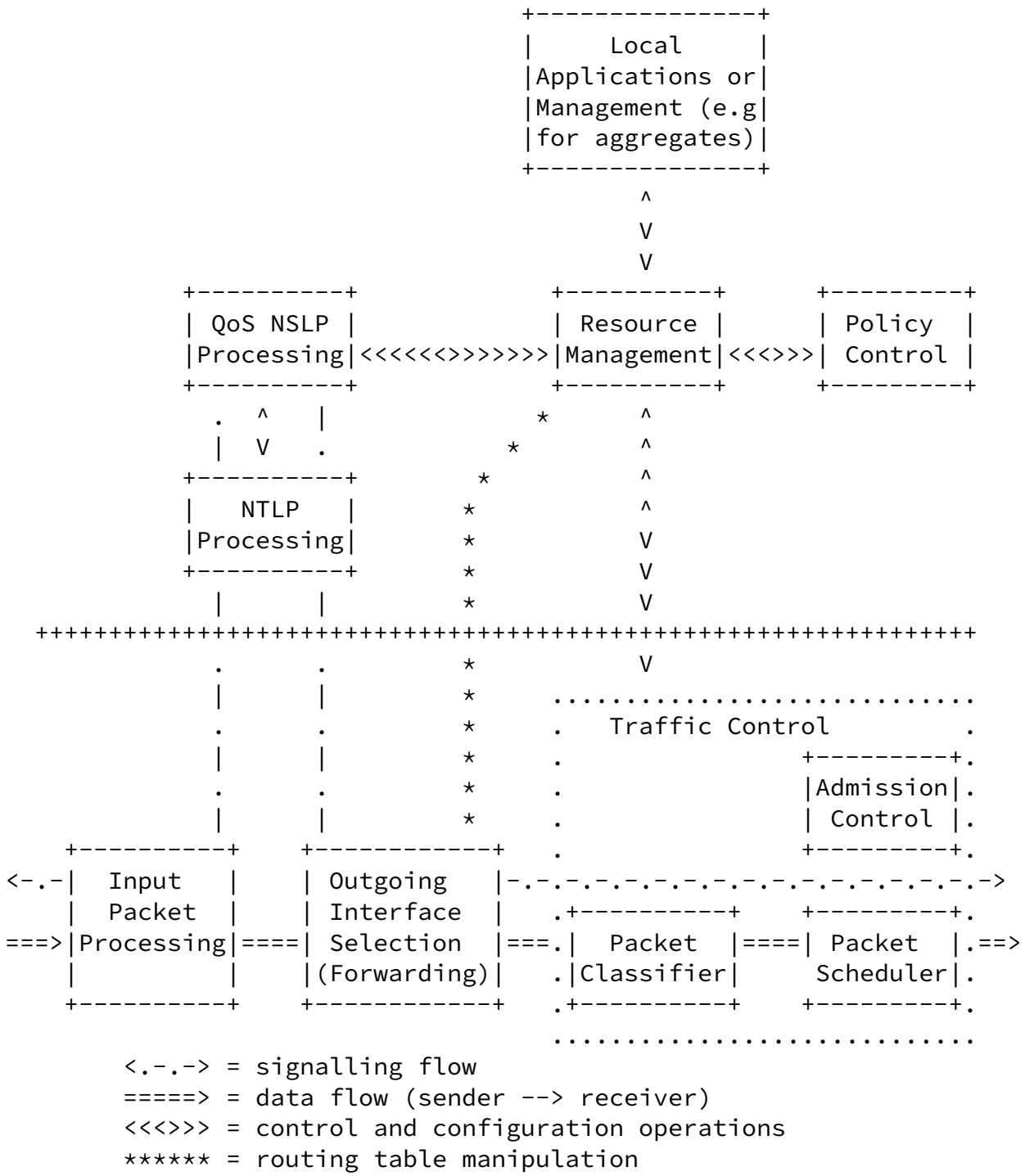


Figure 1: QoS NSLP in a Node

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 more than a simple interaction with the Resource Management Function.

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.

- o 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.

- o 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 a local 'resource management' function, which coordinates the activities required to grant and configure the resource. It also handles policy-specific aspects of QoS signaling.

- o The grant processing involves two local decision modules, 'policy control' and 'admission control'. Policy control determines whether the user has administrative permission to make the reservation. Admission control determines whether the node has sufficient available resources to supply the requested QoS.

- o 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 forwarding 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 a AAA service external to the node itself. Some discussion can be found in a separate document on AAA issues [I-D.tschofenig-nsis-aaa-issues] and one on authorization issues [I-D.tschofenig-nsis-qos-authz-issues]. 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; this is not visible to the protocol operation,

although it may have some influence on the detailed design of protocol messages to allow the stub to be minimally complex. A more detailed discussion on authentication and authorization can be found in Section 3.1.4.

The group of user plane functions, which implement QoS for a flow (admission control, packet classification, and scheduling) is sometimes known as 'traffic control'.

Admission control, packet scheduling, and any part of policy control beyond simple authentication have to be implemented using specific definitions for types and levels of QoS; Our assumption is 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 3.1.3.

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 acknowledgement message in one direction, and may forward the resource request in the other. Message routing is (by default) carried out by the GIST module. Note that while Figure 1 shows a unidirectional data flow, the signalling messages can pass in both directions through the node, depending on the particular message and orientation of the reservation.

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.

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.

Session: A "session" is essentially a signaling application concept, since it is only used in non-trivial state management actions that are application specific. 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. The session identifier should be (probabilistically) globally unique, and it should not be modified end-to-end.

Session Identification (SESSION_ID, SID): This is a cryptographically random and (probabilistically) globally unique identifier of the application layer session that associated with a certain flow. For a given flow, different signaling applications may or may not use the same session identifier. Often there will only be one 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 signalling message (maybe the upstream or the downstream peer). NB: this is not necessarily the QNI.

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

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

Figure 2 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.

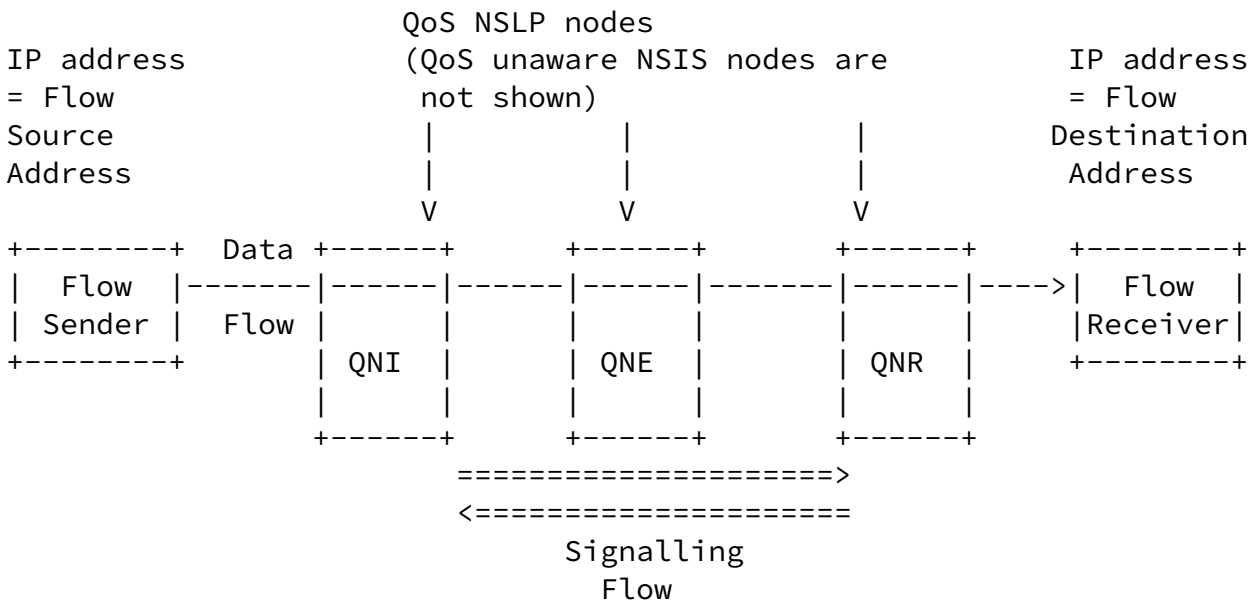


Figure 2: Components of the QoS NSLP architecture.

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

3. Protocol Overview

3.1. Overall approach

3.1.1. GIST Interactions

The QoS NSLP uses GIST for delivery of all its messages. Messages are normally passed from the NSLP to GIST via an API (defined in Appendix D of [I-D.ietf-nsis-ntlp]), which also specifies additional information, including an identifier for the signalling application (e.g. 'QoS NSLP'), the flow/session identifier, 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, flow routing information) can be transferred across this interface.

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.

3.1.2. 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 RESERVE message is idempotent; the resultant effect 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 'probe' the network for path characteristics, for receiver- initiated 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. It does not cause QoS NSLP state to be installed in nodes other than the one that generated the QUERY.

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 is impotent, it 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 pre-empted.

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 flags. 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 authorise the reservation of resources.

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

3.1.3. QoS Models and QoS Specifications

QoS NSLP provides flexibility over the exact patterns of signalling 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.

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, its content is interpreted by the Resource Management Function and the Policy Control Function for the purposes of policy control and traffic control (including admission control and configuration of the packet classifier and scheduler).

QoS NSLP does not mandate any particular behaviour for the RMF, instead demanding interoperability at the signalling protocol whilst leaving the validation of RMF behaviour to SLAs or contracts external to the protocol itself. The RMF may make use of various elements from the QoS NSLP message, not only the QSPEC object.

The QSPEC carries a collection of objects that can describe QoS

specifications in a number of different ways. A QSPEC will usually contain some objects which need to be understood by all implementations, and it can also be enhanced with additional objects 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.

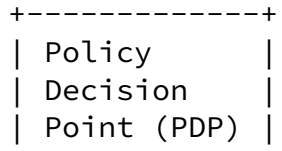
QoS Models may be local (private to one network), implementation/ vendor specific, or global (implementable by different networks and vendors). The authors are currently aware of three efforts related to QoS Model specification: IntServ Controlled Load [I-D.kappler-nsis-qosmodel-controlledload], ITU Y.1541 [I-D.ash-nsis-y1541-qosm], and Resource Management for DiffServ (RMD) [I-D.ietf-nsis-rmd].

The definition of a QoS model may also have implications on how local behaviour should be implemented in the areas where the QoS NSLP gives freedom to implementers. For example, it may be useful to identify recommended behaviour for how a RESERVE message that is forwarded relates to that received, or when additional signalling 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. Moreover, a QoS model may specify reservation pre-emption, e.g., an incoming resource request may cause removal of an earlier reservation.

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.

3.1.4. Policy control

Getting access to network resources typically involves some kind of policy control. One example of this is authorisation of the resource requester. Policy control for QoS NSLP resource reservation signalling is conceptually organised as illustrated below in Figure 3.



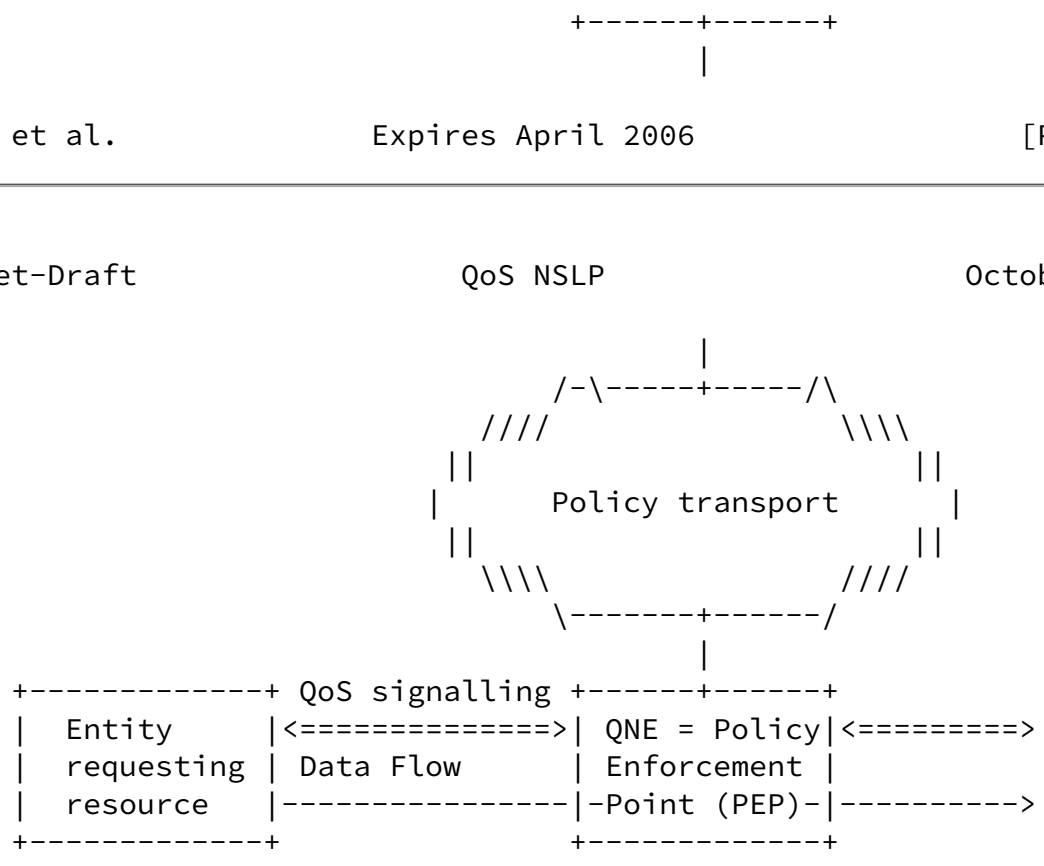


Figure 3: Policy control with the QoS NSLP signaling.

From QoS NSLP point of view, the policy control model is essentially a two-party model between neighbouring 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 input to policy control is referred to as "Policy data", some of which QoS NSLP carries in the POLICY_DATA object while other information is provided across the GIST API. Policy data itself is opaque to the QoS NSLP, which simply passes it to policy control when required. The policy data is independent from the QoS model in use.

Two options are currently considered for support by QoS NSLP policy control:

a. Reuse of GIST channel security mechanisms

b. Carrying (authorisation) tokens

The fist option is preferred as it relies on existing security measures. This can be controlled through the GIST API. The second

option is supported through [FIXME: FILL IN HANNES' WORK]

It is generally assumed that policy enforcement is likely to concentrate on border nodes between administrative domains. In some cases policy objects transmitted across the domain traverse an intermediate Policy Ignorant Node (PIN) that is allowed to process QoS NSLP message but does not handle policy information. The policy peering between ingress and egress edge of a domain therefore relies on the internal chain of trust between the nodes in the domain. If this is not acceptable, a separate signalling session can be set up between the edge node in order to exchange policy information. This is similar to the aggregation mechanism.

3.2. Design decisions

QoS NSLP was designed according to the following principles.

3.2.1. Soft-state

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 signalled for remains valid. Maintaining the reservation beyond this lifetime can be done by sending periodically a RESERVE message.

3.2.2. Sender-receiver initiation

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

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). Before sending the RESERVE, the sender of the data first sends a QUERY message with the R-bit set that creates the required GIST path state. The QUERY message is needed due to the asymmetric nature of IP routing.

Note: these definitions follow the definitions in Section 3.3.1. of RFC 4080 [RFC4080]. The question is, which node is in charge of

requesting and maintaining the resouce 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. Message sequencing

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. Therefore, QoS NSLP supports detection of RESERVE message re-ordering or duplication with Reservation Sequence Number (RSN).

The RSN has local significance only, i.e. between QNEs. Attempting to make an identifier that was unique in the context of a SESSION_ID but the same along the complete path would be very hard. Since RESERVE messages can be sent by any node on the path that maintains reservation state (e.g. for path repair) we would have the difficult task of attempting to keep the identifier synchronized along the whole path. Since message ordering only ever matters between a pair

of peer QNEs, we can make the RSN unique just between a pair of neighbouring stateful QNEs. By managing the sequence numbers in this manner, the source of the RESERVE does not need to determine how the next QNE will process the message.

Note that, since the RSN is unique within a SESSION_ID, it can be used together with a SESSION_ID to refer to particular installed state.

3.2.4. Explicit state installation confirmation and responses

A QoS NSLP instance MAY request an explicit confirmation of its state installation actions from the immediate upstream or downstream peer. This is achieved by using an ACKNOWLEDGE (A) flag in the message header.

In addition to this, a QNE may require other information such as 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 refresh

For scalability, 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). These reduced refreshes require an explicit acknowledgment of state installation to ensure that the RSN reference will be understood. It is up to a QNE that receives a message containing an RII to decide whether it wants to accept reduced refreshes and provide this explicit acknowledgement.

3.2.6. Message scoping

A QNE may use local policy when deciding whether to propagate a message or not. 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, a SCOPING flag limits the part of the path on which state is installed or the downstream nodes that can respond. When set to zero, it indicates that the scope is "whole path" (default). When set to one, the scope is "single hop".

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.

This specification does not support an explicit notion of a region

scope or "to a mobility-related path branch/merge point". If needed, this can be easily proposed as an extension later on,e.g. based on LRSVP [I-D.manner-lrsvp].

3.2.7. Session binding

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

Session binding indicates a (possibly asymmetric) 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 relation to another session with SID_B (the bound session) by including a BOUND_SESSION_ID object containing SID_B in its messages. The dependency is asymmetric if the session with SID_B does not carry a BOUND_SESSION_ID object containing SID_A.

The concept of session binding is used to indicate the dependency 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 dependency between the sessions used for forward and reverse reservation. Note that the dependency indicated by session binding is purely informative in nature and does not automatically trigger any action in a QNE. However, a QNE may use the information for local resource optimisation or to tear down reservations that are no longer useful.

3.2.8. Layering

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 control plane 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.

3.2.8.1. 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. QoS NSLP supports this by allowing QSPEC objects to be stacked.

When a domain wants to apply a certain QoS model to an incoming per-flow reservation request, each edge of the domain is configured to map the incoming QSPEC object to a local QSPEC object and push that object onto the stack of QSPEC objects (typically immediately following the Common Control Information, i.e., before the first QSPEC that is found in the message). QNEs inside the domain look at the top of the QSPEC object stack to determine which QoS model to

apply for the reservation.

The position of the local QSPEC object in the stack implies a tradeoff between the speed with which incoming messages can be processed and the time it takes to construct the outgoing message (if any). By mandating the locally valid object to be on top of the stack we value ease of processing over ease of message construction.

Note that the use of multiple QSPEC objects may require complex processing. A QNE while processing the QSPEC on the top of the stack may also need to process inner QSPEC objects. For example, a domain may want to hide its existence from the end hosts, and for doing that may need to process objects and fields in the inner QSPEC used originally by the end hosts.

3.2.8.2. Local control plane properties

The way signalling messages are handled is mainly determined by the parameters that are sent over GIST-NSLP API and by the Common Control Information. A domain may have a policy to implement local control plane behaviour. 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 control plane 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.

Intermediate node bypass 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.

3.2.8.3. 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 signalling messages. The QoS NSLP, therefore, provides aggregation facilities similar to RFC 3175 [RFC3175]. However, the aggregation scenarios supported are wider than that proposed there. Note that 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 signalling support for it.

The essential difference with the layering approaches described in Section 3.2.8.1 and Section 3.2.8.2 is that the aggregate reservation needs a FlowID, ie., MRI, that describes all traffic carried in the

aggregate (e.g. a DSCP in case of IntServ over DiffServ). The need

for a different FlowID mandates the use of two different sessions, similar to Section 3.2.8.2 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.

3.2.9. Priority

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 signalling 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.

3.2.10. Rerouting

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

Rerouting detection can be performed at three levels. First, routing modules may detect route changes through their interaction with

routing protocols. Certain QNEs or GIST implementations may interact with local routing module to receive quick notification of route changes. This is largely implementation-specific and outside of the scope of NSIS. Second, route changes may be detected at GIST layer. This specification requests GIST design to foresee notification of this information over the API. This is outside the scope of the QoS NSLP specification. Third, rerouting may be detected at the NSLP layer. A QoS NSLP node is able to detect changes in its QoS NSLP peers by keeping track of a Source Identification Information (SII) object that is 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- and receiver-oriented reservations, respectively.

Reservation on the new path happens when a refreshing RESERVE message arrives at the QNE where the old and the new path diverge. The refreshing RESERVE will be interpreted as a new RESERVE on the new path. Depending on the transfer mode, this may require installation of a new messaging association. 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 optimisations, 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 incrementing the RSN and then sending a new RESERVE message. On links that are common to the old and the new path, this RESERVE message is interpreted as a refreshing RESERVE. On new links, it creates the reservation.

After the reservation on the new path is set up, the branching node or the merging node may want to tear down the reservation on the old path (faster than what would result from normal soft-state time-out). This functionality is supported by keeping track of the old SII. This specification requests GIST design to provide support for an SII that is interpreted as a random identifier at the QoS NSLP but that allows, when passed over the API, to forward QoS NSLP messages to the QNE identified by that SII.

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 foreseen in the common header, which, when set to FALSE, indicates that the reservation on the old branch should be kept.

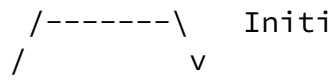
The design of the QoS-NSLP allows reservations to be installed at a subset of the nodes along a path, including cases where those nodes might not included the endpoints of the application data flow.

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 signalling path.

There are three cases for the last node on the signalling 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 signalling 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 signalling. 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.

We can consider two particular cases for rerouting. In the first, referred to as "Path Extension", rerouting occurs such that an additional QNE is inserted into the signalling path between the old last node and the data receiver, as shown in Figure 4.



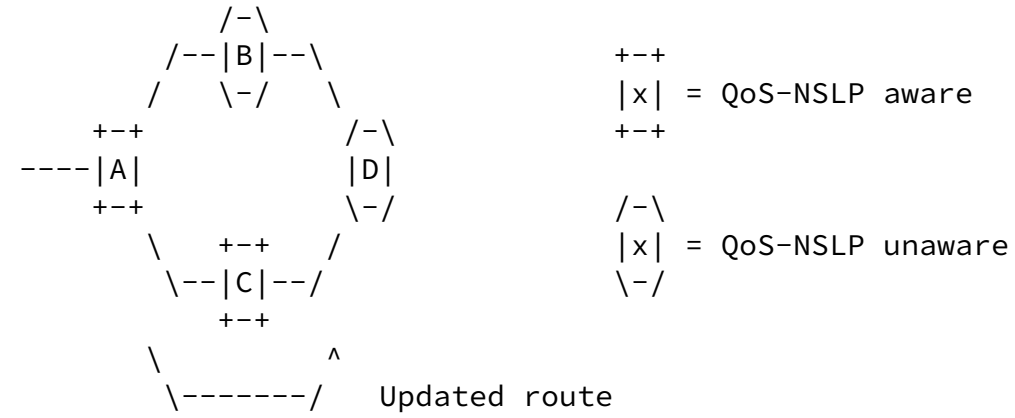


Figure 4: Path Extension

When rerouting occurs, the data path changes from A-B-D to A-C-D. Initially the signalling path ends at A. Despite initially being the last node, node A MUST continue to attempt to send messages downstream to probe for path changes, unless it has been explicitly configured as a signalling proxy for the data flow receiver. This is required so that the signalling 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 signalling path is no longer on the data path. This is shown in Figure 5.

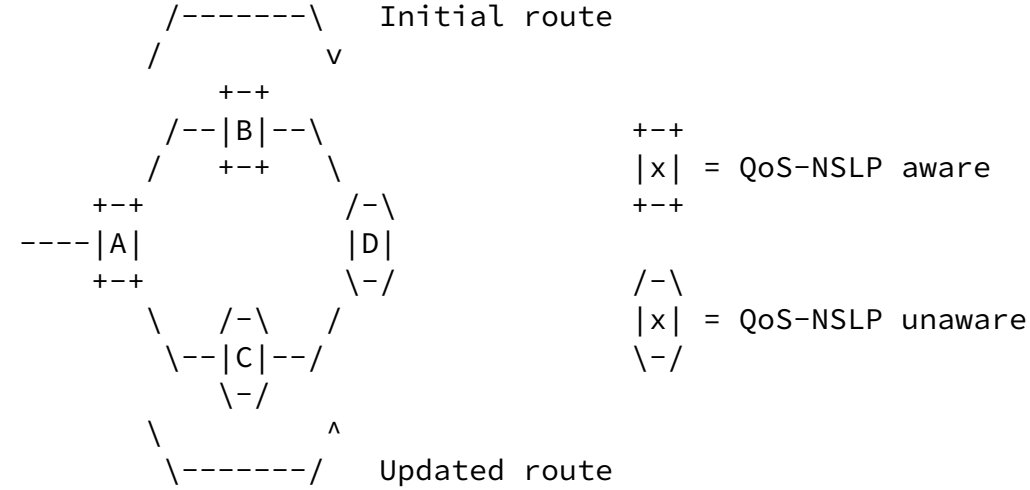


Figure 5: Path Truncation

When rerouting occurs, the data path again changes from A-B-D to A-C-D. The signalling path initially ends at C, 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 signalling application that no downstream GIST nodes supporting the QoS-NSLP are present. Node A MUST take over as the last node on the signalling path

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. Basic sender-initiated reservation

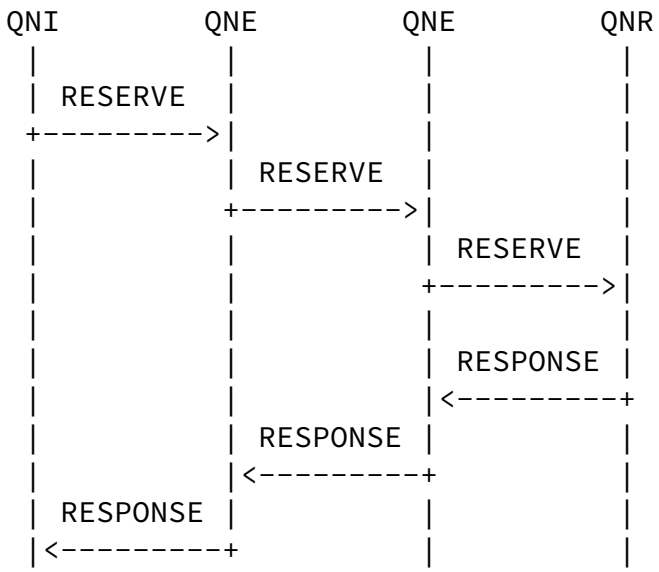


Figure 4: Basic Sender Initiated Reservation

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 some GIST functionality to determine that there are no more QNEs between this node and the data flow destination.

A node can ask a confirmation of the installed state from its immediate peer. It does so by setting the A flag, which causes a RESPONSE message to be sent by the immediate peer. One use of this is to confirm the installation of state, which allows the use of reduced refreshes that later refer to that state. A RESPONSE message can also indicate an error when, e.g., a reservation has failed to be installed.

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, but may simply remember that RII to match the related RESPONSE it is interested in later. When it receives the RESPONSE, it forwards the RESPONSE upstream towards the RII originating node.

The RESPONSE 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 the reverse-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, it 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 object used to match the response back to the query, and an indicator of the query scope (next node, whole path). The QUERY message is passed to GIST to forward it along the path.

A QNE (including the QNR) 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. Into this is copied various objects from the received QUERY 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 back down the path.

4.3. Basic receiver-initiated reservation

As described in the NSIS framework [RFC4080] in some signalling 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) [XREF 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 5). The QUERY must have the R-bit 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), into 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) [XREF OPWA95]. This QUERY message causes GIST reverse-path state to be installed.

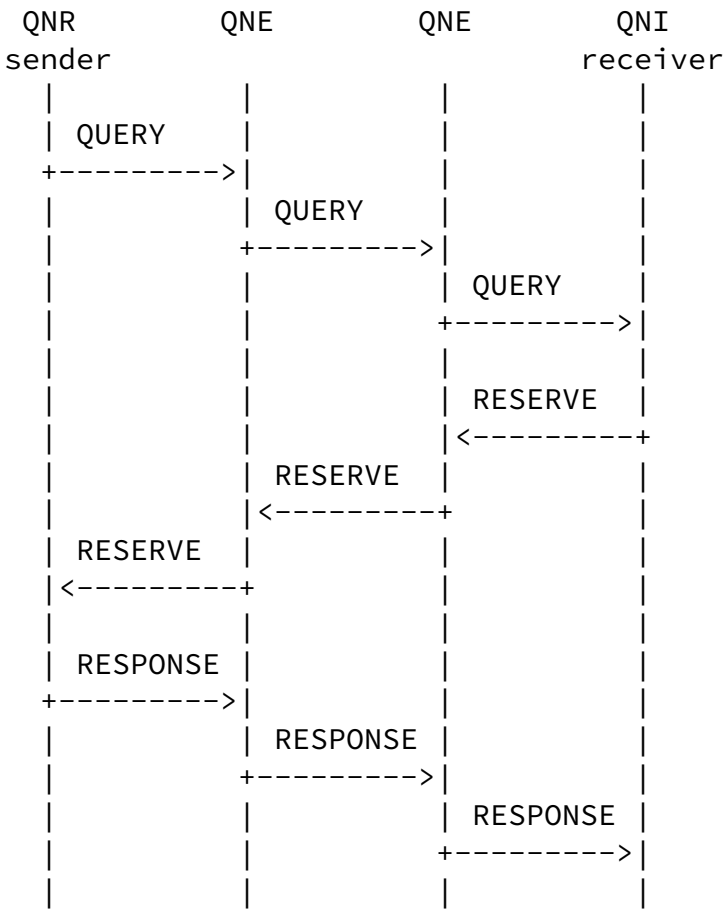


Figure 5: Basic Receiver Initiated Reservation

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 R-bit 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 in the same way as for the sender-initiated approach. This RESERVE message may be also used to refresh GIST reverse path state. Alternatively, refreshing GIST reverse path state could be performed by sending periodic QUERY messages, which are needed in case of route changes anyway.

4.4. Bidirectional Reservations

Bidirectional reservations are supported by binding two uni-directional sessions together. We distinguish two cases:

o Binding two sender-initiated reservations, 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, 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 reservation is shown on Figure 6. 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 on the picture for simplicity.

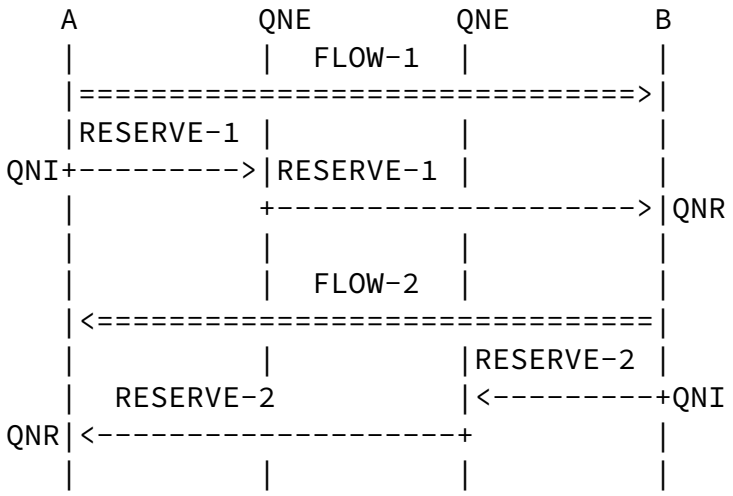
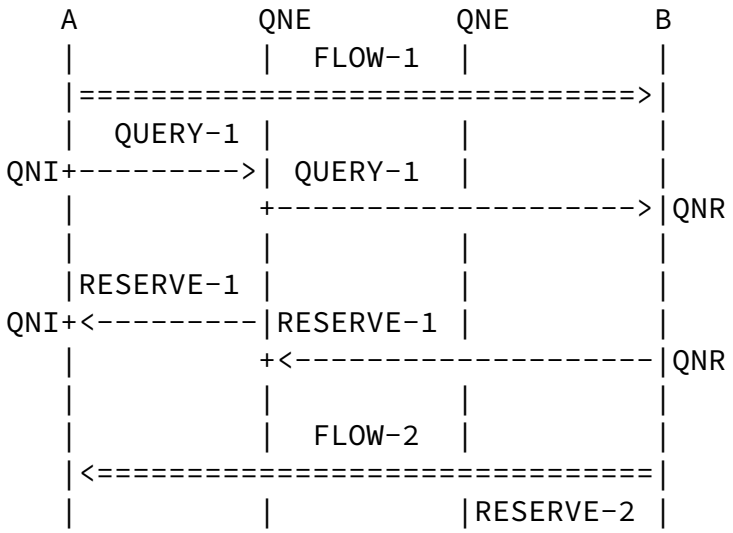


Figure 6: Bi-directional reservation for sender+sender scenario

The scenario with a sender-initiated and a receiver-initiated reservation is shown on Figure 7. In this case, QNI B sends out two RESERVE messages, one for the sender-initiated and one for the receiver-initiated reservation.



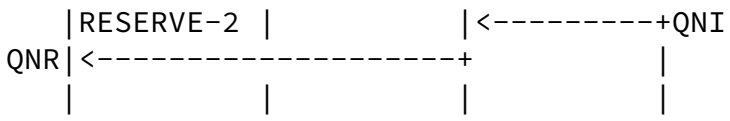


Figure 7: Bi-directional reservation for sender+receiver scenario

4.5. Use of Local QoS Models

In some cases it may be required to use a different QoS model along a particular segment of the signalling. In this case a node at the edge of this region needs to add the additional local QSpec information, based on the end-to-end QSpec. This allows the QoS description to be tailored to the QoS provisioning mechanism available in the network.

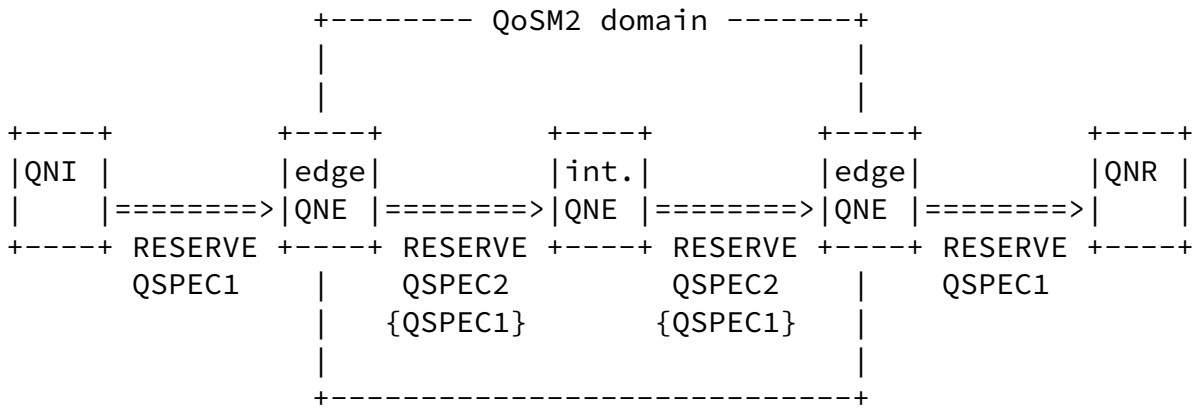


Figure 8: Reservation with local QoS Models

This initially proceeds as for the basic example, with peer-to-peer installation of reservations. However, within a region of the network a different QoSM (QoSM2) needs to be used. At the edge of this region the QNEs support both the end-to-end and local QoS models. When the RESERVE message reaches the QNE at the ingress, the initial processing of the RESERVE proceeds as normal. However, the QNE also determines the appropriate description using QoSM2. The RESERVE message to be sent out is constructed mostly as usual but with a second QSPEC object added on top, which becomes the 'current' one.

When this RESERVE message is received at an node internal to the QoSM2 domain the QoS NSLP only uses the local QSPEC, rather than the end-to-end QSPEC. Otherwise, processing proceeds as usual. The RESERVE message that it generates should include both of the QSPECs

from the message it received.

At the QNE at the egress of the region the local QSPEC is removed from the message so that subsequent QNEs receive only the end-to-end QSPEC.

A message can contain at most two QSpec objects, i.e. the end-to-end QSpec and a local QSpec.

4.6. Aggregate Reservations

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

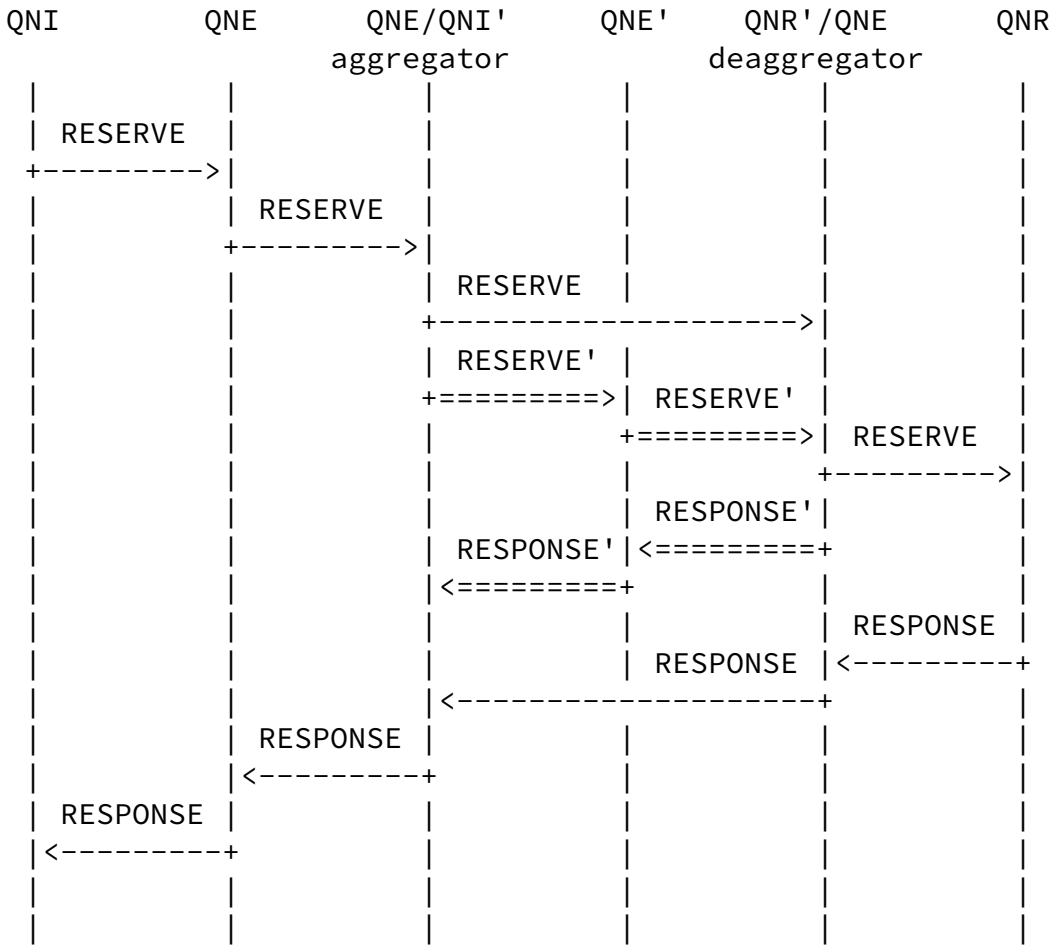


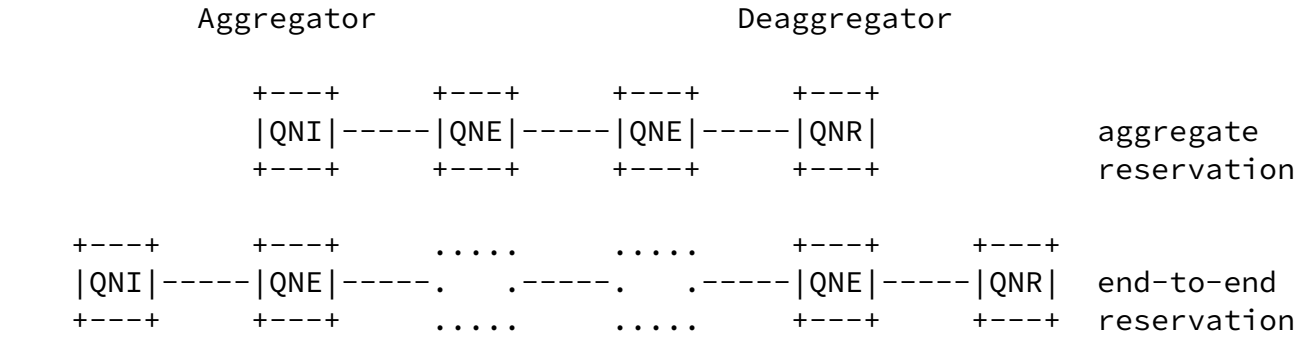
Figure 9: Sender Initiated Reservation with Aggregation

An end-to-end per-flow reservation is initiated as normal (with messages shown in Figure 9 as "RESERVE").

At the aggregator a reservation for the aggregated flow is initiated (shown in Figure 9 as "RESERVE'"). This may use the same QoS model as the end-to-end reservation but has a flow identifier for 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. The marking MUST be accomplished by the Aggregator by modifying the QoS-NSLP default NSLP-ID value to a NSLP-ID predefined value. The De-aggregator MUST stop this marking process by reassigning the QoS-NSLP default NSLP-ID value to these signaling messages. The deaggregator then becomes the next hop QNE for the end-to-end per-flow reservation.



The deaggregator acts as the QNR for the aggregate reservation.

Information is carried in the reservations to enable the deaggregator to associate the end-to-end and aggregate reservations with one another.

The key difference between this example, and previous ones 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.

4.7. 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 ´local' reservation can be different from that of the end-to-end 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.

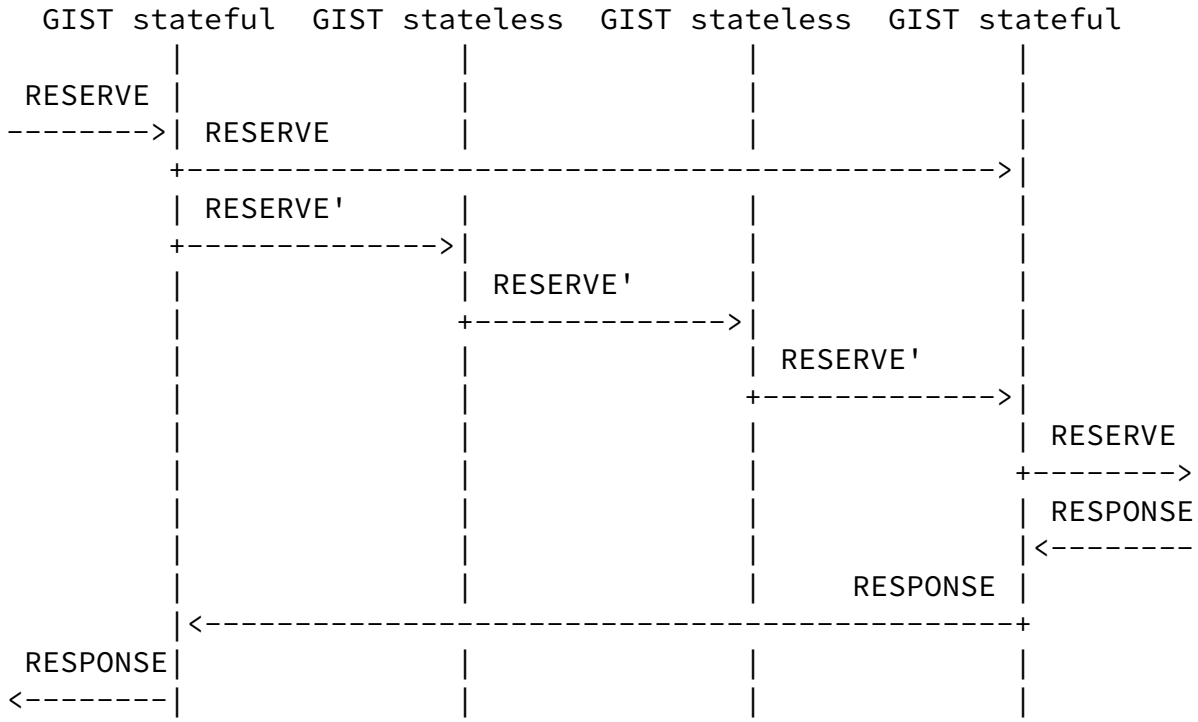


Figure 11: Sender-initiated reservation with Reduced State Interior Nodes

The QNI performs the same processing as before to generate the initial RESERVE message, and it is forwarded by GIST as usual. At the QNEs at 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. 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 stop this

marking process by reassigning the QoS-NSLP default NSLP-ID value to the original 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 that session. After the initial discovery phase using unreliable GIST transfer mode, reliable GIST transfer mode between the ingress and egress can be used. At the egress node the RESERVE message is then forwarded normally.

At the ingress a second RESERVE' message is also built. 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, RESERVE' visits the same nodes that were visited, including the egress QNE.

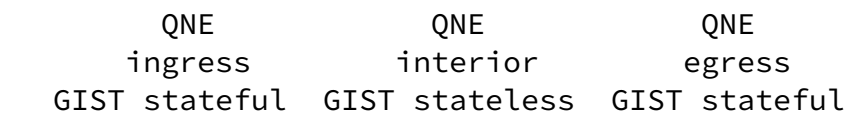
When processed by interior (stateless) nodes the QoS NSLP processing exercises its options to not keep state wherever possible, so that no per flow QoS NSLP state is stored. Some state, e.g. per class, for the QSPEC related data may be held at these interior nodes. The QoS NSLP also requests that GIST use different transport characteristics

(i.e. 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 reduced refreshes).

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 signalling 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.

Since GIST neighbour relations are not maintained in the reduced-state region, only sender initiated signalling can be supported. If a receiver-initiated reservation over a stateless or reduced state domain is required this can be implemented as shown below.



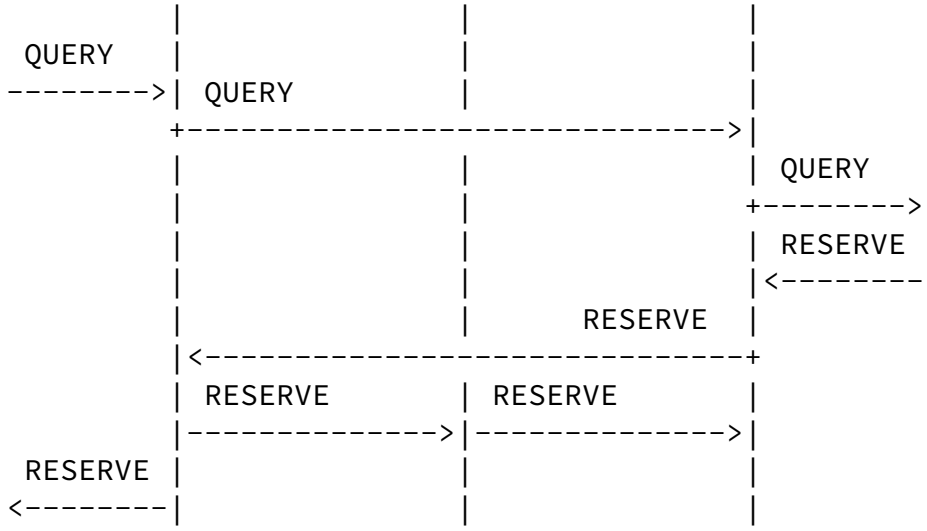


Figure 12: Receiver-initiated reservation with Reduced State Interior Nodes

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 knows it needs to send both a sender-initiated RESERVE over the stateless domain and send a RESERVE message further upstream.

4.8. Re-routing scenario

The QoS NSLP needs to adapt to route changes in the data path. This assumes the capability to detect rerouting events, perform QoS

reservation on the new path and optionally tear down reservations on the old path.

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 incrementing the RSN and sending a RESERVE message. On links that are common to the old and the new path, this RESERVE message is interpreted as a refreshing RESERVE. On new links, it creates the reservation.

After the reservation on the new path is set up, the branching node or the merging node may want to tear down the reservation on the old path (faster than what would result from normal soft-state time-out). This functionality is supported by keeping track of the old SII. This specification requests GIST design to provide support for an

SII. The SII is opaque to the QoS NSLP, i.e. QoS NSLP does not make any assumptions on how this identifier is constructed. When passed over the API, it allows QoS NSLP to indicate that its messages should be sent to the QNE identified by that SII.

In case of a receiver-initiated reservation, a QNE can detect a route change by receiving a RESERVE message with a different SII. In case of a sender-initiated reservation, the same information is learned from a RESPONSE message, or from a NOTIFY message sent by the downstream peer. A QNE that has detected the route change via the SII change sends a RESERVE message towards the QNR on the old path (using the old SII) with the TEAR flag set. Note that in case of receiver-initiated reservations, this involves A QNE that is notified of the route change in another way and wants to tear down the old branch needs to send the RESERVE on the new path with an RII object. When it receives the RESPONSE message back, it can check whether its peer has effectively changed and send a RESERVE with the TEAR flag set if it has. Otherwise, teardown is not needed. A QNE that is unable to support an RII or does not receive a RESPONSE needs to rely on soft-state timeout on the old branch.

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. In that case, it sets the REPLACE flag in the common header to zero. 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.

4.9. Authorization Model Examples

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

4.9.1. Authorization for the two party approach

The two party approach is conceptually the simplest authorization model.

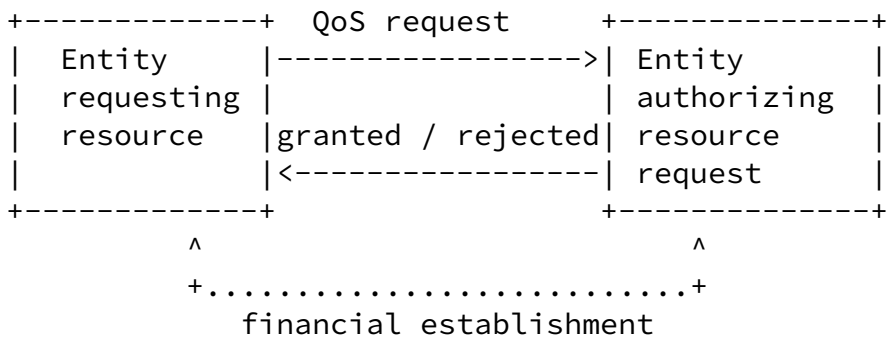


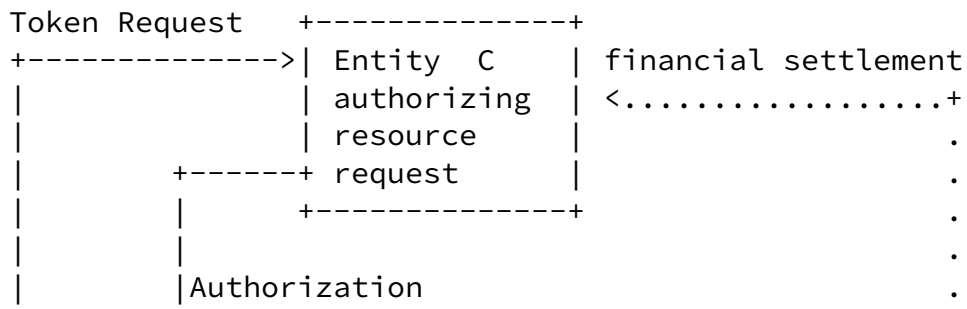
Figure 13: Two party approach

In this example the authorization decision only involves the two entities, or makes use of previous authorisation 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 neighbouring networks (inter-domain signalling) where a long-term contract (or other out-of-band mechanisms) exists to manage charging and provides sufficient information to authorize individual requests.

4.9.2. Token based three party approach

An alternative approach makes use of authorization tokens, such as those described in RFC 3520 [RFC3520] and RFC 3521 [RFC3521] or used as part of the Open Settlement Protocol [OSP]. The former ('authorization tokens') are used to associate two different signalling protocols (i.e. SIP and NSIS) and their authorization with each other whereas the latter is a form of digital money. 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 14. If the request is authorized, then the SIP signalling returns an authorization token which can be included in the QoS signalling 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.



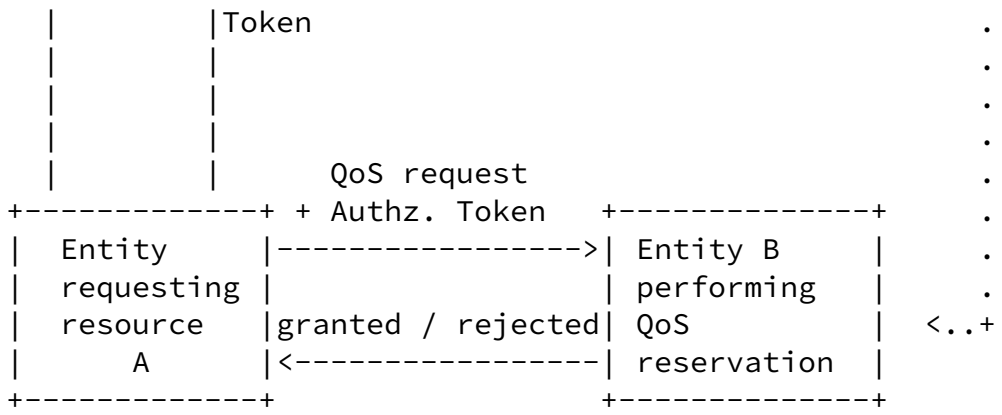


Figure 14: Token based three party approach

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.

4.9.3. 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 15.

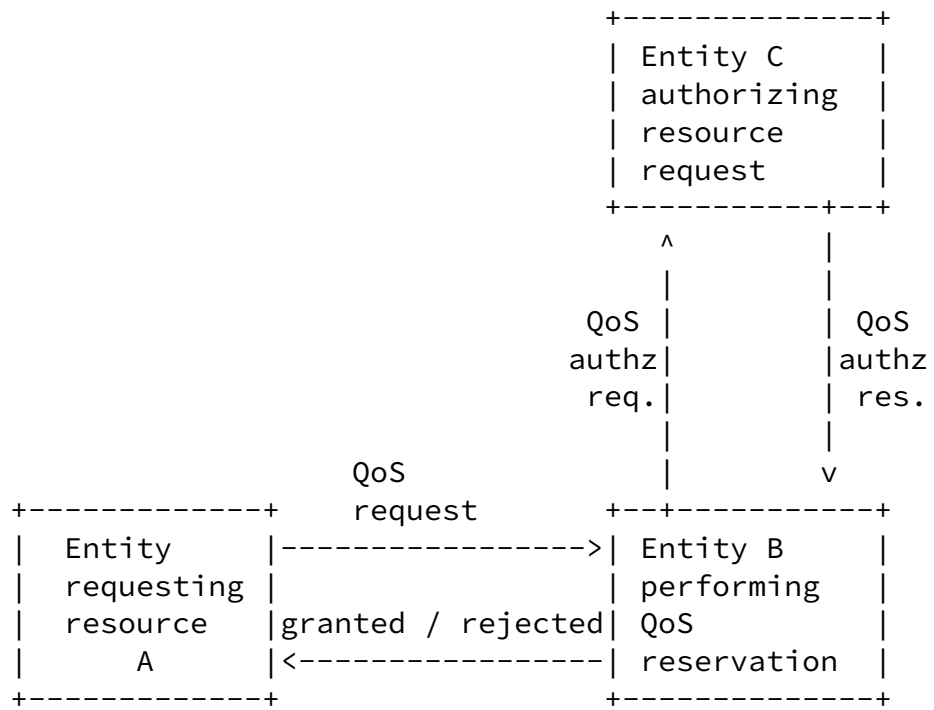


Figure 15: Three party approach

Authorization may be performed on a per-request basis, periodically, or on a per-session basis. The authorization request might make use of EAP authentication between entities A and C, and a subsequent protocol exchange between A and B to create a secure channel for further communications. Such a technique gives flexibility in terms of the authentication and key exchange protocols used.

A further extension to this model is to allow Entity C to reference a AAA server in the user's home network when making the authorization decision.

5. QoS NSLP Functional specification

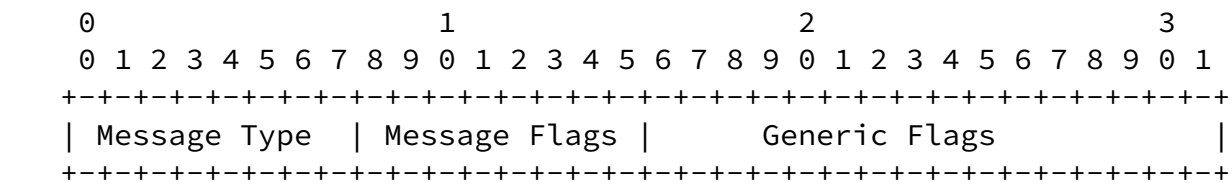
5.1. QoS NSLP Message and Object Formats

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 2234 [RFC2234]. 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 accept the objects in any order, except for QSPEC object(s) which always appear last in the message, and whose mutual order matters.

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).



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

Generic flags: 16 bits

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.

5.1.2. Message formats

5.1.2.1. RESERVE

The format of a RESERVE message is as follows:

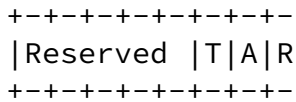
```
RESERVE = COMMON_HEADER
          RSN [ RII ] [ REFRESH_PERIOD ] [ BOUND_SESSION_ID ]
          [ POLICY_DATA ] *2QSPEC
```

The RSN is the only mandatory object and MUST always be present.

If any QSPEC objects are present, they MUST occur at the end of the

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

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

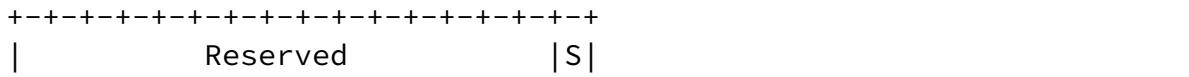


TEAR (T) - when set, indicates that reservation state and QoS NSLP operation state should be torn down. This is indicated to the RMF. Depending on the QoS model, the tear message may include a QSPEC to further specify state removal.

ACKNOWLEDGE (A) - when set, indicates that an explicit confirmation of the state installation action is REQUIRED. This flag SHOULD be set on transmission by default.

REPLACE (R) - when set, indicates that a RESERVE with different Message Routing Information (MRI) 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.

One generic flag is used with the RESERVE message:



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").

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 MUST include the SESSION_ID of that other session in a BOUND_SESSION_ID object.

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 signalling 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. If the routing path changed due to mobility, 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 new MRI and will be propagated end-to-end without requesting a RESPONSE.

Note: It is possible for a host to use this mechanism to constantly force the QNEs on the path to send refresh 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 ]
        [ POLICY_DATA ] *2QSPEC
```

A QUERY message MUST contain an RII object to match an incoming

RESPONSE to the QUERY, unless the QUERY is being used to initiate reverse-path state for a receiver-initiated reservation.

A QUERY message MAY contain one or two QSPEC objects and a POLICY_DATA object. The QSPEC object describes what is being queried for and may contain objects that gather information along the data path. The POLICY_DATA object authorizes the requester of the QUERY message. If any QSPEC objects are present, they MUST occur at the end of the message. There are no other requirements on transmission order, although the above order is recommended.

One message-specific flag is 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.

One generic flag is defined for use in the common header with the QUERY message. This is:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Reserved           |S|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SCOPING (S) - when set, indicates that the message is scoped an 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").

If the session of this message is bound to another session, then the RESERVE message MUST include the SESSION_ID of that other session in a BOUND_SESSION_ID object.

5.1.2.3. RESPONSE

The format of a RESPONSE message is as follows:

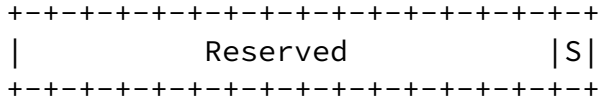
```
RESPONSE = COMMON_HEADER
           [ RII / RSN ] INFO_SPEC
           *2QSPEC
```

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.

If any QSPEC objects are present, they MUST occur at the end of the message. There are no other requirements on transmission order, although the above order is recommended.

No message-specific flags are defined.

One generic flag is defined for use in the common header with the RESPONSE message. This is:



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").

5.1.2.4. NOTIFY

The format of a NOTIFY message is as follows:

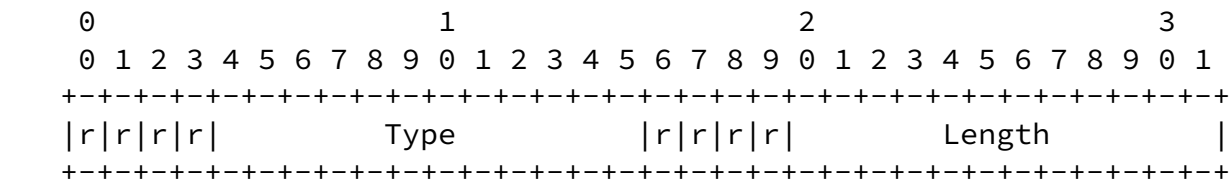
```
NOTIFY = COMMON_HEADER
         INFO_SPEC *2QSPEC
```

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

No flags are defined for use with the NOTIFY message.

5.1.3. Object Formats

The QoS NSLP uses the Type-Length-Value (TLV) object format defined by GIST [I-D.ietf-nsis-ntlp]. Every object consists of one or more 32-bit words with a one-word header. For convenience the standard object header is shown here:



The value for the Type field comes from GIST object type space. 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 object diagrams here use '//' to indicate a variable sized field and ':' to indicate a field that is optionally present.

A QoS NSLP implementation must recognize objects of the following types: RII, RSN, REFRESH_PERIOD, BOUND_SESSION_ID, INFO_SPEC, QSPEC and POLICY_DATA.

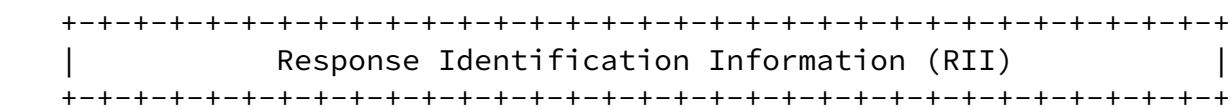
NB: This draft does not currently include the codepoints for the QoS NSLP related object types. The object header is followed by the Value field, which varies for different objects. The format of the Value field for currently defined objects is specified below.

5.1.3.1. Request Identification Information (RII)

Type: RII

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.

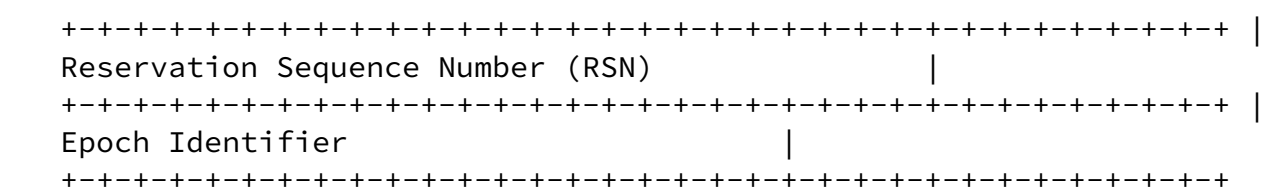


5.1.3.2. Reservation Sequence Number (RSN)

Type: RSN

Length: Fixed - 2 32-bit word

Value: An incrementing sequence number that indicates the order in which state modifying actions are performed by a QNE, and an epoc identifier to allow the identification of peer restarts. The RSN has local significance only, i.e. between a pair of neighbouring stateful QNEs.

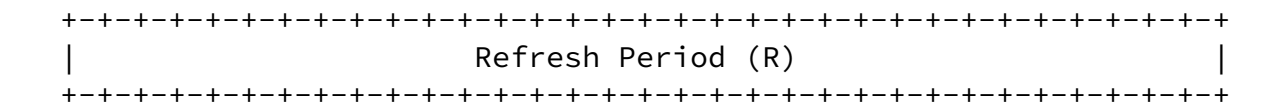


5.1.3.3. REFRESH_PERIOD

Type: REFRESH_PERIOD

Length: Fixed - 1 32-bit word

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



5.1.3.4. BOUND_SESSION_ID

Type: BOUND_SESSION_ID

Length: Fixed - 4 32-bit words

Value: 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.

5.1.3.5. PACKET_CLASSIFIER

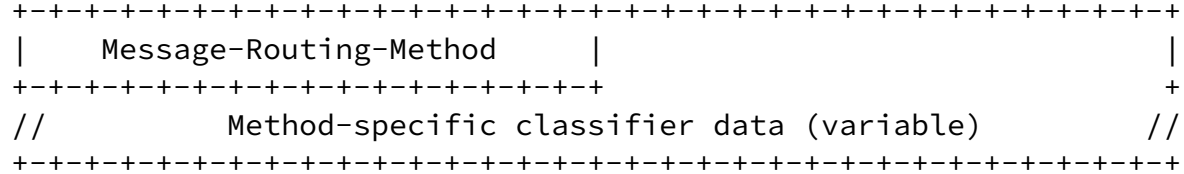
[FIXME: This is a first attempt at this for discussion (see also previous discussion on mailing list). Should the PACKET_CLASSIFIER be in here, or in the QSPEC?]

Type: Packet Classifier

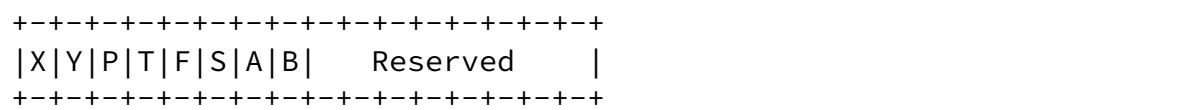
Length: Variable

Value: Contains a 2 byte value indicating the MRM being used, and then additional variable length MRM-specific data

[FIXME: do we need to duplicate the MRM value here? could we just get it from the MRI in the GIST part of the message? however, duplicating it seems not unreasonable from a sanity checking perspective.]



For the basic path-coupled routing MRM, the method specific data is two bytes long and consists of a set of flags:



The flags are:

X - Source Address and Prefix

Y - Destination Address and Prefix

P - Protocol

T - Traffic Class

F - Flow Label

S - SPI

A - Source Port

B - Destination Port

[FIXME: Some of the flag identifiers seem strange. They were selected so that they didn't conflict with any flag names in the GIST MRI, i.e. make D in GIST be something different here]

The flags indicate which fields from the MRI should be used by the packet classifier. Flags MUST only be set if the data is present in the MRI (i.e. if there is a flag for it in GIST, then that must also be set).

The appropriate set of flags set may depend, to some extent, on the QoS model being used.

[FIXME: I assume we don't need flags for prefixes. Do we need separate flags for the two ports?]

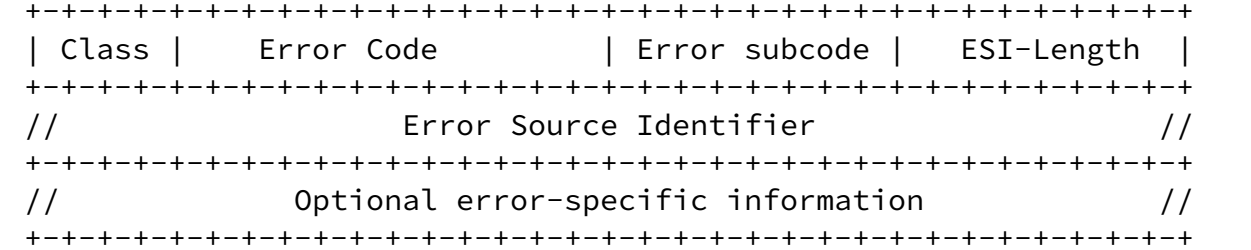
[FIXME: Should this object be mandatory or optional? Optional might mean 'use all information from the MRI'. Currently specified as mandatory.]

5.1.3.6. INFO_SPEC

Type: INFO

Length: Variable

Value: Contains a 4-bit error class, a 12-bit error code, an 8-bit NSLP-specific error subcode, an 8-bit error source identifier length, an error source identifier and optionally variable length error-specific information.



The first four bits of the error code indicates the severity class. The currently defined severity classes are:

- 0x01 - Informational
- 0x02 - Success
- 0x03 - Protocol Error
- 0x04 - Transient Failure
- 0x05 - Permanent Failure
- 0x06 - NSLP-specific Error

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

- Informational:
  - 0x01 - Unknown BOUND_SESSION_ID: the message refers to an unknown SESSION_ID in its BOUND_SESSION_ID object.
- Success:
  - 0x01 - State installation succeeded
  - 0x02 - Reservation created: reservation installed on complete path (sent by last node).
  - 0x03 - Reservation accepted: reservation installed at this QNE, but not yet installed on the rest of the path.
  - 0x04 - Reservation created but modified: reservation installed, but bandwidth reserved was not the maximum requested.

- 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.
  - 0x04 - Mandatory object missing: an object required in a message of this type was missing.
  - 0x05 - Illegal object present: an object was present which must not be used in a message of this type.
  - 0x06 - Unknown object present: an object of an unknown type was present in the message.
  - 0x07 - Wrong object length: the length given for the object did not match the length of the object data present.
  - 0x08 - Unknown QSPEC type: the QoS Model ID refers to a QoS Model which is not known by this QNE.
  - 0x09 - RESERVE received from wrong direction.
- Transient Failure:
  - 0x01 - Requested resources not available
  - 0x02 - Insufficient bandwidth available
  - 0x03 - Delay requirement cannot be met
  - 0x04 - Transient RMF-related error
  - 0x05 - Resources pre-empted
  - 0x06 - No GIST reverse-path forwarding state
  - 0x07 - NSLP soft-state expired

* 0x08 - No path state for RESERVE, when doing a receiver-oriented reservation

* 0x09 - Reservation pre-empted

* 0x10 - RII conflict

o Permanent Failure:

* 0x01 - Authentication failure

* 0x02 - Unable to agree transport security with peer

* 0x03 - Internal or system error

* 0x04 - Resource request denied (authorization failed)

* 0x05000005 - Permanent RMF-related error

o NSLP-specific Error:

This error class may be used, if an NSLP needs to indicate errors, which do not fit any of the pre-defined error levels. The interpretation of these errors is defined in each NSLP separately.

Values in the error subcode field are defined in each NSLP separately. For the QoS NSLP, these are defined in the different QoS model specifications.

5.1.3.7. QSPEC

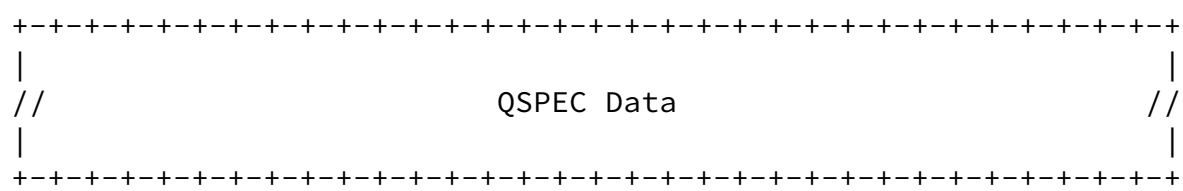
Type: QSPEC

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].



5.1.3.8. POLICY_DATA

POLICY_DATA objects may contain various items to authenticate the user and allow the reservation to be authorised. Some related issues are also discussed in Section 3.1.4.

[FIXME: Need to fix this when Hannes is done]

5.2. General Processing Rules

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 organised in a table with the following structure. The primary key (index) for the table is the SESSION_ID:

SESSION_ID

A large identifier provided by GIST or set locally.

The state information for a given key includes:

Flow ID

Copied from GIST. Several entries are possible in case of mobility events.

QoS Model ID

32 bit identification of the QoS Model.

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 each upstream peer

The RSN is a 32 bit counter.

Current own RSN

A 32 bit random number.

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 signalled for remains valid.

BOUND_SESSION_ID

The BOUND_SESSION_ID is a 128 bit random number.

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 along the entire 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 flag 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 an 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 behaviour depends on the object type value. The usual operation is to skip unknown objects. See the GIST specification for more information.

5.2.4. Retransmissions

QoS-NSLP messages for which a response is requested but fail to elicit a response are retransmitted. 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.

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

5.3. Object Processing

5.3.1. Reservation Sequence Number (RSN)

A QNE's own RSN is a sequence number which applies to a particular NSIS signalling session (i.e. with a particular GIMPS 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 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 (provided that admission control and policy control succeed), and the stored RSN value updated to that for the new reservation. 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 some types of 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 totally 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 not 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.

If an intermediate QNE wants to request a response for an outgoing message, but the message already included an RII when it arrive, the QNE must not add a new RII object nor replace the old RII object, but may simply remember that RII to match the related RESPONSE it is interested in later. 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 either

- The SCOPING flag is set to one ('next hop' scope), or
- This QNE is the last one on the path for the given session.

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

A message contains at most one RII object that is unique within a session and different for each message, in order to allow responses to be matched back to requests (without incorrectly matching at other nodes). Downstream nodes that desire responses may keep track of this RII to identify the RESPONSE when it passes back through them.

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 tearing RESERVE for an aggregate reservation, it MAY use this information to initiate a tearing RESERVE for end-to-end sessions bound to the aggregate.

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. 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.

In more detail (quoting directly from RFC2205):

1. Floyd and Jacobson [XREF FJ94] have shown that periodic messages generated by independent network nodes can become synchronized. This can lead to disruption in network services as the periodic messages contend with other network traffic for link and forwarding resources. Since QoS NSLP sends periodic refresh

messages, it must avoid message synchronization and ensure that any synchronization that may occur is not stable. For this reason, it is recommended that the the refresh timer should be randomly set to a value in the range [0.5R, 1.5R].

2. To avoid premature loss of state, L must satisfy L >= (K + 0.5)*1.5*R, where K is a small integer. Then in the worst case, K-1 successive messages may be lost without state being deleted. To compute a lifetime L for a collection of state with different R values R0, R1, ..., replace R by max(Ri).

Currently K = 3 is suggested as the default. However, it may be necessary to set a larger K value for hops with high loss rate. K may be set either by manual configuration per interface, or by some adaptive technique that has not yet been specified.

3. Each RESERVE message carries a REFRESH_PERIOD object containing the refresh time R used to generate refreshes. The recipient node uses this R to determine the lifetime L of the stored state created or refreshed by the message.

4. The refresh time R is chosen locally by each node. If the node does not implement local repair of reservations disrupted by route changes, a smaller R speeds up adaptation to routing

changes, while increasing the QoS NSLP overhead. With local repair, a router can be more relaxed about R since the periodic refresh becomes only a backstop robustness mechanism. A node may therefore adjust the effective R dynamically to control the amount of overhead due to refresh messages.

The current suggested default for R is 30 seconds. However, the default value Rdef should be configurable per interface.

5. When R is changed dynamically, there is a limit on how fast it may increase. Specifically, the ratio of two successive values R2/R1 must not exceed 1 + Slew.Max.

Currently, Slew.Max is 0.30. With K = 3, one packet may be lost without state timeout while R is increasing 30 percent per refresh cycle.

6. To improve robustness, a node may temporarily send refreshes more often than R after a state change (including initial state establishment).

7. The values of Rdef, K, and Slew.Max used in an implementation should be easily modifiable per interface, as experience may lead to different values. The possibility of dynamically adapting K and/or Slew.Max in response to measured loss rates is for future study.

5.3.5. INFO_SPEC

[FIXME: INFO_SPEC processing rules are still to be defined in more detail.]

We must make a distinction between INFO_SPEC messages used to provide non-fatal information, and fatal error messages. Error messages must be generated even if no RII is included in the incoming message.

5.3.6. QSPEC

The contents of the QSPEC depends on the QoS model being used. There is ongoing work to standardised parts of the QSPEC across multiple

QoS models [QoS-Template].

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 that receives a QSPEC stack MUST only look at the top QSPEC in the stack. If this QSPEC is not understood by the RMF, the QNE MUST send an RESPONSE containing an INFO_SPEC and MUST NOT attempt to recover by inspecting the rest of the stack.

Parameters of the QoS Model that is being signalled for are carried in the QSPEC object. 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 QoS NSLP supports this by allowing QSPEC objects to be stacked.

When a domain wants to apply a certain QoS Model to an incoming per-flow reservation request, each edge of the domain is configured to map the incoming QSPEC object to a local QSPEC object and push that object onto the stack of QSPEC objects (typically immediately following the Common Control Information, i.e. the first QSPEC that is found in the message).

A QNE that knows it is the last QNE to understand a local QSPEC object (e.g. by configuration of the egress QNEs of a domain) MUST remove the topmost QSPEC object from the stack. It SHOULD update the underlying QoS Model parameters if needed.

5.4. Message Processing Rules

This section provides rules for message processing. Not all possible error situation 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 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 sender in order to let it know that there is an inconsistency in the states of adjacent QNEs.

Yet, there is a third possible mode of operation when receiving an unexpected or erroneous message. The QNE may consider the incoming message as fully accpetable, and operate as if there was no error in the processing. This may happen, for example, if the QNE knowns it has just rebooted and has lost its signaling states. Now, the QNE may try to act as if "nothing happened". Note that an implementation must carefully consider this behavior.

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. The format of a RESERVE message is repeated here for convenience:

```
RESERVE = COMMON_HEADER
          RSN PACKET_CLASSIFIER [ RII ]
          [ REFRESH_PERIOD ] [ BOUND_SESSION_ID ]
          [ POLICY_DATA ] *2QSPEC
```

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. It SHOULD exercise its local policy in conjunction with the POLICY_DATA object to do this.

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 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. 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 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 exit

in the message.

If the QNE has reservations which are bound to this session (they contained the SESSION_ID of this session in their BOUND_SESSION_ID object), it MUST send a NOTIFY message for each of these reservations with an appropriate INFO_SPEC. The QNE MAY elect to send RESERVE messages with the TEAR flag set for these reservations.

The default behaviour 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 behaviour. 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 mainains 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, it MAY send a NOTIFY message to its upstream peer, indicating the unknown SESSION_ID. If the message was meant as a refresh, the reply indicates a downstream route change to the upstream peer. The upstream peer SHOULD send a complete RESERVE on the new path (new SII). It identifies the old signalling association (old SII) and MAY start sending complete RESERVE messages for other SESSION_IDs linked to this association.

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 SII 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.

If the REPLACE flag is set, the QNE SHOULD update the reservation state according to the QSPEC contained in the message. 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 the ACKNOWLEDGE flag is set, the QNE MUST acknowledge its state installation action. It does so by sending a RESPONSE with an INFO_SPEC indicating that the reservation is installed at the QNE.

If the SCOPING flag is set, or if the QNE is the last QNE on the path to the destination, the QNE MUST send a RESPONSE message.

When a QNE receives a RESERVE message, its processing may involve sending out another RESERVE message. When sending a RESERVE message, the QNE may insert or remove 'local' QSPEC objects from the message. If any QSPEC is present, the first QSPEC MUST NOT be removed when sending on the RESERVE message.

Upon transmission, a QNE SHOULD set the ACKNOWLEDGE flag. It MUST do so if it wishes to use the reduced overhead refresh mechanism described in Section 3.2.5. It MUST NOT send a reduced overhead refresh message (i.e. a RESERVE with a non-incremented RSN and no QSPEC) unless it has received a RESPONSE message for that RESERVE message.

If the session of this message is bound to another session, then the RESERVE message MUST include the SESSION_ID of that other session in a BOUND_SESSION_ID object.

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 R-bit is included in an arrived QUERY.

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 certain QoS models.

The format of a QUERY message is as follows:

```
QUERY = COMMON_HEADER
        [ RII ] PACKET_CLASSIFIER [ BOUND_SESSION_ID ]
        [ POLICY_DATA ] *2QSPEC
```

When a QNE receives a QUERY message the QSPEC is passed to the RMF for processing. The RMF may return a modified QSPEC which 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 R-bit is set. If the bit is set, the QUERY is used to install reverse path state. In this case, if the QNE is not the QNR, it creates a new QUERY message to send downstream. If the QUERY contained a QSPEC, this MUST be passed to the RMF where it MAY be modified by QoS Model specific QUERY processing. If the QNE is the QNR, 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 or the SCOPING flag is set, 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) into the new QUERY message unchanged. 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 R-bit set should always be answered by the QNR. 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.

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 is impotent, it does not cause any state to be installed or modified.

The format of a RESPONSE message is repeated here for convenience:

```
RESPONSE = COMMON_HEADER
           [ RII / RSN ] PACKET_CLASSIFIER
           INFO_SPEC *2QSPEC
```

A RESPONSE message MUST be sent where the QNE is the last node to process a RESERVE or QUERY message containing an RII object (based on scoping of the RESERVE or QUERY, or because this is the last node on the path). In this case, the RESPONSE MUST copy the RII object from the RESERVE or QUERY.

In addition, a RESPONSE message MUST be sent when the ACKNOWLEDGE flag is set or when an error occurs while processing a received message. If the received message contains an RII object, this object MUST be put in the RESPONSE, as described above. If the RESPONSE is sent as a result of the receipt of a RESERVE message without an RII object, then the RSN of the received RESERVE message MUST be copied into the RESPONSE message.

On receipt of a RESPONSE message containing an RII object, the QNE MUST attempt to match it to the outstanding response requests for that signalling session. If the match succeeds, then the RESPONSE MUST NOT be forwarded further along the path. If the QNE did not insert this RII itself, if must forward the RESPONSE to the next peer. Thus, forwarding should only stop if the QNE inserted the RII by itself.

On receipt of a RESPONSE message containing an RSN object, the QNE MUST compare the RSN to that of the appropriate signalling session. If the match succeeds then the INFO_SPEC MUST be processed. The RESPONSE message MUST NOT be forwarded further along the path whether or not the match succeeds. 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 upstrem nodes, which means that the upstream nodes may release a
part of the resource reservation.

5.4.4. NOTIFY Messages

NOTIFY messages are used to convey information to a QNE asynchronously. The format of a NOTIFY message is as follows:

```
NOTIFY = COMMON_HEADER
      PACKET_CLASSIFIER INFO_SPEC *2QSPEC
```

NOTIFY messages are impotent. They do not cause any state to be installed. However, if the notification indicates an error, the indicated state may be removed. The exact operation depends on the QoS model. NOTIFY message do do 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).

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.

The QoS NSLP Message Type is a 16 bit value. The allocation of values for new message types requires standards action. This specification defines four QoS NSLP message types, which form the initial contents of this registry: RESERVE, QUERY, RESPONSE and NOTIFY.

QoS NSLP Messages have a messages-specific 16 bit flags/reserved field in their header. The allocation policy for new flags is TBD.

The QoS Model Identifier (QoS Model ID) is carried in a QSPEC object. The allocation policy for new QoS Model IDs is TBD.

This specification defines a NSLP for use with GIST. Consequently, a

new identifier must be assigned for it from GIST NSLP Identifier registry.

This document also defines six new objects for the QoS NSLP: RII, RSN, REFRESH_PERIOD, BOUND_SESSION_ID, PACKET_CLASSIFIER, QSPEC and INFO_SPEC. Values are to be assigned for them from NSLP Object Type registry.

In addition it defines a number of Error Codes for the QoS NSLP. These can be found in Section 5.1.3.6 and are to be assigned values from NSLP Error Code registry.

Further consideration of IANA issues can be found in a separate draft [I-D.loughney-nsis-ext].

7. QoS use of GIST service interface

This section describes the use of GIST service interface to implement QoS NSLP requirements on GIST.

7.1. Example sender-initiated reservation

We first describe the use of the service interface in a very basic scenario: message reception and transmission for a RESERVE message in a sender-initiated reservation.

A QNE that wishes to initiate a sender-initiated reservation constructs a new RESERVE message to send downstream. The use of GIST service interface in this case is explained on Figure 35. Note that we assume the SII handling in GIST [I-D.ietf-nsis-ntlp] is extended to distinguish between own and peer SII.

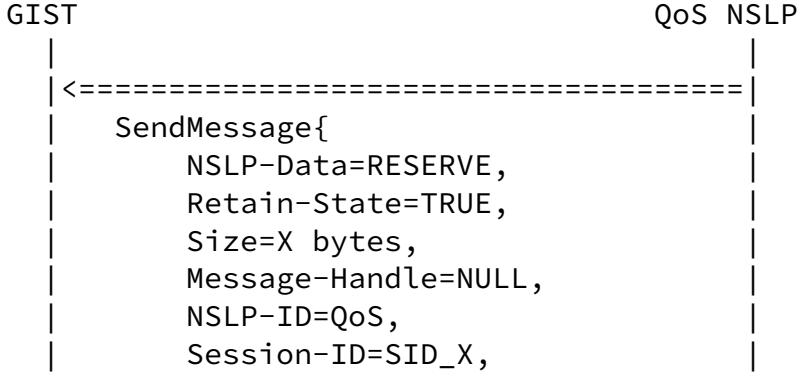


Figure 35: GIST service interface usage for

sending a sender-initiated reservation

Note that an explicit preference for a particular type of transport, such as reliable/unreliable, may change the values of some service interface parameters (e.g. Transfer-attributes=unreliable).

The message is received by the peer QNE. The use of GIST service interface when receiving a RESERVE message for a sender-initiated reservation is explained on Figure 36.

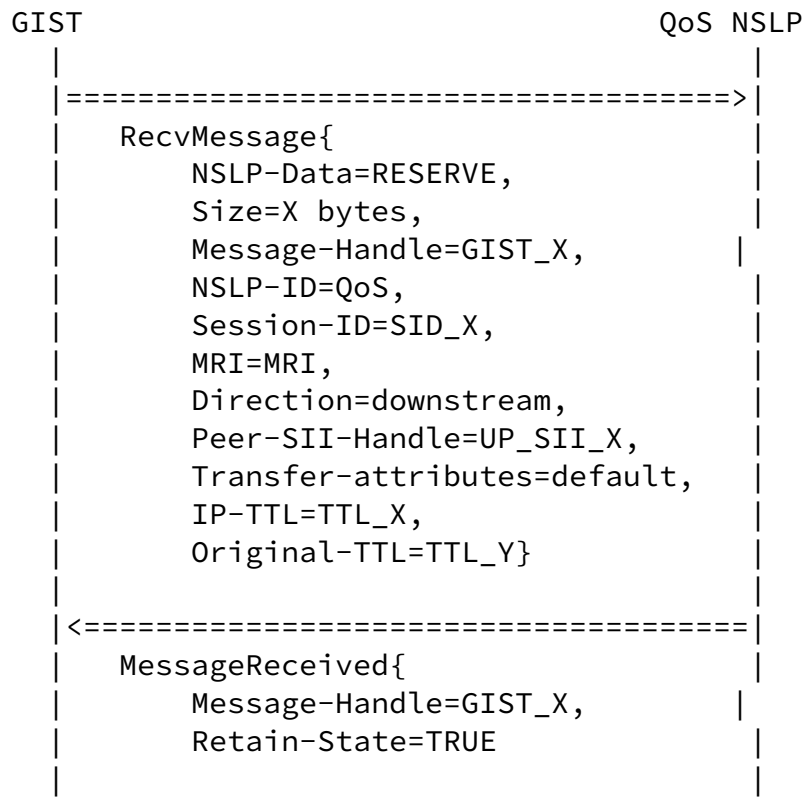


Figure 36: GIST service interface usage for message reception of sender-initiated reservation

7.2. Session identification

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. On Figure 35, QoS NSLP picks a value SID_X for Session-ID. This value is subsequently used by GIST and QoS NSLP to refer to this session.

7.3. 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 3.2.8), when only the edge QNEs of a domain process the message. This requires a mechanism at GIMPS level (which can be invoked by the QoS NSLP) to bypass intermediates 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 MUST be accomplished by the ingress edge by modifying the QoS-NSLP default NSLP-ID value to a NSLP-ID predefined value. The egress MUST stop 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.

7.4. 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 signalling 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.

7.5. 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.

7.6. Last node detection

There are situations in which a QNE needs to determine whether it is the last QNE on the data path (QNR), e.g. to construct and send a

RESPONSE message.

A number of conditions may result in a QNE determining that it is the QNR:

- the QNE may be the flow destination
- the QNE have some other prior knowledge that it should act as the QNR
- the QNE may be the last NSIS-capable node on the path
- the QNE may be the last NSIS-capable node on the path supporting the QoS NSLP

Of these four conditions, the last two can only be detected by GIST. We rely on GIST to inform the QoS NSLP about these cases by providing a trigger to the QoS NSLP when it determines that it is the last NE on the path, which supports the QoS NSLP. GIST supports this by the MessageDeliverError primitive. The error type 'no next node found' which is given as an example can be used. It is expected that additional error codes need to be defined.

7.7. Re-routing detection

Route changes may be detected at GIST layer or the information may be obtained by GIST through local interaction with or notification from routing protocols or modules. GIST allows to pass such information over the service interface using the NetworkNotification primitive with the appropriate 'downstream route change' or 'upstream route change' notification.

7.8. Priority of signalling 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 3.2.9) or from the responsiveness expected by certain applications supported by the QoS NSLP.

GIST design should be able to ensure that performance for one class of messages was not degraded by aggregation with other classes of messages. GIST service interface supports this with the 'priority' transfer attribute.

7.9. Knowledge of intermediate QoS NSLP unaware nodes

In some cases it is useful to know that a reservation has not been installed at every router along the path. It is not possible to determine this using only NSLP functionality.

GIST should be able to provide information to the NSLP about whether the message has passed through nodes that did not provide support for this NSLP.

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.

The QSPEC template also includes a bit "<NON QOSM Hop>" telling that one or more QOSM-aware QNE were encountered on the path from the QNI to the QNR [I-D.ietf-nsis-qspec].

7.10. NSLP Data Size

When GIST passes the QoS NSLP data to the NSLP for processing, it must also indicate the size of that data. This is supported by the NSLP-Data-Size attribute.

7.11. Notification of GIST 'D' flag value

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. This is done in the Direction attribute of the SendMessage and RecvMessage primitives.

7.12. NAT Traversal

The QoS NSLP relies on GIST for NAT traversal.

8. Assumptions on the QoS Model

8.1. Resource sharing

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.

The Resource Management Function (RMF) reserves resources for each flow. We assume that the QoS model supports resource sharing between flows. A QoS Model may elect to implement a more general behaviour 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.

8.2. Reserve/commit support

Reserve/commit behaviour means that the time at which the reservation is made may be different from the time when the reserved resources are actually set aside for the requesting session. This specification acknowledges the usefulness of such a mechanism but assumes that its implementation is opaque to QoS NSLP and is fully handled by the QoS Model.

9. Security Considerations

9.1. Introduction and Threat Overview

The security requirement for the QoS NSLP is to protect the signalling exchange for establishing QoS reservations against identified security threats. For the signalling 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 summarised 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 model 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:

Signalling 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 which includes end-to-middle, middle-to-middle and

even end-to-end protection.

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.

In addition to the above-raised issues we see the following functionality provided at the NSLP layer:

Prevention of Denial of Service Attacks:

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

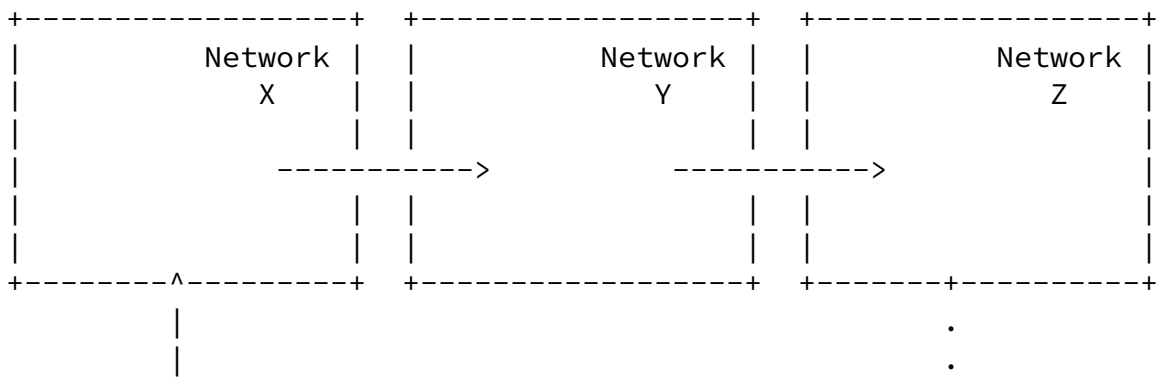
To some extent the QoS NSLP relies on the security mechanisms provided by GIST which by itself relies on existing authentication and key exchange protocols. Some signalling messages cannot be protected by GIST and hence should be used with care by the QoS NSLP. An API must ensure that the QoS NSLP implementation 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.

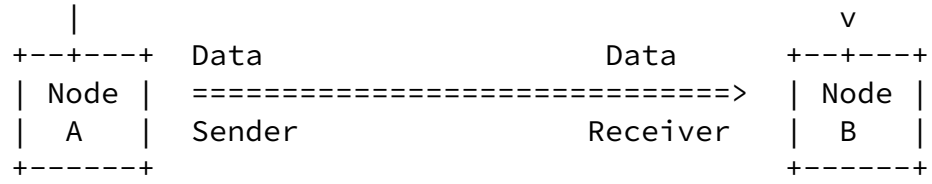
9.2. Trust Model

For this version of the document we will rely on a model which requires trust between neighboring NSLP nodes to establish a chain-of-trust along the QoS signalling path. This 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' model.

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 signalling 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.





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 37: New Jersey Turnpike Model

The model shown in Figure 37 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 This model allows authorization on a request basis or on a per-session basis. Authorization mechanisms will be 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. Price distribution protocols are not covered in this version of the document. This model assumes, per default, that the data sender is authorizing the QoS reservation. Please note that this is only a simplification and further extensions are possible and left for a future version of this document.

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

Charging the data sender as performed in this 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 signalling messages. The identity used during the authentication and key establishment phase would be used by Network X (see Figure 37) 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. Signalling 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-pricing] 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.

9.3. 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.

A detailed description of the authorization handling will be left for a future version of this document. The authors assume that the QoS NSLP needs to provide a number of attributes to support the large range of scenarios.

10. Open Issues

Some of the areas for further work in this draft are indicated with [FIXME] markers.

In addition, a list of open issues is contained in an online issue tracker at http://nsis.srmr.co.uk/cgi-bin/roundup.cgi/nsis-qos-nslp-issues

11. Acknowledgements

The authors would like to thank Eleanor Hepworth, Ruediger Geib, Roland Bless and Nemeth Krisztian for their useful comments.

Bob Braden provided helpful comments and guidance which were gratefully received.

12. 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).

Yacine El Mghazli (Alcatel) contributed text on AAA.

13. References

13.1. Normative References

[I-D.ietf-nsis-ntlp] Schulzrinne, H., and R. Hancock, "GIST: General Internet Messaging Protocol for Signaling", draft-ietf-nsis-ntlp-08 (work in progress), September 2005.

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

[RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

13.2. Informative References

[RFC4080] Hancock, R., "Next Steps in Signaling: Framework", RFC 4080, December 2004.

[RFC4081] Tschofenig, H. and D. Kroeselberg, "Security Threats for NSIS", RFC 4081, October 2004.

[I-D.ash-nsis-y1541-qosm] Ash, J., "Y.1541 QoS Model for Networks Using Y.1541 QoS Classes", draft-ietf-nsis-y1541-qosm-00 (work in progress), August 2005.

[I-D.ietf-nsis-qspec] Ash, J., "QoS NSLP QSPEC Template", draft-ietf-nsis-qspec-06 (work in progress), October 2005.

[I-D.ietf-nsis-rmd] Bader, A., "RMD-QOSM - The Resource Management in Diffserv QoS model", draft-ietf-nsis-rmd-03 (work in progress), June 2005.

[I-D.kappler-nsis-qosmodel-controlledload] Kappler, C., "A QoS Model for Signaling IntServ Controlled-Load Service with NSIS", draft-kappler-nsis-qosmodel-controlledload-01 (work in progress), May 2005.

[I-D.loughney-nsis-ext] Loughney, J. "NSIS Extensibility Model", draft-loughney-nsis-ext-00 (work in progress), May 2005.

[I-D.manner-lrsvp] Manner, J., "Localized RSVP", draft-manner-lrsvp-04 (work in progress), September 2004.

[I-D.tschofenig-nsis-aaa-issues] Tschofenig, H., "NSIS Authentication, Authorization and Accounting Issues", draft-tschofenig-nsis-aaa-issues-01 (work in progress), March 2003.

[I-D.tschofenig-nsis-qos-authz-issues] Tschofenig, H., "QoS NSLP Authorization Issues", draft-tschofenig-nsis-qos-authz-issues-00 (work in progress), June 2003.

[MEF.EthernetServicesModel] Metro Ethernet Forum, "Ethernet Services Model", letter ballot document , August 2003.

[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.

[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.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 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.

[RFC3583] Chaskar, H., "Requirements of a Quality of Service (QoS) Solution for Mobile IP", RFC 3583, September 2003.

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

[_XREF_FJ94] Jacobson, V., "Synchronization of Periodic Routing Messages", IEEE/ACM Transactions on Networking , Vol. 2 , No. 2 , April 1994.

[_XREF_OPWA95] Breslau, L., "Two Issues in Reservation Establishment", Proc. ACM SIGCOMM '95 , Cambridge , MA , August 1995.

[shenker-pricing] Shenker, S., Clark, D., Estrin, D., and S. Herzog, "Pricing in computer networks: Reshaping the research agenda", Proc. of TPRC 1995, 1995.

Authors' Addresses

Jukka Manner
Department of Computer Science University of Helsinki
P.O. Box 26 (Teollisuuskatu 23)
HELSINKI, FIN-00014
Finland
Phone: +358-9-191-44210
EMail: jmanner@cs.helsinki.fi

Sven Van den Bosch
Alcatel
Francis Wellesplein 1
Antwerpen B-2018
Belgium
Email: sven.van_den_bosch@alcatel.be

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 SO51 0ZN
UK
Email: andrew.mcdonald@roke.co.uk

Appendix A. 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

SII: Source Identification Information

SIP: Session Initiation Protocol

RII: Request Identification Information

RMD: Resource Management for DiffServ

RMF: Resource Management Function

RSN: Reservation Sequence Number

RSVP: Resource reSerVation Protocol (see [RFC2205])

Appendix B. Change History

Note to RFC Editor: This section is to be removed before publication.

Changes from -00

* Additional explanation of RSN versus Session ID differences. (Session IDs still need to be present and aren't replaced by RSNs. Explain how QoS NSLP could react once it notes that it maintains stale state.)

* Additional explanation of message types - why we don't just have RESERVE and RESPONSE.

* Clarified that figure 1 is not an implementation restriction.

Changes from -01

* Significant restructuring.

* Added more concrete details of message formats and processing.

* Added description of layering/aggregation concepts.

* Added details of authentication/authorisation aspects.

Changes from -02

* Addressed comments from early review.

* Added text on receiver-initiated and bi-directional reservations.

* Extended description of session binding. Added support for fate sharing.

* Restructured message formats and processing section.

* Clarified refresh reduction mechanism.

* Added assumptions on QSM.

* Added assumptions on operating environment.

Changes from -03

* Removed overlaps between sections.

* Clarified document does not specify how to aggregate individual end-to-end flow from a resource point of view but rather how such an aggregate can be signalled for.

* Made session binding purely informational.

* Clarified QSPEC stacking.

* Added object format for ERROR_SPEC object.

* Made RII a separate object from RESPONSE_REQUEST and outside of the SCOPING object. Then removed RESPONSE_REQUEST and made SCOPING a flag rather than an object.

* Closed open issue of "PATH" message functionality. An empty QUERY is used to install reverse state along the path.

* Made all flag names positive. Removed NO_FATE_SHARING flag: fate sharing is not supported by the signalling.

* Removed the open issue on one-sided bidirectional reservation. Clarified how it can be done, even for stateless or reduced state domains in an example.

* Removed open issue on priority. Message priority will be handled over GIST API, reservation priority is an issue for the RMF.

Changes from -04

* Resolved a number of outstanding comments on clarifications (likelihood of transport type, bidirectional reservations, handle of RESERVE messages inside a domain in case of aggregation or reduced state operation) from the mailing list.

* Introduced a default value for REFRESH_PERIOD.

* Introduced explicit feedback mechanism in case of route changes.

* State acknowledgment is now supported by means of an ACKNOWLEDGE flag. This is made the default case.

* Changed section 7 to reflect the use of GIST service interface.

Changes from -05

* Modified definitions of QoS Model and NSLP/QSPEC relationships. Removed concepts of QoS Signalling Model (QSM) and QoS Signalling Policy (QSP).

* Made changes to the policy control and authentication concepts. Removed old appendix on original POLICY_CONTROL object.

* Added a glossary.

* Added text on reservation collision handling.

* Moved this list of changes to the last appendix to make it easier to remove at publication time.

Changes from -06

* Change of editorship, (Sven -> Jukka) and, as a consequence, change from XML editing to good old nroff. ;)

* Renamed "Summary refresh" to "Reduced refresh", as the old seemed to be a somewhat unclear term, and the new follows the terminology of RFC2961.

* Added some missing figure captions, and an introductory text to Fig. 2

* Added some clarifications to Sections 3.2.2, 3.2.8.1, 4.8, 5.1, 5.2.3, and 5.4.1

* Removed some texts that makes requests about GIST. These should be handled e.g. through the open issues list, and not have these

types of clearly open issues within the main text.

* Added "[FIXME: ...]" entries in various place to be handled at some point.

* Added discussion on using RII as a "Forced Refresh" mechanism that is needed to support changes in routing paths. Now any node that gets to know that a routing change has happened somewhere can force a repair of the installed reservation state.

* Various editorial changes, and typo fixes, e.g., search-replace "QoS-NSLP" > "QoS NSLP" and "QSpec" > "QSPEC"

* Updated sections on "QoS model stacking".

* Added some additional notes on policy control interactions.

* Added initial version of a packet classifier information object.

Changes from -07

* fixed text talking about the GIST API on message processing priorities

* clarified a number of issues, especially on message processing, eg., RESPONSE message processing, and the use of RII

* reformatted the COMMON_HEADER

* reformatted and renamed the ERROR_SPEC to INFO_SPEC

* Added a new flag to the QUERY message to indicate a receiver-initiated reservation is requested

* Updated text and the specification of RSNs

* Added text about retransmissions

* Added text about default message processing when receicing an erroneous message

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Acknowledgment

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