Next Steps in Signaling
Internet-Draft
Expires: November 9, 2004

S. Van den Bosch
Alcatel
G. Karagiannis
University of Twente/Ericsson
A. McDonald
Siemens/Roke Manor Research

May 11, 2004

NSLP for Quality-of-Service signaling
draft-ietf-nsis-qos-nslp-03.txt

Status of this Memo

This Internet-Draft will expire on November 9, 2004.

Copyright Notice

Abstract

This draft describes an NSIS Signaling Layer Protocol (NSLP) for signaling QoS reservations in the Internet. It is in accordance with the framework and requirements developed in NSIS.

Together with GIMPS, 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 version of the draft focuses on the basic protocol structure. It identifies the different message types and describes the basic operation of the protocol to create, refresh, modify and teardown a reservation or to obtain information on the characteristics of the associated data path.

Conventions used in this document

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.

Table of Contents

1. Introduction

1.1 Scope and background

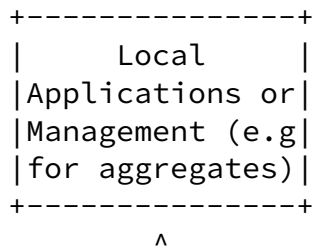
This document defines a Quality of Service (QoS) NSIS Signaling Layer Protocol (NSLP), henceforth referred to as the "QoS-NSLP". This protocol establishes and maintains state at nodes along the path of a data flow for the purpose of providing some forwarding resources for that flow. It is intended to satisfy the QoS-related requirements of [14]. This QoS-NSLP is part of a larger suite of signaling protocols, whose structure is outlined in [15]; this defines a common NSIS Transport Layer Protocol (NTLP) which QoS-NSLP uses to carry out many aspects of signaling message delivery. The specification of the NTLP is done in another document [3].

The design of QoS-NSLP is conceptually similar to RSVP [5], 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 signaling application gateway would be possible in some circumstances. QoS-NSLP extends the set of reservation mechanisms to meet the requirements of [14], 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 support for IP multicast.

QoS-NSLP does not mandate any specific 'QoS Model', i.e. a particular QoS provisioning method or QoS architecture; this is similar to (but stronger than) the decoupling between RSVP and the IntServ architecture [4]. It should be able to carry information for various QoS models; the specification of Integrated Services for use with RSVP given in [6] could form the basis of one QoS model.

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. It is adapted from the discussion in section 1 of [5]. The model is shown in Figure 1.



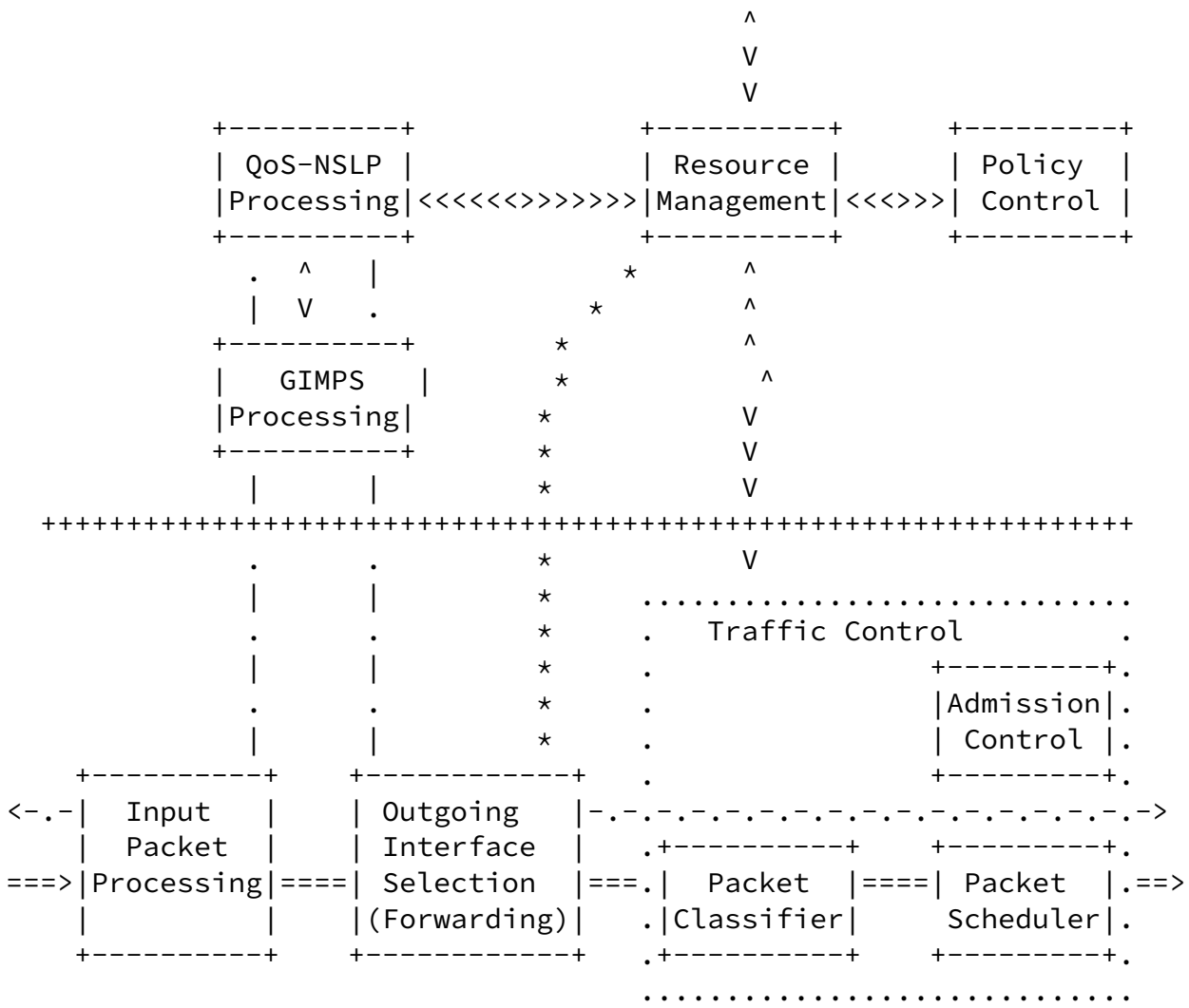


Figure 1: QoS-NSLP in a Node

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.

From the perspective of a single node, the request for QoS may result from a local application request, or from processing an incoming QoS-NSLP message.

- The 'local application case' 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.
- The 'incoming message' case requires NSIS messages to be captured during input packet processing and handled by GIMPS. Only messages related to QoS are passed to the QoS-NSLP. GIMPS 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.

- 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.
- 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 [16] and [17]. 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 4.8. The definition of the POLICY_DATA class is given in Appendix A.7.

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; we refer to this as a QoS model. Our assumption is that the QoS-NSLP is independent of the QoS model, that is, QoS parameters (e.g. IntServ service elements) are 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.

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 propagate the resource request forwards in the other. Message routing is (by default) carried out by GIMPS module. Note that while Figure 1 shows a unidirectional data flow, the signaling messages can pass in both directions through the node, depending on the particular message and orientation of the reservation.

2. Terminology

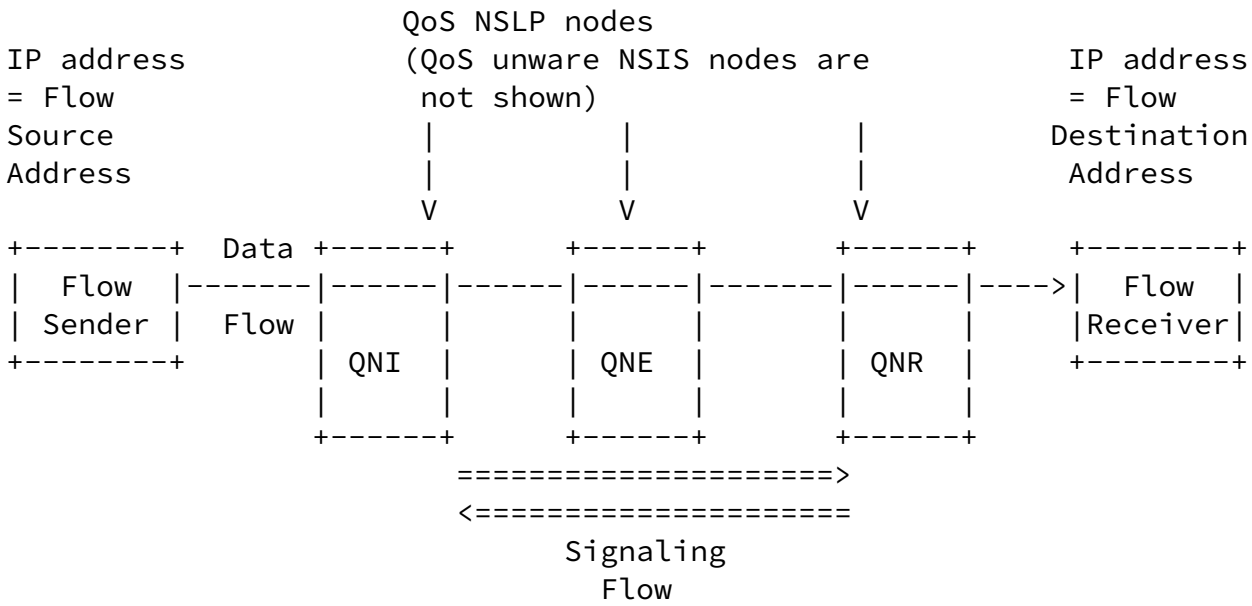
The terminology defined in [3] applies to this draft. In addition, the following terms are used:

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.

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.



3. Protocol Overview

The QoS NSLP uses four message types: RESERVE, QUERY, RESPONSE and NOTIFY. These contain three types of objects: Control Information (CI), QSpecs, and Policy objects. The set of objects permissible depends on the message type.

An interface exists between the NSLP processing and GIMPS processing for sending/receiving NSIS messages. In addition to the NSLP message data itself, other meta-data (e.g. session identifier, flow routing information) can be transferred across this interface.

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

QSpec objects describe the actual resources that are required and are specific to the QoS Model being used. Besides any resource

description they may also contain QoS Model specific control information used by the QoS Model's processing.

The Policy objects contain data used to authorise the reservation of resources.

3.1 QoS Models

A QoS model is a defined mechanism for achieving QoS as a whole. The specification of a QoS model includes a description of its QoS parameter information, as well as how that information should be treated or interpreted in the network. In that sense, the QoS model

goes beyond the QoS-NSLP protocol level in that it could also describe underlying assumptions, conditions and/or specific provisioning mechanisms appropriate for it.

A QoS model provides a specific set of parameters to be carried in the QSpec, or restricts the values these parameters can take. Integrated Services [4], Differentiated Services [8] and RMD [22] are all examples that could provide the basis of an NSIS QoS model. There is no restriction on the number of QoS models. 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: [18], [19] and [20]. This specification will not attempt to select between the moppling number of possible QoS models.

The QSpec contains a set of parameters and values describing the requested resources. It is opaque to the QoS-NSLP and similar in purpose to the TSpec, RSpec and AdSpec specified in [5][6]. At each QNE, its content is interpreted by the resource management function for the purposes of policy control and traffic control (including admission control and configuration of the packet classifier and scheduler).

3.2 GIMPS Interactions

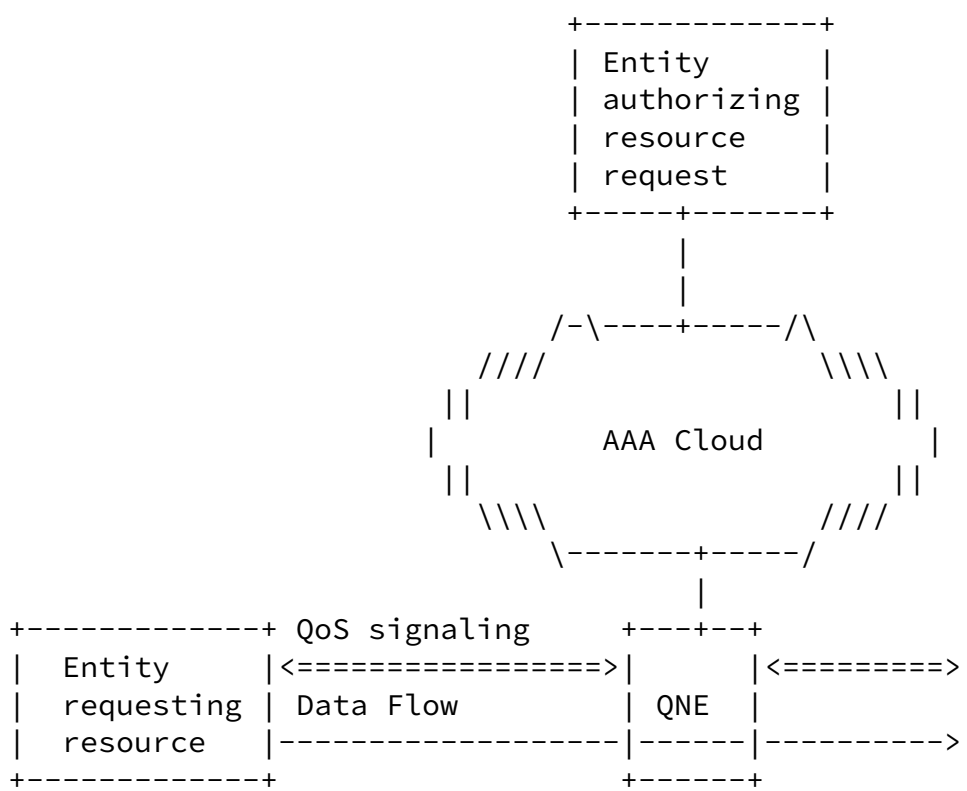
The QoS NSLP uses GIMPS for delivery of all its messages. Messages are normally passed from the NSLP to the GIMPS via an API, which also specifies additional information, including an identifier for the signaling application (e.g. 'QoS-NSLP'), the flow/session

identifier, and an indication of the intended direction - towards data sender or receiver. On reception, GIMPS provides the same information to the QoS-NSLP.

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 the GIMPS.

3.3 Authentication and authorization

The QoS signaling protocol needs to exchange information which is subsequently used as input to the AAA infrastructure. The response from the AAA infrastructure must also returned and processed by the respective entities.



3.4 Aggregation

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 facilities to provide similar aggregation facilities to [10]. However, the aggregation scenarios supported are wider than that proposed there. Aggregate reservations are further described in Section 4.3.3.

3.5 Examples of QoS NSLP Operation

The QoS NSLP can be used in a number 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.

3.5.1 Simple Resource Reservation

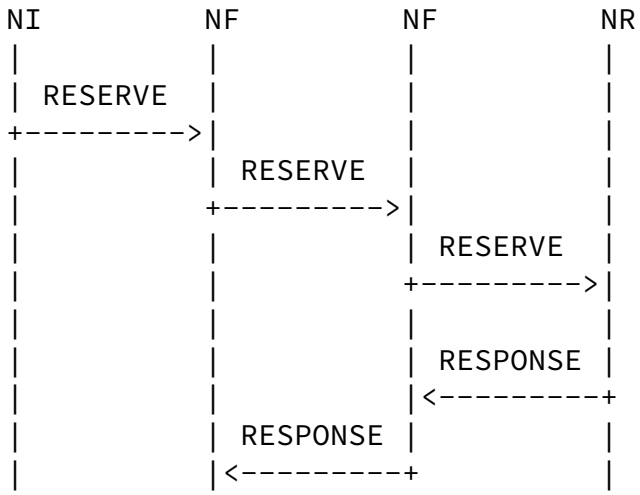


Figure 4: Basic Sender Initiated Reservation

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 GIMPS which transports it to the next QoS NSLP node. 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 the GIMPS, 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 GIMPS functionality to determine that there are no more QNEs between this node and the data flow destination.

Any node may include a request for a RESPONSE in its RESERVE messages. One such use is to confirm the installation of state, which allows the use of summary refreshes that later refer to that state. The RESPONSE is forwarded peer-to-peer along the reverse of the path that the RESERVE message took (using GIMPS 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. A RESPONSE message can also indicate an error when, for example, a reservation has failed to be installed.

The reservation can subsquently 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 QoS model specific 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 [9]. 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.

3.5.2 Sending a Query

QUERY messages can be used to gather information from QNEs along to 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 QoS model specific objects containing the actual query to be performed at QoS NSLP nodes 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 GIMPS 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 GIMPS 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 includes a RESPONSE_REQUEST object. Into this is copied various objects from the received QUERY message. It is then passed to GIMPS to be forwarded peer-to-peer back along the path.

Each QNE receiving the RESPONSE message should inspect the RESPONSE_REQUEST 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 GIMPS to be forwarded back down the path.

3.5.3 Basic Receiver Initiated Reservation

As described in [15] in some signaling applications, a node at one end of the data flow takes responsibility for requesting special treatment - such as a resource reservation - from the network. Both ends then agree whether sender or receiver initiated reservation is to be done. In case of a receiver initiated reservation, both ends agree whether a "One Pass With Advertising" (OPWA) [23] model is being used. This negotiation can be accomplished using mechanisms that are outside the scope of NSIS, see Section 9.7.

To make a receiver initiated reservation, see Figure 5, the QNI constructs a QUERY message containing a QSpec object, from its chosen QoS model, which describes, among others, the required QoS parameters. This QUERY message does not need to trigger a RESPONSE message and therefore, the QNI must not include the RESPONSE_REQUEST object, see Section 5.4.2, into the QUERY message. The QUERY message may be used to 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) [23]. This QUERY message is instructing the QoS-NSLP process running on each QNE's located on the path followed by this QUERY message to install GIMPS reverse-path state.

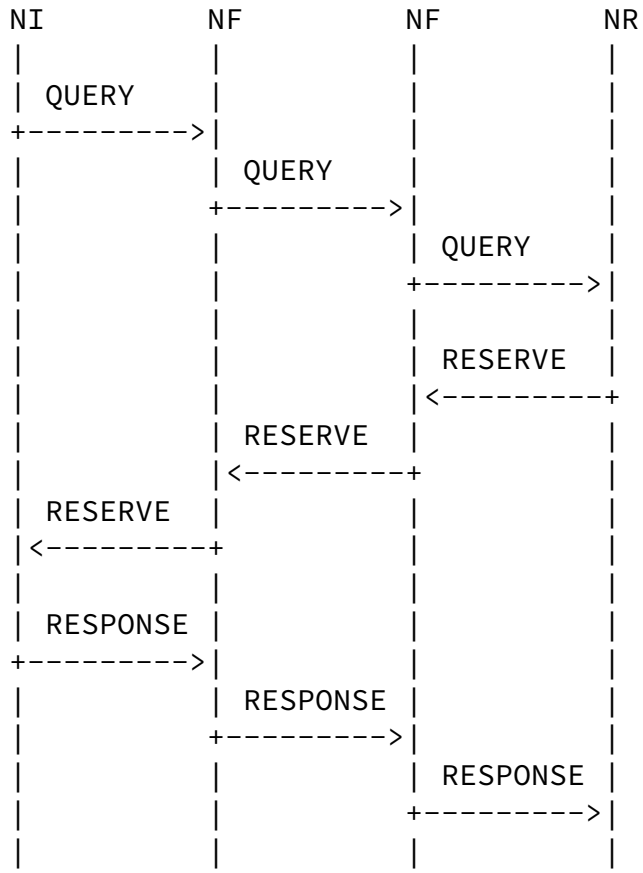


Figure 5: Basic Receiver Initiated Reservation

The QUERY message is transported by the NTLP 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. The node performs appropriate actions, such as installing GIMPS reverse path state. Another action that may be performed by the node is to gather advertising information that may be used to predict the end-to-end QoS.

The QoS NSLP then generates a new QUERY message (usually based on the one received). This is passed to the NTLP, which forwards it to the next QNE. The same processing is performed at further QNEs along the path, up to the receiver, which in this situation is the QNR. The QNR detects that this QUERY message does not carry a RESPONSE_REQUEST object and by using the information contained in the received QUERY message, such as the QSPEC, constructs a RESERVE message, which can be used as a reservation request.

The RESERVE message must follow the same path that has been followed by the QUERY message. Therefore, the NTLP is informed, over the NSLP/NTLP API, to pass the message upstream, i.e., by setting the

GIMPS "D" flag, see [3].

The RESERVE is forwarded peer-to-peer along the reverse of the path that the QUERY message took (using the NTLP reverse path state). It is only forwarded as far as the QNI. The RESERVE message received by a QNE is delivered to the QoS-NSLP processing, which examines the message. The NTLP includes the value of the 'D' flag in the information it passes with the message over the NTLP/NSLP API (Section 7.10). The exact processing of this message is accomplished in the same way as for the processing of a RESERVE message received by a QNE in a sender initiated approach. The main difference is that the RESERVE message has to be forwarded peer-to-peer along the reverse of the path that the QUERY message took. Therefore, the QoS-NSLP functionality of each QNE requires the NTLP, over the NSLP/NTLP API, to pass the message upstream, i.e.,by setting the GIMPS "D" flag.

Similar to the sender initiated approach, any node may include a request for a RESPONSE in its RESERVE messages. The RESPONSE is forwarded peer-to-peer along the path that the initial QUERY message took (using NTLP path state). It is only forwarded as far as the node which requested the RESPONSE. Note that the use of RESPONSE is optional.

The reservation can subsquently be refreshed in the same was as for the refresh procedure of a reservation in a sender initiated approach, by sending further RESERVE messages containing the complete reservation information, as for the initial reservation. This RESERVE message may be also used to refresh the NTLP reverse path state. Additionally, refreshing the NTLP reverse path state could be performed by sending periodic QUERY messages. However, it might potentially also be done by the NTLP, without needing any downstream NSLP messages. This is an open issue (see Section 9.6), and it will be worked out in a future version of this draft.

3.5.4 Bidirectional Reservations

A bi-directional reservation combines the reservations for a pair of coupled flows going in opposite directions. The main difficulty here is that the two flows, although between the same end points, may traverse different paths across the Internet.

We distinguish two types of bi-directional reservations:

- sender+sender: where a sender-initiated reservation is done in one direction, e.g., from QNE A towards QNE B, and then another sender-initiated reservation that is done back, the opposite direction, i.e., from QNE B towards QNE A,
- sender+receiver: where a sender-initiated reservation is done in one direction, e.g., from QNE A towards QNE B, and a receiver-initiated reservation that is done for the opposite

direction, i.e., a reservation from QNE A towards QNE B for the data flow from QNE B to QNE A.

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, see Section 9.7.

In the sender+sender bi-directional reservation scenario (see Figure 6) the QNI, i.e., QNE A, generates a sender initiated reservation by creating and sending a RESERVE message towards the QNR, i.e., QNE B.

When performing this type of bi-directional reservation, there are situations where the QNR, i.e., QNE B, is not able to retrieve the QoS parameter set required to perform a reservation in the downstream direction from QNE B to QNE A, while the QNI, i.e., QNE A, is able to retrieve and maintain the Qos parameter sets required to perform the reservation in both downstream directions. In these situations there is a need to carry both QoS parameter sets in QoS-NSLP messages from the QNI, i.e., QNE A, to the QNR, i.e., QNE B.

The RESERVE is passed to the NTLP, which forwards it to the next QNE. The same processing is performed at further QNEs along the path, up to the receiver, which in this situation is the QNR, i.e., QNE B. Considering that the QNE B, knows that it has to start a sender-initiated reservation, the QNE B creates and sends a RESERVE message downstream towards the QNE A. This RESERVE message carries among others the QoS parameter set required to perform the sender initiated reservation in the downstream direction from QNE B to QNE A. Furthermore, the QNI, (i.e., QNE A), and QNR, (i.e., QNE B), might be willing to bind the two sessions. In this situation the QoS-NSLP messages that are used in the direction from QNE B to QNE A, may carry information that identifies the two bound sessions. This

can be accomplished by using the BOUND_SESSION_ID object.

Note that any node may generate RESPONSE messages as answer to the received RESERVE messages, but this is optional.

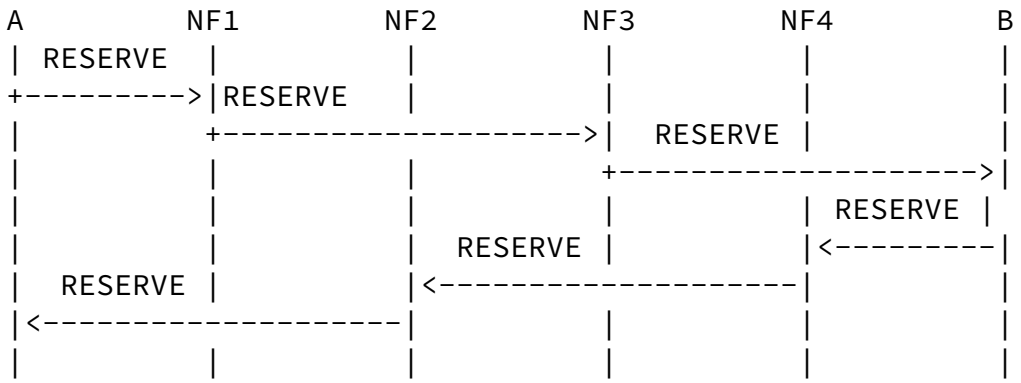


Figure 6: Bi-directional reservation for sender+sender scenario

In the sender+receiver bi-directional reservation scenario (see Figure 7) the QNI, i.e., QNE A, generates both a sender initiated and a receiver-initiated reservation. Note that before that the QNI, i.e., QNE A, is starting this procedure it must receive a QUERY message from the QNR, i.e., QNE B, that requires from the QNI a receiver initiated reservation, see Section 3.5.3. Note that the QUERY message is carrying the QSPEC that can be used by the receiver initiated reservation.

The QNI, i.e., QNE A, creates two RESERVE messages. One RESERVE message that is used for the sender initiated reservation (session A->B), see Section [(reference to the sender initiated section)] and

another RESERVE message that is used for the receiver initiated reservation (session B->A), see Section 3.5.3. Furthermore, the QNI, (i.e., QNE A), and QNR, (i.e., QNE B), might be willing to bind the two sessions. In this situation the QoS-NSLP messages that are used in the bi-directional scenario may carry information that identifies the two bound sessions. This can be accomplished by using the BOUND_SESSION_ID object. When these two RESERVE messages follow exactly the same path, then it should be possible that these two RESERVE messages are bundled at the NTLP level, by using the GIMPS message bundling feature.

Note that any node may generate RESPONSE messages as answer to the received RESERVE messages, but this is optional.

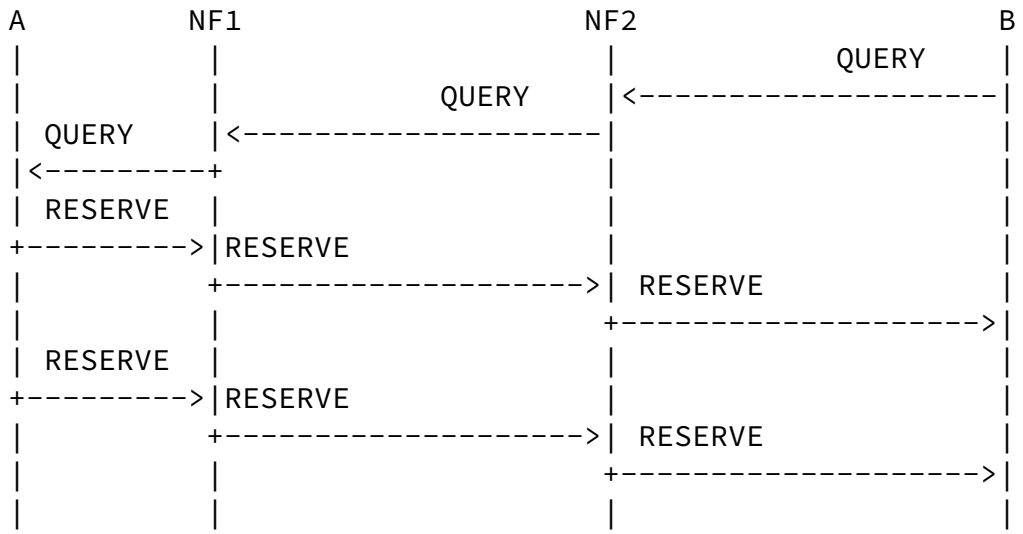


Figure 7: Bi-directional reservation for sender+receiver scenario

3.5.5 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 map between the two resource descriptions (and any auxiliary data).

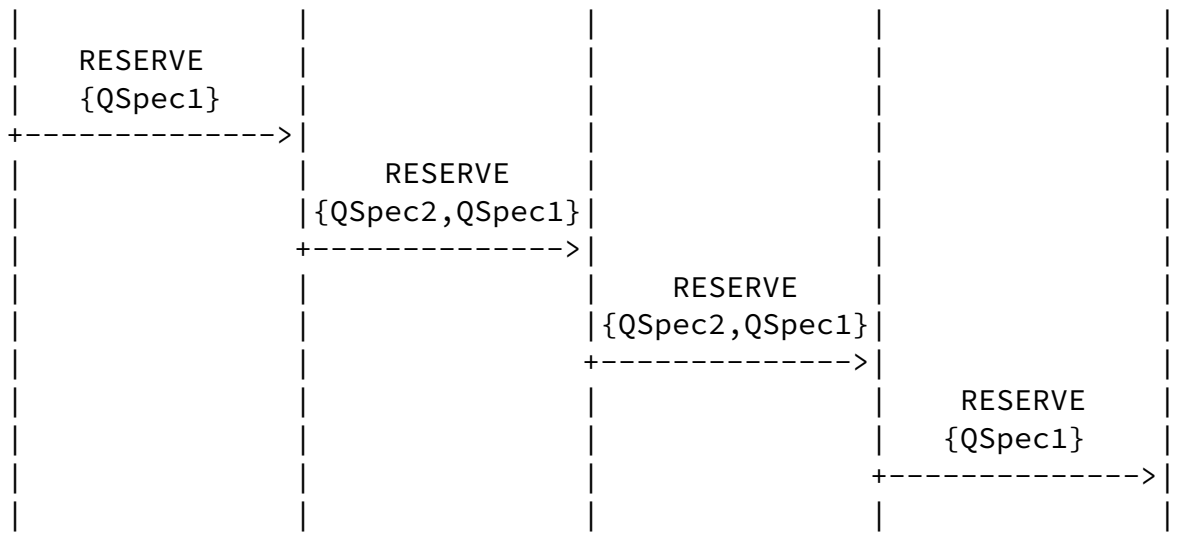


Figure 8: Reservation with local QoS Models

This initially proceeds as for the basic example, with peer-to-peer installation of reservations. However, within a region of the network a different QoS Model 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 the second QoS model. The RESERVE message to be sent out is constructed mostly as usual but with a second QSpec object added, which becomes the 'current' one.

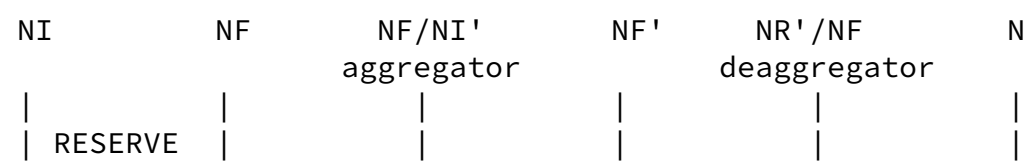
When this RESERVE message is received at the next node the QoS NSLP only uses the QSpec at the top of the stack (i.e. the 'current' one), rather than the end-to-end QSpec. Otherwise, processing proceeds as usual. The RESERVE message that it generates should include the complete stack of QSpecs from the 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.

QSpecs can be stacked in this way to an arbitrary depth.

3.5.6 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.



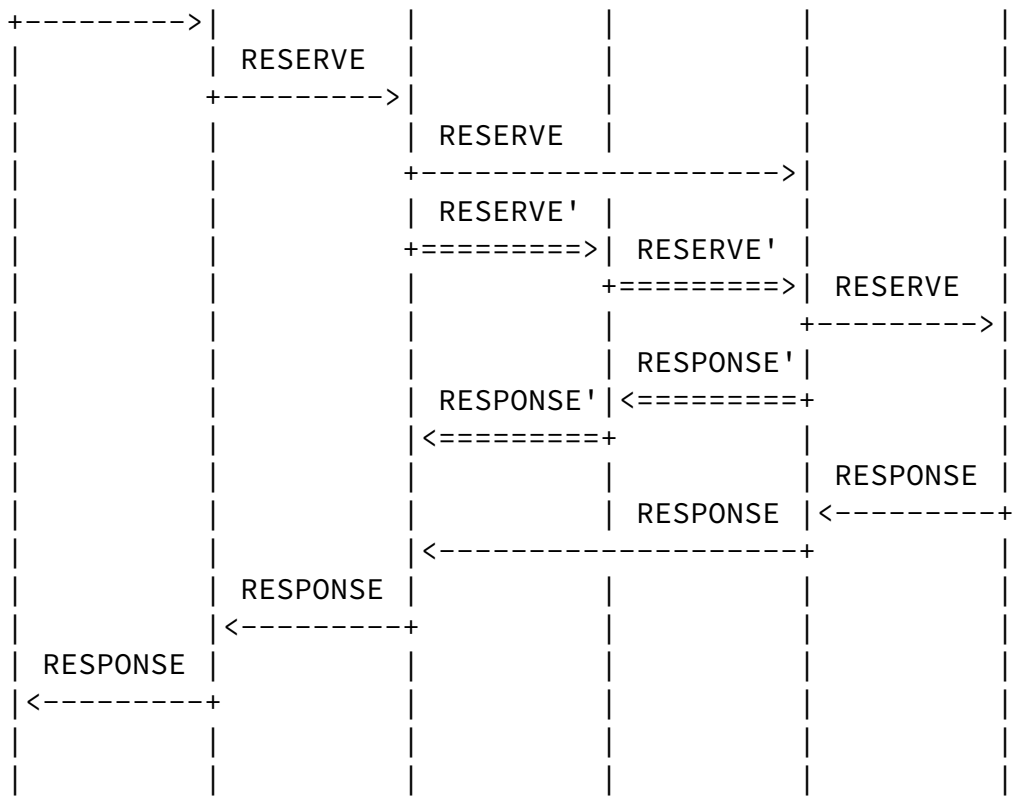
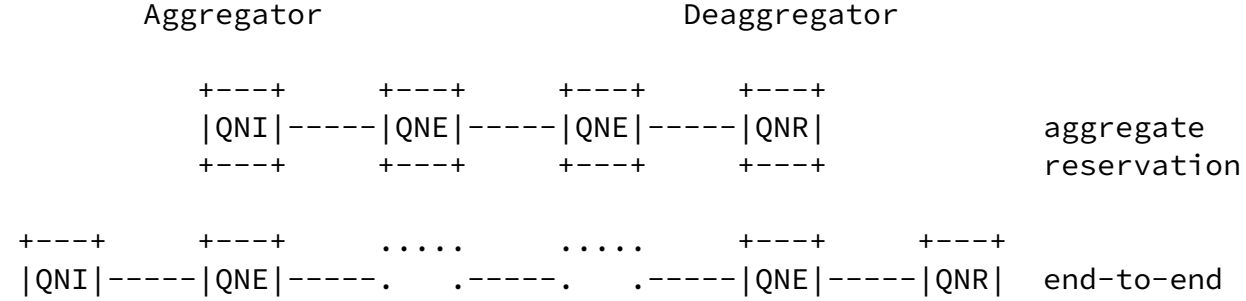


Figure 9: Sender Initiated Reservation with Aggregation

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.

Markings are used so that intermediate routers do not need to inspect the individual flow reservations. The deaggregator then becomes the next hop QoS NSLP node 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. For example, this is necessary so that the size of the aggregate reservation can be reduced when the end-to-end reservation is removed.

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.

3.5.7 Reduced State or Stateless Interior Nodes

This example uses a different QoS model within a domain, in conjunction with GIMPS and NSLP functionality which allows the interior nodes to avoid storing GIMPS and QoS NSLP state. As a result the interior nodes only store the QoS model specific 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 reservation state is needed (or created).

The key difference between this example and the use of different QoS Models in Section 3.5.5 is that the transport characteristics for the 'local' reservation can be different from that of the end-to-end reservation, i.e. GIMPS 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.

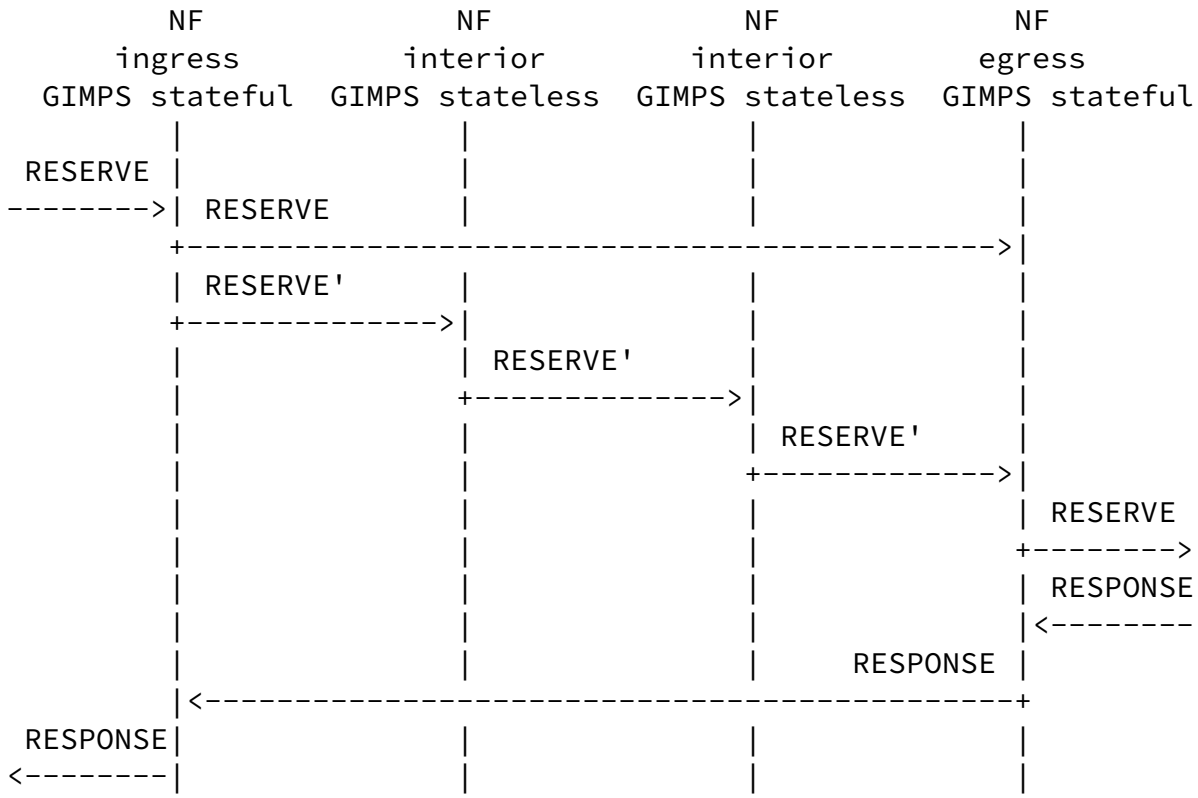


Figure 11: Reservation with Reduced State Interior Nodes

The QNI performs the same processing as before to generate the initial RESERVE message, and it is forwarded by GIMPS 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. The egress node is the next QoS NSLP hop for that session. After the initial discovery phase using datagram mode, connection 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). When processed by interior (stateless) nodes the QoS NSLP processing excercises 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 QoS model related data may be held at these interior nodes. The QoS NSLP also requests that GIMPS use different transport characteristics (i.e. sending of messages in datagram mode, and not retaining optional reverse path 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 summary 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.

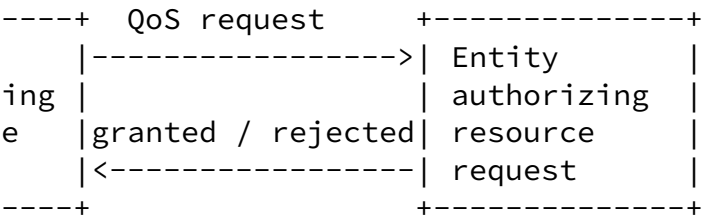
Since GIMPS neighbour relations are not maintained in the reduced-state region, only sender initiated signalling can be supported. If a bi-directional reservation is required then the end-to-end QoS model must provide an object that requests the last node to generate a sender initiated session in the reverse direction.

3.6 Authorization Model Examples

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

3.6.1 Authorization for the two party approach

The two party approach is conceptually the simplest authorization model.



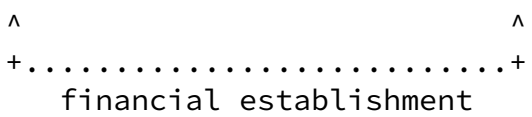


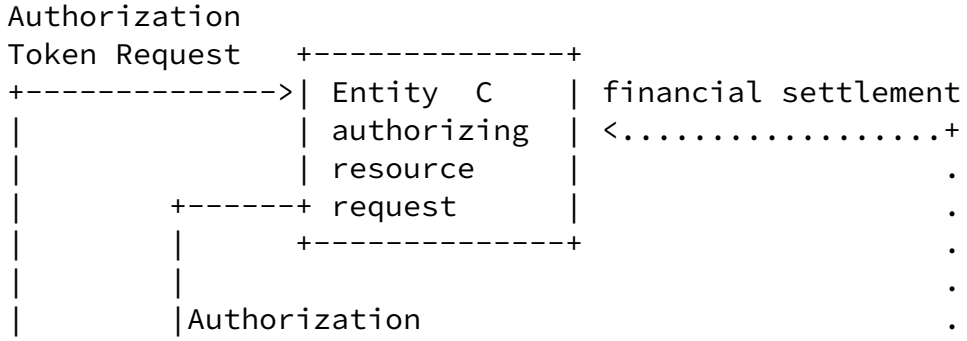
Figure 12: Two party approach

In this example the 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, for example, between two neighboring networks (inter-domain signaling) where a long-term contract (or other out-of-band mechanisms) exist to manage charging and provide sufficient information to authorize individual requests.

3.6.2 Token based three party approach

An alternative approach makes use of authorization tokens, such as those described in [11] and [12] or used as part of the Open Settlement protocol [27]. The former ('authorization tokens') are used to associate two different signaling 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 13. If the request is authorized, then the SIP signaling returns an authorization token which can be included in the QoS signaling protocol messages to refer to the previous authorization decision. The tokens themselves may take a number of different forms, some of which may require the entity performing the QoS reservation to query external state.



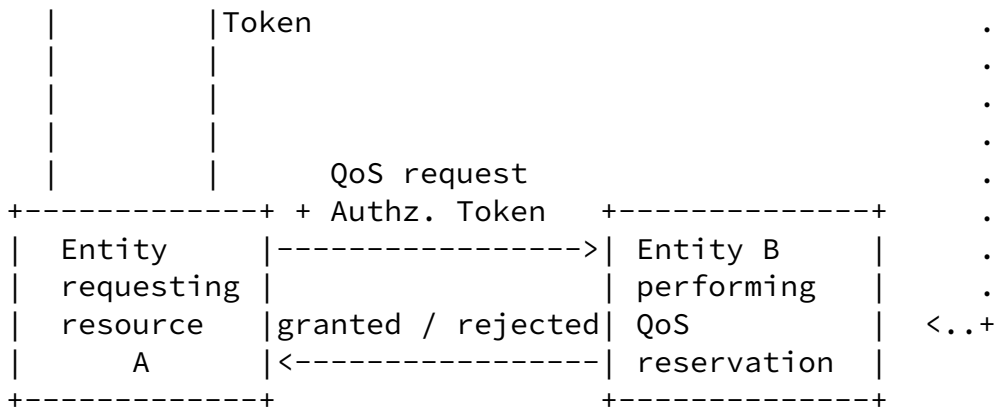


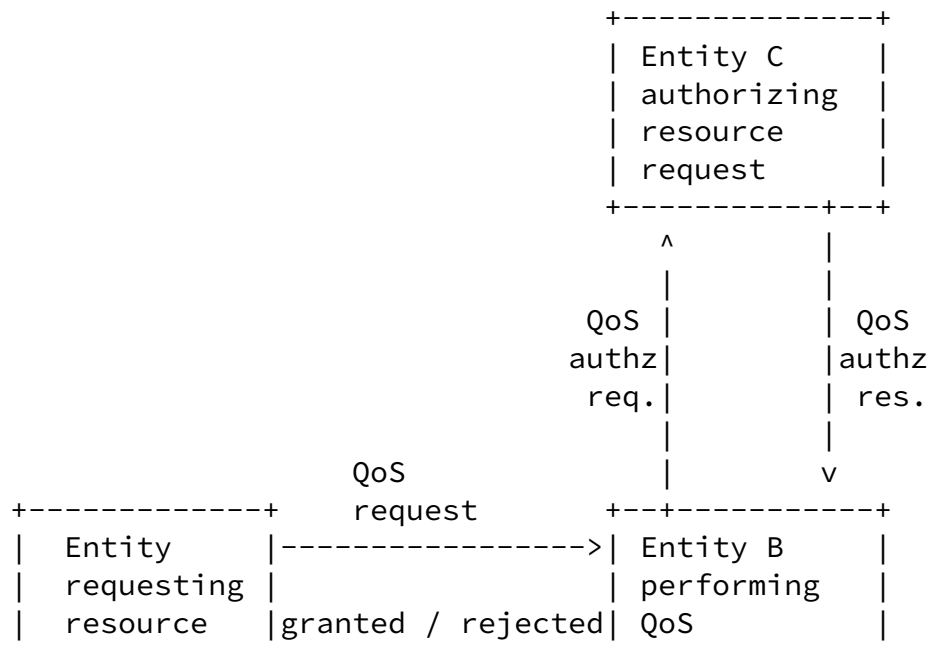
Figure 13: Token based three party approach

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.

3.6.3 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 14.



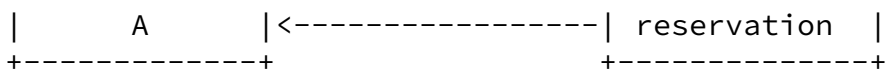


Figure 14: Three party approach

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.

4. Design decisions

4.1 Message types

The QoS-NSLP specifies four message types: RESERVE, QUERY, RESPONSE and NOTIFY.

The fundamental properties of each message determine how it is analyzed from the perspective of re-ordering, loss, end-to-end reliability and so on. It is important for applications to know whether NSLP messages can be repeated, discarded or merged and so on (e.g. for edge mobility support, rerouting, etc). Indeed, the ordering of messages that do not manipulate state at QNEs does not

matter, whereas the way that messages that manipulate state are interleaved matters very much. Therefore NSLP is designed such that the message type identifies whether a message is manipulating state (in which case it is idempotent) or not (it is impotent).

4.1.1 RESERVE

The RESERVE message is the only message that manipulates QoS reservation state. It is used to create, refresh, modify and remove such state. The common message header contains a TEAR flag that indicates complete QoS NSLP state removal (as opposed to a reservation of zero resources). This QoS NSLP state comprises

reservation state and QoS NSLP operation state. The QoS NSLP indicates to GIMPS that it is no longer interested in the corresponding GIMPS state. The GIMPS then autonomously decides whether or not to keep this state.

The RESERVE message opaquely transports one or more QSPEC objects, describing the desired service level and a POLICY_DATA object, authorizing the requestor of the service. It carries the lifetime of the reservation in the Common Control Information.

RESERVE messages are sent peer-to-peer. This means that a QNE considers its adjacent upstream or downstream peer to be the source of the RESERVE message.

The RESERVE message is idempotent; the resultant effect is the same whether a message is received once or many times. In addition, the ordering of RESERVE messages matters - an old RESERVE message should not replace a newer one. Both of these features are required for protocol robustness - messages may be re-ordered on route (e.g. because of mobility, or at intermediate GIMPS nodes) or spuriously retransmitted. Handling of message re-ordering is supported by the inclusion of the Reservation Sequence Number (RSN) in the RESERVE message.

The sender of a RESERVE message may want to receive confirmation of successful state installation from a downstream node. Therefore, a RESERVE message optionally contains a RESPONSE_REQUEST object (Section 4.2.2).

4.1.2 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 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, nor does it cause state to be installed in nodes other than the one that generated the QUERY.

A QUERY message contains one or more 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 requestor of the QUERY message.

A QUERY message may be scoped if a RESPONSE message from some other node than the QNR is desired.

A QUERY message may contain a RESPONSE_REQUEST object (Section 4.2.2), the contents of which allow matching back RESPONSE messages to the QUERY request. The RESPONSE_REQUEST object is transported unchanged along the data path and may be used to scope the RESPONSE to a QUERY message (Section 4.2.3).

4.1.3 RESPONSE

The REPONSE 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. For this purpose, the RESPONSE message carries one or more QSPEC objects.

The RESPONSE message is impotent, it does not cause any reservation state to be installed or modified.

4.1.4 NOTIFY

NOTIFY messages are used to convey information to a QNE. NOTIFY messages are impotent (they do not cause a change in state directly). They may carry one or more QSPEC objects. An example use of NOTIFY would be to indicate when a reservation has been pre-empted.

NOTIFY messages differ from RESPONSE messages in that they need not refer to any particular state or previously received message. They are sent asynchronously. The NOTIFY message itself does not trigger or mandate any action in the receiving QNE.

The information conveyed by a NOTIFY message is typically related to error conditions. One example would be notification to an upstream peer about state being torn down.

4.2 Control information

Control information conveys information on how specific messages should be handled by a QNE, e.g. sequencing of messages. This may include some mechanisms that are useful for many QoS models (Common Control Information) and some that are for a particular QoS model only (QoS-model specific Control Information). QoS-model specific Control Information is specified together with a QoS model. This specification only defines Common Control Information. Currently, Common Control Information is defined for session identification, message sequencing, response request, message scoping and session lifetime.

4.2.1 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, a QNE may need to detect re-ordered or duplicated RESERVE messages.

Detection of RESERVE message re-ordering or duplication is supported by the Reservation Sequence Number (RSN). The RSN is a counter, locally chosen to be unique (on a hop-by-hop basis) within a session. 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, this means that we can make the Reservation Sequence Number unique just between a pair of neighboring stateful QNEs. By managing the sequence numbers in this manner, the source of the RESERVE does not need to determine how the next NSLP node will process the message.

The RSN refers to a particular instance of the RESERVE state. This allows explicit acknowledgment of state installation actions (by including the RSN in a RESPONSE). It also allows an abbreviated form of refreshing RESERVE message ("summary refresh"). In this case, the refreshing RESERVE references the reservation using the RSN (and the SESSION_ID), rather than including the full reservation specification (including QSPEC, ...). Note that summary 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 RESPONSE_REQUEST to decide whether it wants to accept summary refreshes and provide this explicit acknowledgment.

4.2.2 Requesting responses

Some QNEs may require explicit responses to messages they send. A QNE which sends a QUERY message (Section 4.1), for instance, will require a response with the requested information to be sent to it. A QNE which sends a RESERVE message may want explicit confirmation that the requested reservation state was installed.

A QNE that desires an explicit response includes a RESPONSE_REQUEST object in its message. RESPONSE_REQUEST objects are used in RESERVE and QUERY messages. The RESPONSE_REQUEST object may be used in combination with message scoping (Section 4.2.3) to influence which QNE will respond.

A message contains at most one RESPONSE_REQUEST object. The RESPONSE_REQUEST object contains Request Identification Information (RII) 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.

A message containing a RESPONSE_REQUEST object causes a RESPONSE message to be sent back. The RESPONSE message contains the original RESPONSE_REQUEST object and may be scoped, e.g. using the RII (Section 4.2.3), to influence which (upstream) QNEs will receive the RESPONSE.

4.2.3 Message scoping

For some messages, QNEs may want to restrict message propagation. For a RESERVE message, this may be the case when state installation is required on part of the path towards the destination only. For a QUERY message, it allows limiting the nodes that can respond to the QUERY. For a RESPONSE message, it allows limiting the nodes that receive the RESPONSE.

Message scoping is supported by a SCOPING object. Different scopes are supported. By default, no SCOPING object is present which indicates that the scope is either "whole path" or limited by configuration (and therefore not signalled). Other supported scopes are "single hop" and "back to me". The latter is supported by

copying the RII from the RESPONSE_REQUEST object into the SCOPING object that is put in the RESPONSE message, so that its forwarding can be terminated by the node that requested the RESPONSE.

This specification does not support an explicit notion of a region scope or "to the CRN". If needed, this can be easily proposed as an

extension later on.

4.2.4 State handling

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 NO_REPLACE flag in the common header to indicate that the new session does not replace the existing one. A QNE that receives a RESERVE with the NO_REPLACE flag set but with the same SII will update the flow ID and indicate NO_REPLACE to the RMF (where it will be used for the resource handling). If the SII is different, this means that the QNE is a merge point. In that case, the NO_REPLACE also indicates that a tearing RESERVE SHOULD NOT be sent on the old branch.

At a QNE, resource handling is performed by the RMF. For sessions with the NO_REPLACE flag set, we assume that the QoS-model specific control information 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 summary 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. The combination of different SESSION_IDs (and SIIs) in the same refreshing RESERVE message is currently an open issue.

4.2.5 State timers

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 seconds indicating how long the state that is

signalled for remains valid. Maintaining the reservation beyond this lifetime can be done by sending a ("refreshing") RESERVE message.

The REFRESH_PERIOD 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 given in Section 5.

4.2.6 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 [3]).

A relation between two sessions is indicated by including the BOUND_SESSION_ID object in the messages. 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. Note that the session with SID_B may or may not carry a BOUND_SESSION_ID object containing SID_A.

Three examples where session binding can be used for aggregate reservation, bi-directional reservation and fate sharing are described below.

Aggregated sessions may have a different flow ID from the end-to-end message. If the edge QNEs of the aggregation domain want to maintain some end-to-end properties, they may establish a peering relation by sending the end-to-end message transparantly over the domain. 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.

Including the BOUND_SESSION_ID object in a session indicates a dependency relation. By default, a session that is bound to another

session with the BOUND_SESSION_ID object shares fate with it. This means that if a bound session is torn down, every binding session should be torn down as well. The reverse is not true. If a binding session is torn down, the bound session and other binding sessions remain. A QNE that wants to prevent fate sharing sets a flag in the common header (NO_FATE_SHARING).

Bi-directional reservations are a special case of fate sharing. In this case, a reservation in one direction only makes sense if the reservation in the reverse direction is also up. In some cases bi-directional reservations also allow local optimizations. Therefore, when a reverse reservation is set up, it should carry a BOUND_SESSION_ID containing the SESSION_ID of the forward direction. This SESSION_ID is copied from the sender-initiated (forward) RESERVE or from the QUERY message triggering the sender-initiated (reverse) RESERVE message. Alternatively, it can be inserted by the QNE that sets up both the sender-initiated RESERVE and the receiver-initiated RESERVE.

4.3 Layering

The QoS NSLP supports layered reservations. Layered reservations may occur when certain parts of the network (domains) implement one or more local QoS models, or when they locally apply specific control plane characteristics (e.g. datagram mode instead of connection mode). They may also occur when several per-flow reservations are locally combined into an aggregate reservation.

4.3.1 Local QoS models

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). 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.

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) SHOULD remove the topmost QSPEC object from the stack. It SHOULD update the underlying QoS model parameters if needed.

A QNE that receives a message with a QSPEC object stack of which the topmost object is not understood MUST NOT forward the message and MUST send an error indication to its upstream neighbour. It MUST NOT attempt local recovery by inspecting the stack for a QSPEC object it understands.

4.3.2 Local control plane properties

The way signalling messages are handled is mainly determined by the parameters that are sent over GIMPS-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.

The local session and the end-to-end session are bound at the egress QNE by means of the BOUND_SESSION_ID object. One approach could be that the end-to-end session carries the SESSION_ID of the local session in its session binding object. Another approach could be that the local session carries the SESSION_ID of the end-to-end session in its BOUND_SESSION_ID object. This allows the QNE that performs session binding to maintain end-to-end connection mode.

4.3.3 Aggregate reservations

For scalability reasons, a domain may want to combine two or more end-to-end reservations into a single local aggregate reservation. The domain over which the aggregation is done is limited by configuration.

The essential difference with the layering approaches described in Section 4.3.1 and Section 4.3.2 is that the aggregate reservation needs a FlowID 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 4.3.2 and to the RSVP aggregation solution [10]. In addition to the different FlowID, the aggregate session may specify a local QoS model and local control plane parameters as explained above.

The aggregate reservation may or may not change source and destination IP addresses, i.e. either the end-to-end adresses may be used (if possible) or the IP address of ingress and egress of the domain may be used as source and destination IP address. In some cases, the latter option may cause data plane divergence between both sessions. RSVP solves this by using tunnelling between the edges of the domain.

In any case, session binding and a solution for intermediate node bypass (as explained before) are required in this case as well.

4.4 Extensibility

The QoS NSLP specification foresees future specification of new error codes and new Common Control Information objects. Specification of new messages is not foreseen but not explicitly precluded.

Specification of new error codes and Common Control Information objects is subject to IANA approval and assignment of ClassNum and CType. ClassNum and CType of currently existing objects and error codes are described in Section 6. New Common Control Information objects need to specify whether they are mandatory or optional to implement. Mandatory CCI that is not understood by a QNE needs to generate an error. Optional CCI that is not understood by a QNE needs to be passed transparantly.

The QoS NSLP specification allows future QoS model specific extensions, including the definition of new QoS models, the specification of new objects for existing QoS models, the specification of new processing rules for new or existing objects and the specification of new QoS model specific error codes.

Different types of QoS models are foreseen: standardized QoS models, well-known QoS models and QoS models for private use. We assume the IANA registry of QoS models to distinguish between those. Apart from the QoS model ID, all QoS model specific extensions are opaque to the QoS NSLP (and have no impact on its IANA considerations section).

4.5 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 then during handover. This situation only occurs when GIMPS or QoS NSLP processing is the congested part or scarce resource. This specification requests GIMPS design to foresee a mechanism to support a number of levels of message priority that can be requested over the NSLP-GIMPS API.

Priority of certain reservations over others may be required when QoS resources are oversubscribed. In that case, existing reservations may be preempted in other 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 resource 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 QoS model specific issue and therefore outside the scope of this specification. However, the concepts of setup and holding priority are widely accepted and we expect the specification of a Priority object in the QSPEC template to be useful for a wide range of QoS models.

4.6 Rerouting

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.

Rerouting detection can be performed at three levels. First, routing modules may detect route changes through their interaction with routing protocols. Certain QNEs or GIMPS 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 GIMPS layer. This specification requests GIMPS 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 [5]. When a RESERVE message with an existing SESSION_ID and a different SII is received, the QNE knows its upstream peer has changed.

Reservation on the new path automatically happens when a refreshing RESERVE message arrives at the QNE where the old and the new path diverge. 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 GIMPS peering relations (e.g. by TTL counting the number of GIMPS hops between NSLP peers or the observing changes in the outgoing interface towards GIMPS peer). These mechanisms are outside the scope of this specification.

When the QoS NSLP is aware of the route change, it needs to set up the reservation on the new path. This is done by sending a RESERVE message with RSN+1. 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 GIMPS 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 QNE that has detected the route change via the SII change sends a RESERVE with the TEAR flag set. A QNE that is notified of the route change in another way and want to tear down the old branch needs to send the RESERVE on the new path with a RESPONSE_REQUEST. 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 perform RESPONSE_REQUEST or does not receive a RESPONSE needs to rely on sof-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 NO_REPLACE flag in the common header.

4.7 State storage

For each flow, a QNE stores (QoS model specific) reservation state which is different for each QoS model 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 GIMPS or set locally.

The state information for a given key includes:

Flow ID

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

QoS model ID

8 bit identification of the QoS model.

SII for each upstream and downstream peer

The SII is a large identifier (minimum 128 bits) generated by the GIMPS and passed over the API.

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.

4.8 Authentication and authorization

QoS NSLP requests allow particular user(s) to obtain preferential access to network resources. To prevent abuse, some form of an access control (also known as policy based admission control) will generally be required on users who make reservations. Typically, such authorization is expected to make use of an AAA service external to the node itself. In any case, cryptographic user identification and selective admission will generally be needed when a reservation is requested.

The QoS NSLP request is handled by a local 'resource management' function, which coordinates the activities required to grant and configure the resource. The grant processing involves two local decision modules, 'policy control' and 'admission control'. Policy control determines whether the user is sufficiently authorized to make the reservation. Admission control determines whether the node has sufficient available resources to offer the requested QoS.

4.8.1 Policy Ignorant Nodes

It is generally assumed that policy enforcement is likely to concentrate on border nodes between administrative domains. Figure 15 below illustrates a simple administrative domain with:

- two boundary nodes (A, C), which represent QNEs authorized by AAA entities.
- A core node (B) represents an Policy Ignorant QNE (PIN) with capabilities limited to default admission control handling.

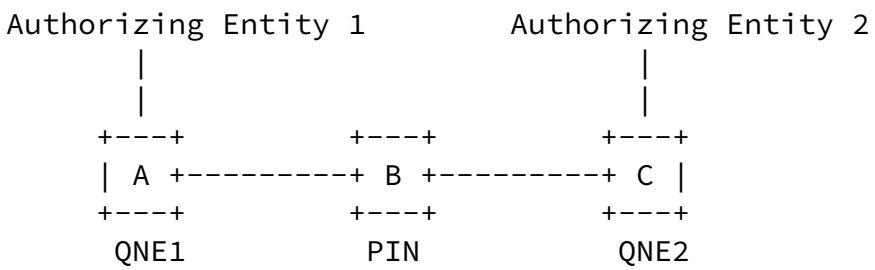


Figure 15: Administrative Domain scenario

Here, policy objects transmitted across the domain traverse an intermediate PIN node (B) that is allowed to process QoS NSLP message but considered non-trusted for handling policy information.

4.8.2 Policy Data

The input to policy control is referred to as "Policy data", which QoS NSLP carries in the Policy object. Policy data may include credentials identifying entities and traits depending on the authorization model in use (2-party, 3-party, token-based 3-party). There are no requirements for all nodes to process this object. 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.

Policy control depends on successful user authentication and authorization of a QoS NSLP reservation request. The authorization decision might be valid for a certain amount of time or even for the entire lifetime of the session. It is a decision of the involved party to trigger a re-authorization procedure. This feature is supported by the Policy Refresh Timer (PRT) option of the Policy object.

Policy objects are carried by QoS NSLP messages and contain policy information. All policy-capable nodes (at any location in the network) can generate, modify, or remove policy objects, even when senders or receivers do not provide, and may not even be aware of policy data objects.

The exchange of Policy objects between policy-capable QNEs along the data path, supports the generation of consistent end-to-end policies.

Furthermore, such policies can be successfully deployed across multiple administrative domains when border nodes manipulate and translate Policy objects according to established sets of bilateral agreements.

5. QoS-NSLP Functional specification

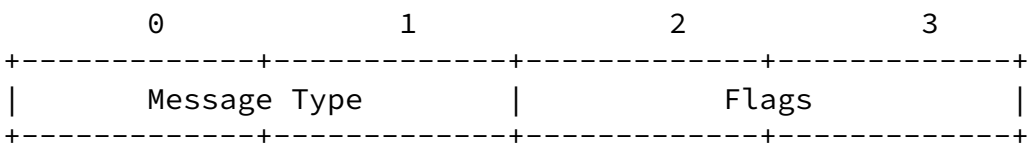
5.1 QoS-NSLP Message and Object Formats

A QoS-NSLP message consists of a common header, followed by a body consisting of a variable number of variable-length, typed "objects". The following subsections define the formats of the common header, the standard object header, and each of the QoS-NSLP message types.

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 (BNF) specified in [2]. The BNF 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 permissible order.

5.1.1 Common header

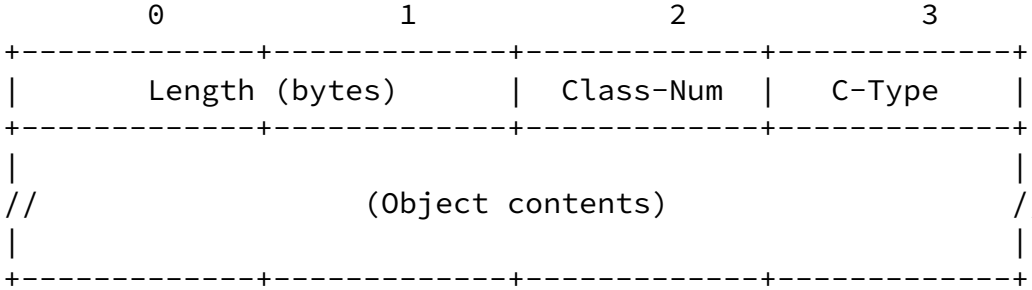


The fields in the common header are as follows:

Msg Type: 16 bits
1 = RESERVE
2 = QUERY
3 = RESPONSE
4 = NOTIFY
Flags: 16 bits
The set of appropriate flags depend 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 ignored on receiving.

5.1.2 Object Formats

Every object consists of one or more 32-bit words with a one-word header, with the following format:



An object header has the following fields:

Length:
A 16-bit field containing the total object length in bytes. Must always be a multiple of 4, and at least 4.

Class-Num:
Identifies the object class; values of this field are defined in Appendix A. Each object class has a name, which is always capitalized in this document. An QoS-NSLP implementation must recognize the following classes:

RESPONSE_REQUEST:
Contains the request for the generation of a response message and the Request Identification Information (RII).

RSN:
The Reservation Sequence Number (RSN) contains an incrementing sequence number that indicates the order in which state modifying actions are performed by a QNE. The RSN has local significance only, i.e. between a pair of neighbouring stateful QNEs. RSN is a common control information object.

REFRESH_PERIOD:
Contains the value for the refresh period R used by the creator of the message. Required in every RESERVE message. REFRESH_PERIOD is a common control information object.

BOUND_SESSION_ID:
It represents the Session ID as specified in [15] of the session that must be bound to the session associated to the message carrying this object.

SCOPING:
contains information that limits the scope of the message carrying this object. When no SCOPING object is available in a message it means that its scoping is either the whole path or it is defined by configuration. SCOPING is a common control information object.

ERROR_SPEC:
Contains an error code and can be carried by a Response or a NOTIFY message. ERROR_SPEC is a common control information object.

POLICY_DATA:

Carries authentication, authorization and accounting information.

QSPEC:
Carries the information that is QoS model specific. This information consists of the QoS model specific control information and the QoS specification parameters.

C-Type:
Object type, unique within Class-Num. Values are defined in Appendix A.

The maximum object content length is 65528 bytes. The Class-Num and C-Type fields may be used together as a 16-bit number to define a unique type for each object.

The high-order two bits of the Class-Num are used to determine what action a node should take if it does not recognize the Class-Num of an object. The first two bits of the Class-Num take one of the following three values:

00 - Abort processing and send error back
01 - Ignore object, and do not forward it in onward message
10 - Ignore object, but forward it in onward message

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.

The state used by the QoS NSLP is listed in Section 4.7.

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 NI or NR) as the message received.

The decision on whether to generate a message to forward may be affected by the presence of a SCOPING 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 first two bits in the type value.

5.3 Object Processing

5.3.1 Reservation Sequence Number

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. Once the RSN has reached its maximum value, the next value it takes is zero.

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 the same as 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 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 Response Request

A QNE sending some types of message may require a response to be sent. It does so by including a RESPONSE_REQUEST object.

When creating a RESPONSE_REQUEST object the sender MUST select the value for the RII such that it is probabilistically unique within the given session.

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 is selected by another QNE then RESPONSE messages may be incorrectly terminated, and not passed back to the node that requested them.

When sending a RESPONSE_REQUEST the sending node MUST remember the

value used in the RII to match back any RESPONSE received. It 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.

When receiving a message containing a RESPONSE_REQUEST object the node MUST send a RESPONSE if either

- The message contained a SCOPING object with a value of 'next hop', 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.

5.3.3 Bound Session ID

As shown in the examples in Section 3.5, 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.

5.3.4 Refresh Period

Refresh timer management values are carried by the REFRESH_PERIOD object. The details of timer management and timer changes (slew handling and so on) are identical to the ones specified in Section 3.7 of [5]. 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:

1. Floyd and Jacobson [26] 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 Scoping

When sending some types of message, the node may wish to limit the distance that the message should travel along the path (the default being the whole path). To do so a node MUST include a SCOPING object.

When receiving a message, before sending the message further, the QNE MUST inspect any scoping object to determine if it has reached the end of the scoped region. If so, it MUST NOT pass it further, and

MUST generate a RESPONSE if a RESPONSE_REQUEST object is present.

5.3.6 Error Spec

The ERROR_SPEC object contains a value which indicates the condition that occurred. Error values are to be specified for various conditions, such as:

- OK
- Message type not supported
- Object type not supported
- Insufficient resources
- Authentication failure
- Authorisation denied
- QoS model specific condition occurred
- ...

5.3.7 Policy Data

POLICY_DATA objects may contain various items to authenticate the user and allow the reservation to be authorised. Some possible contents are given in Appendix A.7, and some issues are also discussed in Section 3.3.

5.3.8 QSpec

The contents of the QSpec depends on the QoS model being used. It may be that parts of the QSpec are standardised across multiple QoS models. This topic is currently the topic of further study.

5.4 Message Processing Rules

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 as follows:

```
RESERVE = COMMON_HEADER
          RSN [ SCOPING ] [ RESPONSE_REQUEST ]
          [ REFRESH_PERIOD ] [ BOUND_SESSION_ID ]
          [ POLICY_DATA ] [ *QSPEC ]
```

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

- TEAR - when set, indicates that reservation state and QoS NSLP operation state should be torn down. This is indicated to the RMF.
- NO_REPLACE - when set, indicates that a RESERVE with different Flow Routing Information (FRI) does not replace an existing one, so the old one should not be torn down immediately
- NO_FATE_SHARING - when set, indicates that sessions in the bundle should not share fate with one another

An RSN MUST be present.

RESERVE messages MUST only be sent towards the NR.

If the QNE sending a RESERVE message wishes to use the reduced overhead refresh mechanism described in Section 4.2.1, then it SHOULD include a RESPONSE_REQUEST in the RESERVE message. 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.

Before admitting a reservation 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 a QNE receives a RESERVE message, its processing may involve sending out a RESERVE message. When doing so, the QNE may insert or remove 'local' QSPEC objects from the top of the stack. If there are one or more QSPECs in the received RESERVE message, the last QSPEC MUST NOT be removed when sending on the RESERVE message.

When a reservation is no longer required the QNI SHOULD send a RESERVE message with the TEAR bit set in the header. On receiving such a RESERVE message the QNE MUST remove any reservation state for that session. It SHOULD remove the QoS NSLP state. It MAY signal to the NTLP that it is no longer interested in NTLP state for that session.

5.4.2 QUERY Messages

A QUERY message is used to request information about the data path without making a reservation. This functionality can be used to 'probe' the network for path characteristics or for support of

certain QoS models. 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, nor does it cause state to be installed in nodes other than the one that generated the QUERY.

The format of a QUERY message is as follows:

```
QUERY = COMMON_HEADER
        [ SCOPING ] [ RESPONSE_REQUEST ]
        [ BOUND_SESSION_ID ]
```

[POLICY_DATA] [*QSPEC]

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 flags are defined are defined for use with the QUERY message.

A QUERY message MAY be scoped using the SCOPING object.

A QUERY message MUST contain a RESPONSE_REQUEST object, unless the QUERY is being used to initiate reverse-path state for a receiver-initiated reservation.

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.

On receiving a QUERY message, the QoS model specific QUERY processing MAY cause the QSpec to be modified, to provide the answer to the query. Future QoS NSLP objects may be added to the protocol with similar properties in this respect.

If this is last node to process this QUERY message (either because it is the last node on the path, or because of scoping), and a RESPONSE_REQUEST object is present, then a RESPONSE message MUST be generated and passed back along the reverse of the path used by the QUERY.

If this is the last node and a RESPONSE_REQUEST object is not present then the QUERY is a trigger to initiate a reservation in the reverse direction. If this node was not expecting to perform a receiver-initiated reservation then an error MUST be sent back along the path.

When generating a QUERY to send out to pass the query further along the path, the QNE MUST copy the RESPONSE_REQUEST (if present) into

the new QUERY message unchanged.

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 as follows:

```
RESPONSE = COMMON_HEADER
           [ SCOPING / RSN ] ERROR_SPEC
           [ *QSPEC ]
```

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 flags are defined are defined for use with the RESPONSE message.

A RESPONSE message MUST be sent where the QNE is the last node to process a RESERVE or QUERY message containing a RESPONSE_REQUEST 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 contain a SCOPING object of type RII. The contents of the RESPONSE_REQUEST object in the received RESERVE or QUERY message MUST be copied into this SCOPING object in the RESPONSE.

In addition, a RESPONSE message MUST be sent when an error occurs while processing a received message message. If the received message contains a RESPONSE_REQUEST object, then an RII SCOPING 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 a RESPONSE_REQUEST object, then the RSN of the received RESERVE message MUST be copied into the RESPONSE message.

A RESPONSE message MUST contain an ERROR_SPEC object which indicates the success of a reservation installation or an error condition. Depending on the value of the ERROR_SPEC, the RESPONSE MAY also contain a QSPEC object.

On receipt of a RESPONSE message containing an RII SCOPING 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 match fails, then the QNE MUST attempt to forward the RESPONSE to the next peer QNE.

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 error MUST be processed. The RESPONSE message MUST NOT be forwarded further along the path whether or not the match succeeds.

5.4.4 NOTIFY Messages

NOTIFY messages are used to convey information to a QNE asynchronously. NOTIFY messages are impotent (they do not cause a change in state directly).

The format of a NOTIFY message is as follows:

```
NOTIFY = COMMON_HEADER
         ERROR_SPEC [ QSPEC ]
```

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

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

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 [7].

The QoS NSLP requires IANA to create two registries. One for QoS NSLP message types, the other for QoS NSLP objects.

This specification defines four message types: RESERVE=1, QUERY=2, RESPONSE=3 and NOTIFY=4. Values are taken from the Message type name space (8 bits). New Message types may be defined and assigned values by IANA. For this, standards action is required.

Common Control Information has a Class and C-type assigned by IANA. This specification defines the following Common Control Information objects
RESPONSE_REQUEST: Class=1

C-type=1: empty
C-type=2: Request Identification Information

RSN: Class=2

C-type=1: RSN

REFRESH_PERIOD: Class=3

C-type=1: REFRESH_PERIOD

SESSION_ID: Class=4

C-type=1: SESSION_ID

SCOPING: Class=5

C-type=1: single hop
C-type=2: Region scoping
C-type=3: RII scoping

ERROR_SPEC: Class=6

C-type=1: empty

IANA will assign new ClassNum values and/or C-type for Common Control Information upon specification. The required specification needs to indicate what the correct behaviour is in case the new ClassNum or C-type is not understood.

This specification defines a QSPEC object with assigned class = 8. The C-type identifies the QoS model, which can be standardized, well-known or private.
Standardized

Standardized QoS models have a C-type value in the range of 1-64. C-type values for standardized QoS models are assigned by IANA and require standards action.

Well-known

Well-known QoS models have a C-type value in the range of 65-128. They are assigned by IANA and require IETF consensus.

Private

C-type values from the range 129-256 are for private use.

7. Requirements for the NSIS Transport Layer Protocol (GIMPS)

For the moment this section will merely describe what we assume and/or request to be available from GIMPS. This section will later be updated to describe the eventual interface when GIMPS work gets finalized.

7.1 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. QoS NSLP will pick a value for the SESSION_ID and pass it over the API.

7.2 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 4.3), 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.

As a suggestion, we identified two ways for bypassing intermediate nodes. One solution is for the end-to-end session to carry a different protocol ID (QoS-NSLP-E2E-IGNORE protocol ID, similar to the RSVP-E2E-IGNORE that is used for RSVP aggregation ([10]). Another solution is based on the use of multiple levels of the router alert option. In that case, internal routers are configured to handle only certain levels of router alerts. The choice between both approaches or another approach that fulfills the requirement is left

to GIMPS design.

7.3 Support for peer change identification

There are several circumstances where it is necessary for a QNE to identify the adjacent QNE peer, which is the source of a signaling application message; for example, 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.

We rely on GIMPS to provide this functionality and suggest it be implemented as an opaque identifier (Source Identification Information (SII)) which, by default, all outgoing QoS-NSLP messages are tagged with at GIMPS layer. This identifier is propagated to the

next QNE, where it can be used to identify the state associated with the message; The SII is logically similar to the RSVP_HOP object of [5]; however, any IP (and possibly higher level) addressing information is not interpreted in the QoS-NSLP. Indeed, the intermediate GIMPS nodes could enforce topology hiding by masking the content of the SII (provided this is done in a stable way).

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 send initiate a teardown along the old section of the path. This functionality would require GIMPS to be able to route based on the SII. We would like this functionality to be available as a service from GIMPS.

7.4 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 GIMPS level as well. Such nodes should not worry about keeping reverse state, message fragmentation and reassembly (at GIMPS), congestion control or security associations. A stateless or reduced state QNE will be able to inform the underlying GIMPS of this situation. We rely on GIMPS design to allow for a mode of operation that can take advantage of this information.

7.5 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 GIMPS. We rely on GIMPS 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. It requires GIMPS to have an error message indicating that no more NSLPs of a particular type are available on the path.

7.6 Re-routing detection

Route changes may be detected at the GIMPS layer or the information may be obtained by GIMPS through local interaction with or notification from routing protocols or modules. This specification requests the GIMPS design to foresee notification of a route change (over the API) to the QNEs upstream of the NE where the route change is detected.

7.7 Priority of signalling messages

The QoS-NSLP will generate messages with a range of performance requirements for GIMPS. These requirements may result from a prioritization at the QoS-NSLP (Section 4.3) or from the responsiveness expected by certain applications supported by the QoS-NSLP.

GIMPS design should be able to ensure that performance for one class of messages was not degraded by aggregation with other classes of

messages. It is currently an open issue how many priority levels are required.

7.8 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.

GIMPS 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.

This might be realised by GIMPS by a mixture of GIMPS node counting, and examination of the IP TTL or Hop Limit. The QoS NSLP, however, does not need to know the number of intermediate nodes, only that one or more exists.

7.9 NSLP Data Size

When GIMPS passes the QoS NSLP data to the NSLP for processing, it must also indicate the size of that data. (It is assumed that GIMPS message structure will indicate how long this part of GIMPS message is.)

7.10 Notification of NTLP 'D' flag value

When the NTLP passes the QoS NSLP data to the NSLP for processing, it must also indicate the value of the 'D' (Direction) flag for that

message.

7.11 NAT Traversal

The QoS NSLP relies on GIMPS 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 NO_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. A COMMIT flag in the QoS-model specific control information is suggested as a way to support this functionality.

9. Open issues

9.1 Region scoping

This specification allows QNEs to scope their messages, i.e. to restrict the extent to which messages may travel along and be interpreted on the path. For this, the scopes of whole path, single hop and back to me (RII) are defined. Also, a region can be configured administratively or it can be derived from some other means (e.g. RAO levels) in case of aggregation.

This specification currently does not define and support a more generic notion of region (e.g. to implement region policies independent from aggregation regions,...). It is proposed to use the concept of Localized RSVP for regions.

9.2 Priority of reservations

Priority of certain reservations over others may be required when QoS resources are oversubscribed. In that case, existing reservations may be preempted in other 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 resource 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 QoS model specific issue and therefore outside the scope of this specification. However, the concepts of setup and holding priority are widely accept and we expect the specification of a Priority object in the QSPEC template to be useful for a wide range of QoS models.

It is an open question to the NSIS community whether the concepts of setup and holding priority are useful enough to define a priority object in this specification. Alternatively, this could be left as QoS model specific.

9.3 Peering agreements on interdomain links

This specification proposes ways to carry AAA information that may be used at the edges of a domain to check whether the requestor is allowed to use the requested resources. It is less likely that the AAA information will be used inside a domain. In practice, there may be peering relations between domains that allow for a certain amount of traffic to be sent on an interdomain link without the need to check the authorization of each individual session (effectively making the peering domain the requestor of the resources). The per-session authorization check may be avoided by setting up an aggregate reservation on the inter-domain link for a specified amount of resources and relating the end-to-end sessions to it using the BOUND_SESSION_ID. In this way, the aggregate session is authorized once (and infrequently updated). An alternative is for the edge node of a domain to insert a token that authorizes the flow for the next domain.

9.4 GIMPS Modifications for Refresh Overhead Reduction

As described in Section 4.2.4 a mechanism is available to reduce the overhead of refresh messages. As currently specified, GIMPS may opt to bundle several NSLP messages together. However, this still has some additional overhead, particularly due to the requirement to include Flow Routing Information (FRI) in every message. This may not be strictly necessary for a message going only to the next GIMPS hop.

It is an open issue to examine what additional optimisations are possible and appropriate for refresh overhead reduction.

9.5 Path state maintenance implementation at NSLP

As currently described in Section 3.5.3, a QoS NSLP message is required to create the necessary GIMPS reverse path state so that the receiver-initiated RESERVE message can be sent upstream. (This must be an NSLP message so that it visits the correct subset of GIMPS nodes on the path.) This message may also be used to refresh the GIMPS path state (see Section 9.6).

In this version, we have implemented this functionality by making the RESPONSE_REQUEST object in QUERY optional. Basically, this means that we send an empty QUERY message along the path. It is not clear whether this path state maintenance functionality is sufficiently different from QUERY to warrant the definition of a separate (NULL) message.

It is also worth investigating whether other NSLPs have a similar need and whether in that case it would be better to define a separate NULL message across all NSLPs.

9.6 GIMPS Path State Maintenance

As currently described in Section 3.5.3, when performing receiver-initiated reservations the QoS NSLP sends periodic downstream QUERY messages to ensure that GIMPS reverse path state is refreshed.

It is unclear how often such messages need to be sent, since the lifetime of the path state is a matter for GIMPS. It, therefore, may be necessary for GIMPS to inform the NSLP about this state lifetime.

An alternative solution to the refreshing of GIMPS path state by NSLP messages is for GIMPS to refresh (and, where necessary, detect changes in) its path state automatically, by sending periodic probe/ refresh messages of its own. The transmission of upstream messages

(such as the RESERVE in a receiver-initiated QoS NSLP reservation) can be used to indicate that the reverse path state is still needed. Whether such a technique is appropriate is for further consideration.

9.7 Protocol Operating Environment Assumptions

For receiver initiated and bidirectional reservations the question arises of what assumptions to make about what end-to-end information should be determined outside the NSIS protocols and what should be carried end-to-end in NSLP messages in order to initiate signalling. The NSIS protocol is not used alone. It is used in conjunction with a variety of applications. The question arises of what the interactions between NSIS (and the NSLP in particular) and these applications is.

For a receiver initiated reservation, the we have the questions: How do the sender and receiver determine that a receiver initiated reservation is to be performed? And, how does information needed by the receiver to perform the reservation, but only available at the sender, be made transferred to the receiver so that the RESERVE message can be sent?

In the bi-directional reservation case, we can either perform this as a pair of two sender-initiated reservations or as a combination of sender-initiated and receiver-initiated reservations. The latter case has the same issues as for the general receiver initiated reservation problem. The former raises similar questions: How does the remote end know that a reservation is needed? And, how does it know what resources to request?

Is it reasonable to assume that the decision that an end should initiate a reservation is made totally outside the QoS NSLP itself (e.g. through prior configuration, or application end-to-end signalling such as SIP) or, should the QoS NSLP messages include some method to trigger the other end to perform a reservation (whether that be a receiver initiated reservation, or a sender initiated reservation for the first bidirectional reservation case)?

In addition, should the QoS NSLP messages be able to carry extra data (e.g. a QSpec object for the reverse direction) end-to-end that is needed by the remote end to perform its reservation? (And, should this be in the QoS NSLP, or through individual QoS models?) The

alternative to providing support in the QoS NSLP for this is to leave it to application signalling to transfer any required information.

10. Security Considerations

10.1 Introduction and Threat Overview

The security requirement for the QoS NSLP is to protect the signaling exchange for establishing QoS reservations against identified security threats. For the signaling problem as a whole, these threats have been outlined in [21]; the NSIS framework [15] assigns a subset of the responsibility to GIMPS 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 requestor. 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:

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

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

Prevention of Denial of Service Attacks:

GIMPS 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.

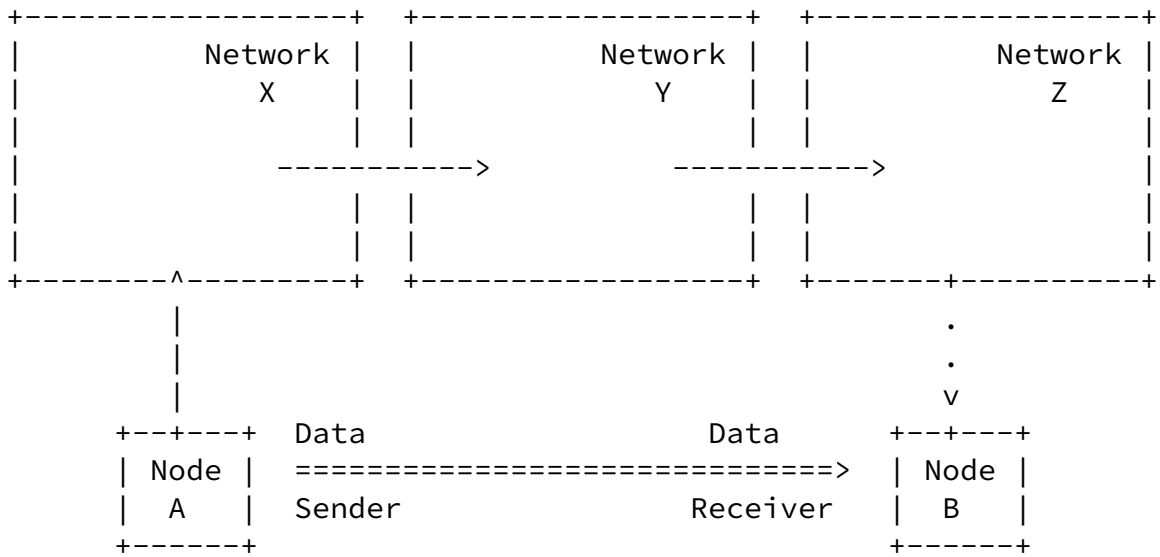
To some extent the QoS NSLP relies on the security mechanisms provided by GIMPS which by itself relies on existing authentication and key exchange protocols. Some signaling messages cannot be protected by GIMPS and hence should be used with care by the QoS

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 GIMPS peers. If a level of security protection for QoS NSLP messages is required which goes beyond the security offered by GIMPS 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.

10.2 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 signaling 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 signaling in the Internet this procedure is roughly similar. In most cases the data sender is charged for transmitted data traffic where charging is provided only between neighboring entities.



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 22: New Jersey Turnpike Model

The model shown in Figure 22 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:

- This model allows authorization on a request basis or on a per-session basis. Authorization mechanisms will be elaborated in Section 3.6. 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.
- 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.

- 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 signaling messages. The identity used during the authentication and key establishment phase would be used by Network X (see Figure 22) to perform the so-called policy-based admission control procedure. In our context this user identifier would be used to establish the necessary infrastructure to provide authorization and charging. Signaling messages later

exchanged between the different networks are then also subject to authentication and authorization. The authenticated entity thereby is, however, the neighboring network and not the end host.

The New Jersey Turnpike model is attractive because of its simplicity. S. Schenker et. al. [24] 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.

10.3 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.

11. Change History

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 QoS model.
* Added assumptions on operating environment.

12. Acknowledgements

The authors would like to thank Eleanor Hepworth for her useful comments.

13. 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.

14. References

14.1 Normative References

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

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

[3] Schulzrinne, H., "GIMPS: General Internet Messaging Protocol for Signaling", draft-ietf-nsis-ntlp-01 (work in progress), February 2004.

14.2 Informative References

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

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

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

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

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

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

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

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

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

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

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

[15] Hancock, R., "Next Steps in Signaling: Framework", draft-ietf-nsis-fw-05 (work in progress), October 2003.

[16] Tschofenig, H., "NSIS Authentication, Authorization and

Accounting Issues", draft-tschofenig-nsis-aaa-issues-01 (work in progress), March 2003.

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

[18] Ash, J., "NSIS Network Service Layer Protocol QoS Signaling Proof-of-Concept", draft-ash-nsis-nslp-qos-sig-proof-of-concept-01 (work in progress), February 2004.

[19] Kappler, C., "A QoS Model for Signaling IntServ Controlled-Load Service with NSIS", draft-kappler-nsis-qosmodel-controlledload-00 (work in progress), February 2004.

[20] Bader, A., "RMD (Resource Management in Diffserv) QoS-NSLP model", draft-bader-rmd-qos-model-00 (work in progress), February 2004.

[21] Tschofenig, H. and D. Kroeselberg, "Security Threats for NSIS", draft-ietf-nsis-threats-04 (work in progress), February 2004.

[22] Westberg, L., "Resource Management in Diffserv (RMD) Framework", draft-westberg-rmd-framework-04.txt, work in progress, September 2003.

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

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

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

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

[27] 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.

Authors' Addresses

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. Object Definitions

The currentlly specified C-Types definitions are contained in this Appendix. To accommodate other address families, additional C-Types could easily be defined.

All unused fields should be sent as zero and ignored on receipt.

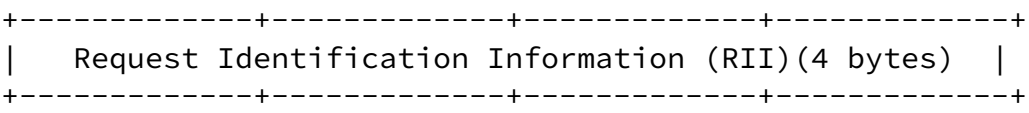
A.1 RESPONSE_REQUEST Class

RESPONSE_REQUEST Class = 1.

RESPONSE_REQUEST object: Class = 1, C-Type = 1

The object content is empty

RESPONSE_REQUEST object: Class = 1, C-Type = 2



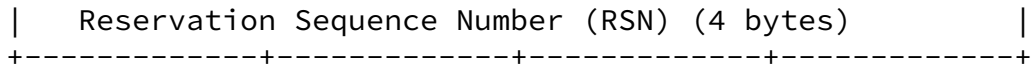
Request Identification Information (RII) (4 bytes)

An identifier which must be (probabilistically) unique within the context of a SESSION_ID, and SHOULD be different for each response request. Used by a node to match back a RESPONSE to a request in a RESERVE or QUERY message.

A.2 RSN Class

RSN class = 2.

RSN object: Class = 2, C-Type = 1



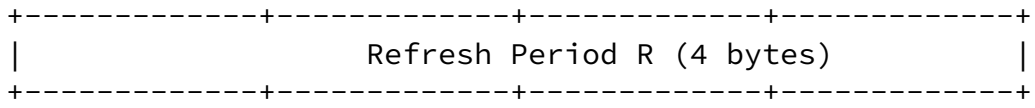
Reservation Sequence Number (RSN) (4 bytes)

An incrementing sequence number that indicates the order in which state modifying actions are performed by a QNE. It has local significance only, i.e. between a pair of neighbouring stateful QNEs.

A.3 REFRESH_PERIOD Class

REFRESH_PERIOD class = 3.

REFRESH_PERIOD Object: Class = 3, C-Type = 1



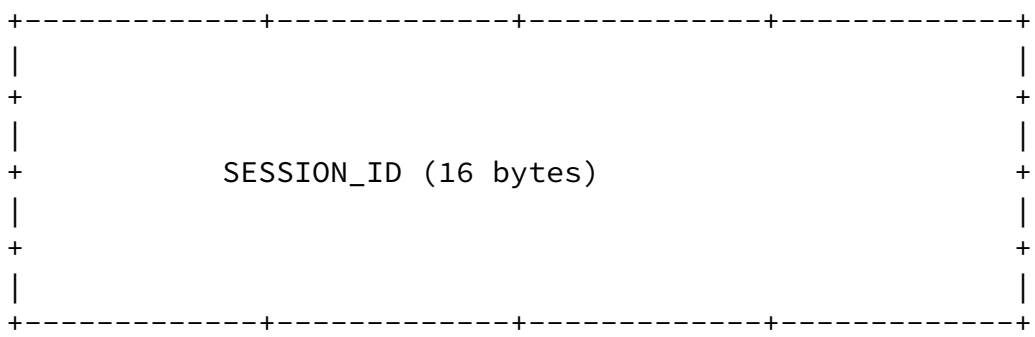
Refresh Period R (4 bytes)

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

A.4 SESSION_ID Class

SESSION_ID class = 4.

SESSION_ID Object: Class = 4, C-Type = 1



SESSION_ID (16 bytes)

It represents the SESSION_ID as specified in [15] of the session that must be bound to the session associated to the message carrying this object.

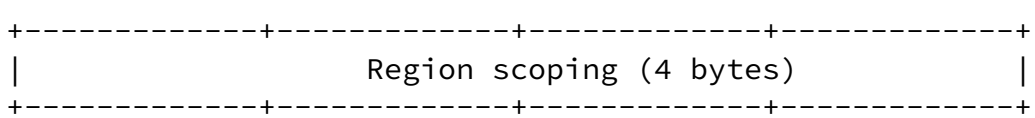
A.5 SCOPING Class

SCOPING class = 5.

SCOPING Object: Class = 5, C-Type = 1

No content value. Selection of a single hop message scoping.

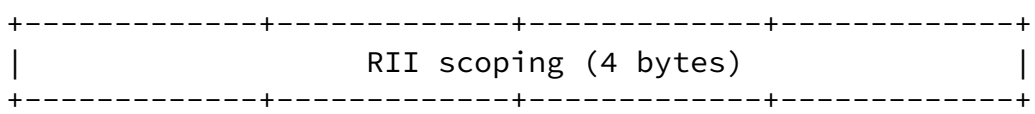
SCOPING Object: Class = 5, C-Type = 2



Region scoping (4 bytes)

Ordered number, forwarded by routers belonging to region with same or higher number;

SCOPING Object: Class = 5, C-Type = 3



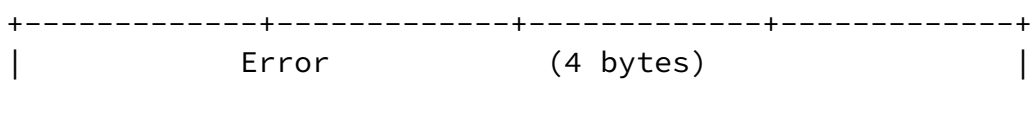
RII (back to me) scoping (4 bytes)

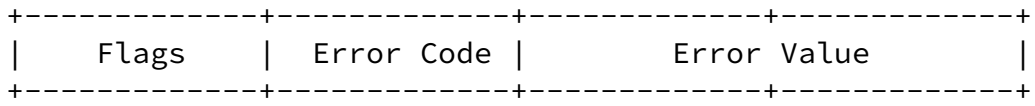
An identifier which must be (probabilistically) unique within the context of a SESSION_ID, and SHOULD be different for each response request. Used by a node to match back a RESPONSE to a request in a RESERVE or QUERY message.

A.6 ERROR_SPEC Class

ERROR_SPEC class = 6.

ERROR_SPEC object: Class = 6, C-Type = 1





Error (4 bytes)

To be done

Flags (1 byte)

To be done

Error Code (1 byte)

A one-octet error description.

Error Value (2 bytes)

A two-octet field containing additional information about the error. Its contents depend upon the Error Type.

The values for Error Code and Error Value are defined in Appendix B (to be done).

A.7 POLICY_DATA Class

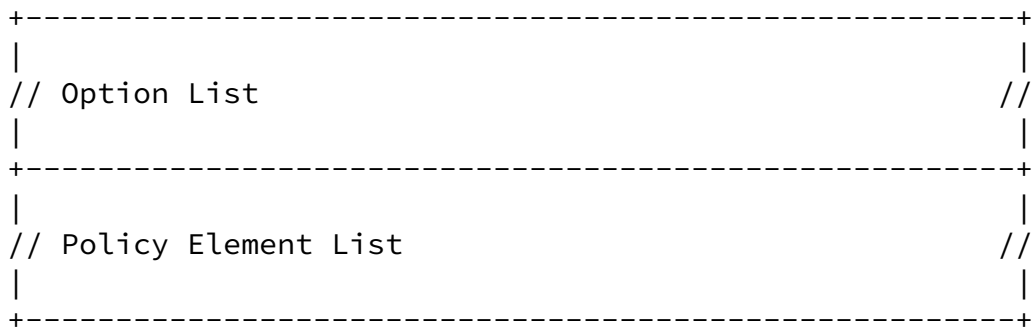
This section presents a set of specifications for supporting generic authorization in QoS NSLP. These specs include the standard format of POLICY_DATA objects, and a description of QoS NSLP handling of authorization events. This section does not advocate a particular authorization approach (2-party, 3-party, token-based 3-party).

The traffic control block is responsible for controlling and

enforcing access and usage policies.

A.7.1 Base Format

POLICY_DATA object: Class=7, C-Type=1



Option List: Variable length. See more details in Appendix A.7.2.
Policy Element List: Variable length. See more details in Appendix A.7.3.

A.7.2 Options

This section describes a set of options that may appear in POLICY_DATA objects. Some policy options appear as QoS NSLP objects but their semantic is modified when used as policy data options.

Policy Refresh TIME_VALUES (PRT) object:

The Policy Refresh TIME_VALUES (PRT) option is used to slow policy refresh frequency for policies that have looser timing constraints relative to QoS NSLP. If the PRT option is present, policy refreshes can be withheld as long as at least one refresh is sent before the policy refresh timer expires. A minimal value for PRT is the NSLP session refresh period R; lower values are assumed to be R (neither error nor warning should be triggered). This option is especially useful to combine strong (high overhead) and weak (low overhead) authentication certificates as policy data. In such schemes the weak certificate can support admitting a reservation only for a limited time, after which the strong certificate is required. This approach may reduce the overhead of POLICY_DATA processing. Strong certificates could be transmitted less frequently, while weak certificates are included in every QoS NSLP refresh.

Policy Source Identification Information (PSII) object:

The Policy SII object identifies the neighbor/peer policy-capable QN that constructed the policy object. When policy is enforced at border QNEs, peer policy nodes may be several NSLP hops away from each other and the SII is the basis for the mechanism that allows them to recognize each other and communicate safely and directly. As stated above, we assume such an (P)SII to be available from a service from GIMPS. If no PSII object is present, the policy data is implicitly assumed to have been constructed by the QoS NSLP HOP indicated in the SII (i.e., the neighboring QoS NSLP node is policy-capable).

Integrity object:

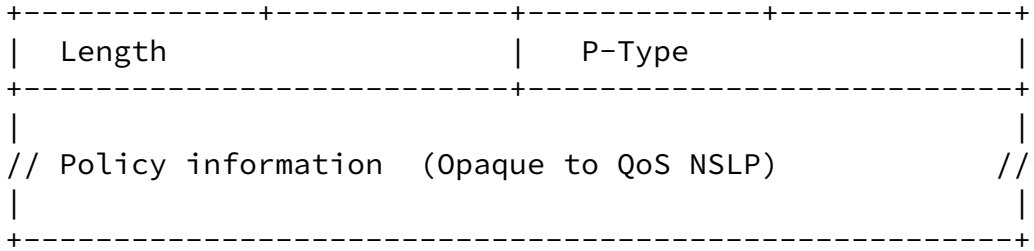
The INTEGRITY object option inside POLICY_DATA object creates direct secure communications between non-neighboring policy aware nodes without involving PIN nodes.

A.7.3 Policy Elements

There are no requirements for all nodes to process this container. Policy data is opaque to NSLP, which simply passes it to policy control when required.

The content of policy elements is opaque to the QoS NSLP layer. Only policy peers understand their internal format and NSLP layer simply passes it to policy control when required.

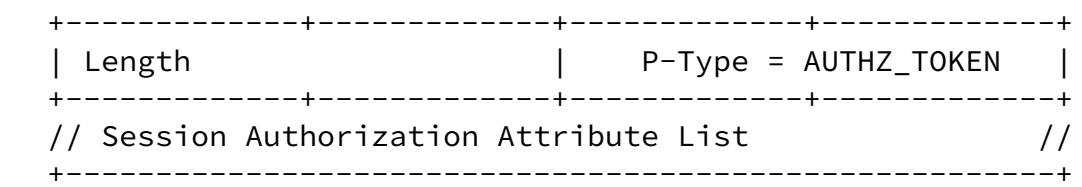
Policy Elements have the following format:



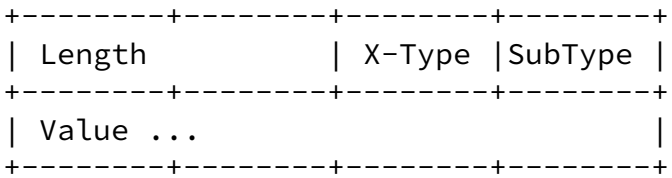
A.7.3.1 Authorization token Policy Element

The AUTHZ_TOKEN policy element contains a list of fields, which

describe the session, along with other attributes.



Session Authorization Attribute List: variable length. The session authorization attribute list is a collection of objects which describes the session and provides other information necessary to verify the resource reservation request. See [11] for a details.
Session Authorization Attributes. A session authorization attribute may contain a variety of information and has both an attribute type and subtype. The attribute itself MUST be a multiple of 4 octets in length, and any attributes that are not a multiple of 4 octets long MUST be padded to a 4-octet boundary. All padding bytes MUST have a value of zero.



Length: 16 bits

The length field is two octets and indicates the actual length of the attribute (including Length, X-Type and SubType fields) in number of octets. The length does NOT include any bytes padding to the value field to make the attribute a multiple of 4 octets long.

X-Type: 8 bits

Session authorization attribute type (X-Type) field is one octet.

IANA acts as a registry for X-Types as described in Section 6. Initially, the registry contains the following X-Types:

1 AUTH_ENT_ID: The unique identifier of the entity which authorized the session.
2 SESSION_ID: Unique identifier for this session.
3 SOURCE_ADDR: Address specification for the session originator.

4 DEST_ADDR: Address specification for the session end-point.
5 START_TIME: The starting time for the session.
6 END_TIME: The end time for the session.
7 RESOURCES: The resources which the user is authorized to request.
8 AUTHENTICATION_DATA: Authentication data of the session authorization policy element.

SubType: 8 bits

Session authorization attribute sub-type is one octet in length. The value of the SubType depends on the X-Type.

Value: variable length

The attribute specific information is defined in [11].

A.7.3.2 OSP Token Policy Element

To be completed.

A.7.3.3 User Identity Policy element

To be completed.

A.8 QSPEC Class

QSPEC class = 8.

QSPEC object: Class = 8, C-Type = (QoS model ID)

This object contains the QSPEC (QoS specification) information. Its content has a variable length and it is QoS model specific. Such a QoS model can be a standardized one, a private one, or a well-known one. The C-Type contains the QoS model ID that identifies the used QSPEC.

The contents and encoding rules for this object are specified in other documents, prepared by QoS model designers.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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