NSIS
Internet-Draft
Expiration Date: April 3, 2008

X. Fu
B. Schloer
Univ. Goettingen
H. Tschofenig
T. Tsenov
Nokia Siemens Networks

October 4, 2007

QoS NSLP State Machine
draft-fu-nsis-qos-nslp-statemachine-06.txt

Status of this Memo

Copyright Notice

Abstract

This document describes a state machine for the NSIS Signaling Layer Protocol for Quality-of-Service signaling (QoS NSLP). A combined state machine for QoS NSLP entities at different locations of a flow path is presented in order to illustrate how QoS NSLP may be implemented.

Table of Contents

1. Introduction

This document describes a state machine for QoS NSLP [1], trying to show how QoS NSLP can be implemented to support its deployment. The state machine described in this document is illustrative of how the QoS NSLP protocol defined in [1] may be implemented for QoS NSLP nodes in the flow path. Where there are differences [1] are authoritative. The state machine diagrams are informative only. Implementations may achieve the same results using different methods.

According to [1], there are several possibilities for QoS NSLP signaling, at least including the following: - end-to-end signaling vs. scoped signaling - sender-initiated signaling vs. receiver-initiated signaling.

The messages used in the QoS NSLP protocol can be summarized as follows:

Requesting message	Responding message
RESERVE	None or RESERVE or RESPONSE
QUERY	RESERVE or RESPONSE
RESPONSE	NONE
NOTIFY	NONE

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [2].

3. Notational conventions used in state diagrams

The following text is reused from [3] and the state diagrams are based on the conventions specified in [4], Section 8.2.1. Additional state machine details are taken from [5].

The complete text is reproduced here:

State diagrams are used to represent the operation of the protocol by a number of cooperating state machines each comprising a group of connected, mutually exclusive states. Only one state of each machine can be active at any given time.

All permissible transitions between states are represented by arrows,

the arrowhead denoting the direction of the possible transition. Labels attached to arrows denote the condition(s) that must be met in order for the transition to take place. All conditions are expressions that evaluate to TRUE or FALSE; if a condition evaluates to TRUE, then the condition is met. The label UCT denotes an unconditional transition (i.e., UCT always evaluates to TRUE). A transition that is global in nature (i.e., a transition that occurs from any of the possible states if the condition attached to the arrow is met) is denoted by an open arrow; i.e., no specific state is identified as the origin of the transition. When the condition associated with a global transition is met, it supersedes all other exit conditions including UCT. The special global condition BEGIN supersedes all other global conditions, and once asserted remains asserted until all state blocks have executed to the point that variable assignments and other consequences of their execution remain unchanged.

On entry to a state, the procedures defined for the state (if any) are executed exactly once, in the order that they appear on the page. Each action is deemed to be atomic; i.e., execution of a procedure completes before the next sequential procedure starts to execute. No procedures execute outside of a state block. The procedures in only one state block execute at a time, even if the conditions for execution of state blocks in different state machines are satisfied, and all procedures in an executing state block complete execution before the transition to and execution of any other state block occurs, i.e., the execution of any state block appears to be atomic with respect to the execution of any other state block and the transition condition to that state from the previous state is TRUE when execution commences. The order of execution of state blocks in different state machines is undefined except as constrained by their transition conditions. A variable that is set to a particular value in a state block retains this value until a subsequent state block executes a procedure that modifies the value.

On completion of all of the procedures within a state, all exit conditions for the state (including all conditions associated with global transitions) are evaluated continuously until one of the conditions is met. The label ELSE denotes a transition that occurs if none of the other conditions for transitions from the state are met (i.e., ELSE evaluates to TRUE if all other possible exit conditions from the state evaluate to FALSE). Where two or more exit conditions with the same level of precedence become TRUE simultaneously, the choice as to which exit condition causes the state transition to take place is arbitrary.

In addition to the above notation, there are a couple of clarifications specific to this document. First, all boolean

variables are initialized to FALSE before the state machine execution begins. Second, the following notational shorthand is specific to this document:

<variable> = <expression1> | <expression2> | ...

Execution of a statement of this form will result in <variable> having a value of exactly one of the expressions. The logic for which of those expressions gets executed is outside of the state machine and could be environmental, configurable, or based on another state machine such as that of the method.

4. State Machine Symbols

( )
Used to force the precedence of operators in Boolean expressions and to delimit the argument(s) of actions within state boxes.

;
Used as a terminating delimiter for actions within state boxes. Where a state box contains multiple actions, the order of execution follows the normal English language conventions for reading text.

=
Assignment action. The value of the expression to the right of the operator is assigned to the variable to the left of the operator. Where this operator is used to define multiple assignments, e.g., a = b = X the action causes the value of the expression following the right-most assignment operator to be assigned to all of the variables that appear to the left of the right-most assignment operator.

!
Logical NOT operator.

&&
Logical AND operator.

||
Logical OR operator.

if...then...
Conditional action. If the Boolean expression following the if evaluates to TRUE, then the action following the then is executed.

{ statement 1, ... statement N }

Compound statement. Braces are used to group statements that are executed together as if they were a single statement.

!=
Inequality. Evaluates to TRUE if the expression to the left of the operator is not equal in value to the expression to the right.

==
Equality. Evaluates to TRUE if the expression to the left of the operator is equal in value to the expression to the right.

>
Greater than. Evaluates to TRUE if the value of the expression to the left of the operator is greater than the value of the expression to the right.

<=
Less than or equal to. Evaluates to TRUE if the value of the expression to the left of the operator is either less than or equal to the value of the expression to the right.

++
Increment the preceding integer operator by 1.

+
Arithmetic addition operator.

&
Bitwise AND operator.

5. Common Rules

Throughout the document we use terms defined in the [1], such as flow sender, flow receiver, QUERY, RESERVE or RESPONSE.

5.1 Common Procedures

```
tx_reserve():
   Transmit RESERVE message

tx_response():
   Transmit RESPONSE message

tx_query():
   Transmit QUERY message

tx_notify():
   Transmit NOTIFY message

install_qos_state():
   Install the local QoS state.

delete_qos_state():
   Delete the local QoS state.

send_info_to_app():
   Report information to the application.

RMF():
   Performs Resource Management Function and returns the following
   values{AVAIL, NO_AVAIL}.

is_local(RII):
   Checks the RII object of received RESPONSE message if it is
   requested by current node or other upstream node. Returns values
   {true, false}.

is_local(RSN):
   Checks The RSN object of the received RESPONSE message if it is
   requested by current node. Returns values {true, false}.

process_query():
   Processes a Query message and provides the requested info
```

5.2 Common Variables

```
RII:
```

Request Identification Information (RII) object.

RSN:
Reservation Sequence Number (RSN) object.

INFO:
Info_Spec object. Takes values:
- 0x02 - Success values
- 0x04 - Transient Failure values

QSPEC:
QoS specification object.

T-Flag:
Tear flag. Indicates to tear down reservation state. Takes values {true, false}.

Q-Flag:
Request Reduced Refreshes flag of common message header. Takes values {true, false}.

R-Flag:
Reserve-Init flag (QUERY) or Replace flag (RESERVE). Indicates a Receiver Initiated Reservation request in a QUERY message or an replacing RESERVE in a RESERVE message. Takes values {true, false}.

S-Flag:
Scoping flag of common message header. Takes values {true="Next_hop", false="Whole_path"}.

setRII:
If set a RII object will be included into the message. Takes values {true, false}.

ReducedRefresh:
Keeps information if Reduced refresh method may be used for refreshing a installed QoS state. Takes value {"On","Off"}.

FlowID:
Flow ID kept by the installed QoS state.

Nodepos:
Position of the QoS NSLP node. Takes values {"QNI", "QNE", "QNR"}.

Toggle:
Flag to indicate whether the direction of a new message has to be changed compared to the direction of a received one. Takes values

{true, false}.

Direction:
Direction, in which the message has to be sent. Takes values {DOWNSTREAM, UPSTREAM}.

SII:
Source Identification Information entry. Takes values:
- CurrSII - SII entry stored for current installed QoS state. (Assumed to be the one for the direction where the message comes from e.g.Upstream/Downstream)
- newSII - SII of the received message is different from the SII stored for the current installed QoS state.

5.3 Events

EV_TG_QUERY:
External trigger to send a QUERY message.

EV_RX_QUERY:
QUERY message received

EV_RX_NOTIFY:
NOTIFY message received

EV_TG_RESERVE:
External trigger to send a RESERVE message.

EV_RX_RESERVE:
RESERVE message received

EV_RX_RESPONSE:
RESPONSE message received

EV_TIMEOUT_RESPONSE:
Wait-Response interval timer expiration

EV_TIMEOUT_REFRESH:
Refresh interval timer expiration

EV_TIMEOUT_STATE_LIFETIME:
State lifetime timer expiration

5.4 Assumptions

- For simplification not all included objects in a message are shown. Only those that are significant for the case are shown. The State

Machine does not present handling of messages that are not significant for management of the states such as certain NOTIFY and QUERY messages.

- The State Machine represents handling of messages of the same Session ID and with no protocol errors. Separate parallel instances of the state machines should handle messages for different Session IDs.
- Default message handling should be defined for messages with different Session IDs that have impact on current session state and error messages. This is not included in the current version.

6 Basic State Machine Concept

6.1 The QoS NSLP Daemon

The QoS NSLP Daemon (qosd) listens for incoming messages from the local application and for messages coming over the network from GIST. For each new SessionID (SID) a new State Machine (FSM) is created as shown in the diagram below.

Incoming messages from the client application are checked for the type of the message (QUERY or RESERVE) and for the SID. A table is searched for the given SID. If it is not found, a new FSM is created and its address together with the SID is added to the table. If the type of the message is a QUERY with set R-Bit, a Receiver Initiated Reservation is requested and the node position is QNR, in all other cases the node position is QNI. If the SID is found, the address of the FSM is returned. Now the FSM is triggered with the corresponding event of the message (EV_TG_QUERY, EV_TG_RESERVE). If the FSM returns to the qosd in ST_IDLE, it is deleted together with its table entry.

For a message arriving from GIST, the procedure is almost the same. The table is searched for the given SID. If it is not found, then the IP-address of the MRI is compared to the local IP-address. If the arriving message is requesting a Receiver Initiated Reservation and the destination address is equal to the local address then the node position is QNR. If the addresses are equal and no Receiver Initiated Reservation is requested then the node position is QNI. If the addresses are not the same, a new FSM for a QNE is created. Also here, the corresponding FSM is triggered with the event according to the arrived message and deleted when returning in ST_IDLE.

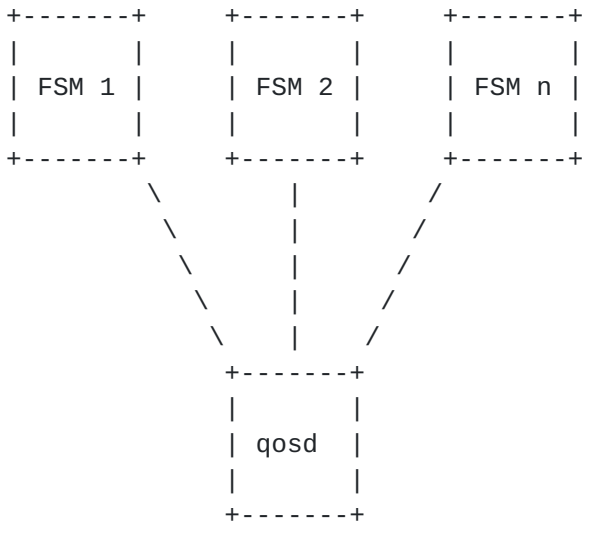

Figure 1

6.2 States, Events and Callback Functions

Three States are defined: ST_IDLE, ST_WR and ST_INST. A new created FSM is starting automatically in ST_IDLE. In this state no reservation state is installed no responses to previously sent messages are expected. In ST_WR the FSM is waiting for a response to a previously sent message, but no reservation state is installed. In ST_INST reservation state has been installed and incoming messages are processed.

The following table provides to a given state and a triggered event the function which has to be executed by the FSM. Example: when EV_RX_QUERY is triggered in state ST_IDLE, the function idle__rx_query is executed.

State	Event	Executed Function
ST_IDLE	EV_RX_QUERY	idle__rx_query
ST_IDLE	EV_RX_NOTIFY	idle__rx_notify
ST_IDLE	EV_RX_RESERVE	idle__rx_reserve
ST_IDLE	EV_RX_RESPONSE	idle__rx_response
ST_IDLE	EV_RX_RMF_MSG	idle__rx_rmf_msg
ST_WR	EV_RX_QUERY	wr__rx_query
ST_WR	EV_RX_NOTIFY	wr__rx_notify
ST_WR	EV_RX_RESERVE	wr__rx_reserve
ST_WR	EV_RX_RESPONSE	wr__rx_response
ST_WR	EV_RX_RMF_MSG	wr__rx_rmf_msg
ST_WR	EV_TIMEOUT_WAITRESP	wr__timeout_response
ST_WR	EV_TIMEOUT_STATELIFETIME	wr__timeout_statelifetime
ST_INST	EV_RX_QUERY	inst__rx_query
ST_INST	EV_RX_NOTIFY	inst__rx_notify
ST_INST	EV_RX_RESERVE	inst__rx_reserve
ST_INST	EV_RX_RESPONSE	inst__rx_response
ST_INST	EV_RX_RMF_MSG	inst__rx_rmf_msg
ST_INST	EV_TIMEOUT_WAITRESP	inst__timeout_response
ST_INST	EV_TIMEOUT_REFRESH	inst__timeout_refresh
ST_INST	EV_TIMEOUT_STATELIFETIME	inst__timeout_statelifetime
ST_INST	EV_SII_CHANGED	inst__sii_changed
ST_INST	EV_NETWORK_NOTIFICATION	inst__network_notification

Figure 2

6.3 Timer

The Response Timer at QNI and QNE is started when a RESPONSE message is expected to a sent QUERY or RESERVE. When a reservation is set up, the Refresh Timer are started at QNI and QNE and the StateLife Timer are started at QNE and QNR. When the Refresh Timer times out, a refreshing RESERVE is sent peer to peer towards the QNR and the Response Timer are started because a confirmation is expected. On arrival the StateLife Timer is restarted.

If the confirmation is not sent back, then the refreshing RESERVEs are resent up to MAX_RETRY. After MAX_RETRY has been reached, reservation state is removed and a RESERVE with set T-Flag is sent to the QNR to remove reservation state along the path. When no refreshing RESERVE arrive at QNE and QNR, then the StateLife Timer expires and reservation state is also removed and a RESERVE with set T-Flag is sent towards the QNR.

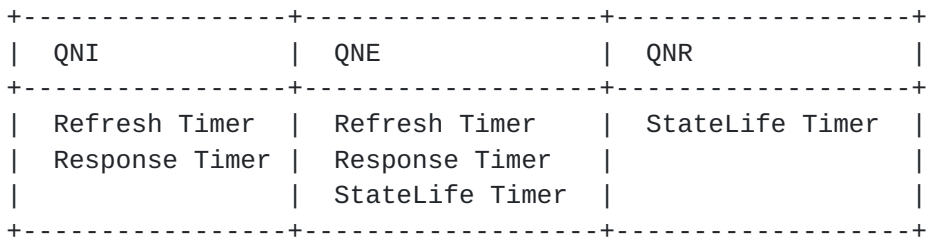

Figure 3

6.4 The Toggle Flag

The Toggle Flag manipulates the direction of the message. When set, the message is sent in the opposite direction compared to the received one. The boolean operation XOR is used. Example, where true is the toggle flag: DOWNSTREAM XOR true = UPSTREAM (0 XOR 1 = 1).

7. State machine

The following section presents the state machine diagrams of QoS NSLP. Please note that stateless nodes do not perform transitions to other states than ST_IDLE. A C-like pseudo code example for update_rsn() and for send_rmf() is presented in Appendix A.4. The send_rmf function is located in the RMF code and update_rsn() is part of the FSM code.

7.1 State ST_IDLE

(see the .pdf version for missing diagram or
refer to Appendix A.1 if reading the .txt version)

Figure 4

(see the .pdf version for missing diagram or
refer to Appendix A.1 if reading the .txt version)

Figure 5

7.2 State ST_WR

(see the .pdf version for missing diagram or
refer to Appendix A.2 if reading the .txt version)

Figure 6

(see the .pdf version for missing diagram or
refer to Appendix A.2 if reading the .txt version)

Figure 7

(see the .pdf version for missing diagram or
refer to Appendix A.2 if reading the .txt version)

Figure 8

7.3 State ST_INST

(see the .pdf version for missing diagram or
refer to Appendix A.3 if reading the .txt version)

Figure 9

(see the .pdf version for missing diagram or
refer to Appendix A.3 if reading the .txt version)

Figure 10

(see the .pdf version for missing diagram or
refer to Appendix A.3 if reading the .txt version)

Figure 11

(see the .pdf version for missing diagram or
refer to Appendix A.3 if reading the .txt version)

Figure 12

8. Actions and Transitions

This chapter describes the operation of the FSM.

8.1 State ST_IDLE

8.1.1 idle__rx_query

This function is executed when a QUERY message has arrived over the network from GIST and the FSM is in State ST_IDLE. If a receiver initiated reservation has been requested and the node position is QNI, Direction is set to upstream and the message is passed to the RMF. Transition is made to ST_WR. In all other cases a resource query has been received, the message is passed to the RMF and transition is done to ST_WR.

8.1.2 idle__rx_notify

A NOTIFY message has been received. If the node position is not the final destination for this message, it is passed further along the path.

8.1.3 idle__rx_reserve

A RESERVE from the network in ST_IDLE has arrived. If the T-Flag is set, then the message is passed to the RMF to tear down the reservation. Transition is made to ST_IDLE.

In the other case where a reservation request has arrived, Direction is set to UPSTREAM at QNR's. On statefull nodes the state timer is started. In any case the message is passed to the RMF and transition is made to ST_WR.

8.1.4 idle__rx_response

In ST_IDLE the node is not expecting a RESPONSE message. It is forwarded further to the final destination. The state ST_IDLE is kept.

8.1.5 idle__rx_rmf_msg

If the RMF is triggering a RESPONSE message and the containing INFO_SPEC object indicates either a reservation failure or that the teardown was successful the message is passed further and transition is done to ST_IDLE.

If a QUERY message has been triggered, it is checked whether the R-Flag is set or an RII object is present. In this case the response timer is started and transition is made to ST_WR. If a normal QUERY containing no R-Flag and no RII object has been triggered, no response timer is started and transition is made to ST_IDLE.

In case the RMF has triggered a RESERVE message at statefull intermediate nodes the state timer is started. If it contains no RII object, then no response is triggered and the refresh timer is started immediately to frequently refresh the session. If an RII object is included in the message the response timer is started instead of the refresh timer. The message is sent further along the path.

At the QNI no state timer is started as this is the node which takes care of the refreshes. It does not receive any refreshes which would restart the state timer. The message is sent further along the path.

The QNR is starting only the state timer. As this is the end node it does not propagate the message any further. No responses are expected and no refresh messages will be sent.

Transition is made to ST_INST.

8.2 State ST_WR

8.2.1 wr__rx_query

A QUERY message has arrived over the network from GIST and the FSM is in State ST_WR. The message is entirely passed to the RMF using sendrmf(). Transition is made to ST_WR.

8.2.2 wr__rx_notify

If the node is not the final destination of the NOTIFY message it is sent further along the path and transition is made to ST_WR.

8.2.3 wr__rx_reserve

If a reserve message with set T-Flag has been received, the message is passed to the RMF to tear down the reservation and to construct a new tearing reserve message. If no confirmation of state removal has been requested, i.e. no RII object is present, transition is done to ST_IDLE. QNR's and stateless nodes also go directly to ST_IDLE.

If the RII object in the tearing reserve is present, statefull nodes except QNR's start the state timer for one refresh period and pass the message to the RMF to remove reservation state. Transition is

done to ST_WR .

If the received message is requesting a reservation the state timer is started. The message is passed to the RMF to install reservation state and transition is made to ST_WR. If the node position is QNR then this is the requested RESERVE message for the Receiver Initiated Reservation. The Response Timer is stopped. The state ST_WR is kept.

8.2.4 wr__rx_response

In ST_WR a RESPONSE message has arrived. If the node is configured as QNE_Egress the message is forwarded with the BYPASS_STATELESS_ID as nslp_id that stateless nodes do not intercept this message. If an Ingress node receives a response with the BYPASS_STATELESS_ID, the nslp_id is translated to the QOS_NSLP_ID and the message is forwarded up to the requesting node. Normal QNE's just forward the response message. At this point no state transition is done. The relevant checkings follow in the next paragraph.

If the received RII is from the local node the response timer is stopped. If there are no more outstanding RII's, either from the local node or from foreign nodes transition is done to ST_IDLE otherwise ST_WR. If the received RII was from a foreign node the same transitions are done depending on the outstanding RII's. For a stored foreign RII no response timer was started and therefore no response timer has to be stopped.

8.2.5 wr__rx_rmf_msg

See the section 8.1.5 where a RESERVE message has been triggered. Transition is made to ST_INST.

8.2.6 wr__timeout_response

A Response Timer has timed out while in ST_WR. If the maximum number of retries has been reached, transition is done to ST_IDLE, if no more response timers are pending, otherwise transition is done to ST_WR. If the maximum number of retransmissions has not been reached, then the query message is resent. The response timer is restarted and transition is made to ST_WR.

8.2.7 wr__timeout_statelifetime

If the state timer in ST_WR times out the confirmation message for state removal has not been sent. In this case no retransmissions of the message is done and transition is done to ST_IDLE after stopping all timers.

8.2.8 wr__rx_rmf_msg

In ST_WR any triggered message from the RMF is forwarded and the same state is kept.

8.3 State ST_INST

8.3.1 inst__rx_query

See wr__rx_query. Transition is made to ST_INST.

8.3.2 inst__rx_notify

Reduced Refreshes are the default for refreshing RESERVEs. Notify messages are checked whether the next peer accepts reduced refreshes or not. The state ST_INST is kept.

8.3.3 inst__rx_reserve

All incoming RESERVEs are checked for their RSN value. if it is smaller than the stored peer RSN the message is rejected. If the value is the same to the stored one the reservation is refreshed. Larger RSN value modify the reservation state.

A tearing reserve is passed to the RMF to remove the existing reservation state. If no confirmation is requested, i.e. no RII object is present, transition is done to ST_IDLE. If a confirmation is requested, the state timer is started for one refresh interval and transition is done to ST_WR.

If no T_Flag is set the message is passed to the RMF to install or modify the reservation if the RSN value is larger than the stored peer RSN. The state timer is restarted for equal and larger RSN values. The state ST_INST is kept.

8.3.4 inst__rx_response

In ST_INST a RESPONSE message has arrived. If the node position is QNE and no Scoping Flag is set, then the RESPONSE message is forwarded. If the RII value matches, then the Response Timer is stopped. If the received Error Code of the INFO object is SUCCESS, then the Response Timer is started. Transition is done to ST_INST.

If the Error Code is FAILURE, then the reservation state is deleted and pending timers are stopped. Transition is made to ST_IDLE.

8.3.5 inst__timeout_response

A Response Timer has timed out while in ST_INST. If the maximum number of retries has been reached, then all pending timer are stopped and the RMF is informed to remove existing reservation state. Transition is made to ST_IDLE. If maximum numbers of retransmissions has not been reached, then previously sent message is resent. The Response Timer is restarted and the state ST_INST is kept.

8.3.6 inst__timeout_refresh

A refreshing RESERVE message is created. If Reduced Refreshes are accepted, no QSPEC object is added. The refresh timer is being restarted. Transition is made to ST_INST.

8.3.7 inst__timeout_statelifetime

The Statelife Timer timed out in ST_INST. All active timer are stopped and existing reservations are removed. The RMF is informed to teardown reservation state and to trigger a teardown message. Transition is done to ST_IDLE.

8.3.8 inst__sii_changed

A different SII-Handle indicates that a route change has occurred. A full reserve including QSPEC is sent on the new path. The reservation on the old path is torn down using the old SII-Handle. The state ST_INST is kept.

8.3.9 inst__network_notification

A network notification from GIST is indicating that a route change has occurred. A full RESERVE including QSPEC is sent on the new path. The state ST_INST is kept.

8.3.10 inst__rx_rmf_msg

See idle__rx_rmf_msg for details.

9. Security Considerations

This document does not raise new security considerations. Any security concerns with QoS NSLP are likely reflected in security related NSIS work already (such as [1] or [6]).

For the time being, the state machine described in this document does not consider the security aspect of QoS NSLP protocol itself.

10. Open Issues

This document tries to describe possible states and transitions for QoS NSLP according to its current specification [1], Section 5. We found some issues during the development of the state machines.

1. Bi-directional reservation is difficult to support as the state machine becomes quite complex (note at one particular point in time the protocol state engine can be only in one state).
2. How to signal unsuccessful reservation for Receiver initiated reservation (No RII included; a resulting Response(RSN) cannot be forwarded further than the next peer). We use NOTIFY message.
3. The case of unsuccessful reservation at a QNE node and no RII specified by upstream nodes. According to the spec RESPONSE(RSN) should not be forwarded further than the next peer. Currently we use NOTIFY(RSN) that is sent further to the upstream nodes.
4. We assume that handling of QoS state lifetime expiration event is based on the local policy of the node. NOTIFY/Reserve(Ton) messages might be sent to other peers.
5. The draft states that RESERVE message MUST be sent only towards the QNR. This is not the case when re-routing procedure is done and RESERVE(Ton) message should be sent from merging QNE node for deleting the old branch. We believe this is towards the QNI.
6. Re-routing functionality described in this document is not complete and need further consideration.

11. Change History

11.1 Changes in Version -01

1. Notation of the nodes changed to QNI, QNE and QNR.
2. Description of soft state refresh functionality.
3. Support of ACK flag in the common header.
4. Include of QoS NSLP objects, flags from the common header and entries stored with the installed QoS state in a node: ACK, Replace, RSN, Error_SPEC, QSPEC, FlowID, SII.
5. Initial description of Re-routing functionality.
6. For support of all listed changes, some notations are changed.

11.2 Changes in Version -02

1. Switch to .pdf format of the draft and include graphic diagrams.
2. Update notation from "Summary refresh" to "Reduced refresh"
3. Description of QoS reservation update/upgrade

11.3 Changes in Version -03

1. Deep review of the state machine architecture

11.4 Changes in Version -04

1. Reduced the three state machines of QNI, QNE and QNR to one for all nodes.
2. Introduced new flags to have a finer control of the direction of the message to be sent.

11.5 Changes in Version -05

1. Combined ST_WR2 and ST_INST into ST_INST
2. Support for Q-Flag in common header
3. Explanations on the execution of the State Machine added

11.6 Changes in Version -06

1. Interaction with RMF added
2. Trigger from local application removed

12. Acknowledgments

The authors would like to thank Sven Van den Bosch and Christian Dickmann for their feedback.

13. References

13.1. Normative References

[1] Manner, J., Karagiannis, G. and McDonald, A., "NSLP for Quality-of-Service Signaling", Internet draft, draft-ietf-nsis-qos-nslp-15 (work in progress), July 2007.

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

13.2. Informative References

[3] Vollbrecht, J., Eronen, P., Petroni, N., and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator", RFC 4137, August 2005.

[4] Institute of Electrical and Electronics Engineers, "DRAFT Standard for Local and Metropolitan Area Networks: Port-Based Network Access Control (Revision)", IEEE 802-1X-REV/D11, July 2004.

[5] Ohba, Y., "State Machines for Protocol for Carrying Authentication for Network Access (PANA)", draft-ohba-pana-statemachine-01 (work in progress), February 2005.

[6] Tschofenig, H. and D. Kroeselberg, "Security Threats for NSIS", RFC 4081, June 2005.

Appendix A. ASCII versions of state diagrams

This appendix contains the state diagrams in ASCII format. Please use the PDF version whenever possible: it is much easier to understand.

The notation is as follows: for each state there is a separate table that lists in each row:
- an event that triggers a transition,
- actions taken as a result of the incoming event,
- and the new state at which the transitions ends.

A.1. State ST_IDLE

ST_IDLE::EV_RX_QUERY

Action	new State

if(!(R-Flag && QNI)) {
  sendrmf();
}
	ST_WR

if(R-Flag && QNI) {
  Direction=UPSTREAM;
  sendrmf();
}
	ST_WR

Figure 13

ST_IDLE::EV_RX_NOTIFY

Action	new State
if(!QNI) tx_notify(Toggle=false);	ST_IDLE

Figure 14

ST_IDLE::EV_RX_RESERVE

Action	new State
if(T-Flag) sendrmf();	ST_IDLE

if(!T-Flag) {
  if(QNR) {
  Direction=UPSTREAM;
  startStateTimer();
  sendrmf();
}
	ST_WR

Figure 15

ST_IDLE::EV_RX_RESPONSE

Action	new State
if(!QNI) tx_response(Toggle=false);	ST_IDLE

Figure 16

ST_IDLE::EV_RX_RMF_MSG

Action	new State
if(rx_response && !QNI) tx_response(Toggle=false);	ST_IDLE

if(rx_query && (R-Flag || RII)) {
  startResponseTimer();
}
	ST_WR

```
| if(rx_reserve && !T-Flag)                                  |  ST_INST  |
|    if(QNE || QNE_Ingress || QNE_Egress) {                  |           |
|       startStateTimer();                                   |           |
|       if(RII) startResponseTimer();                        |           |
|       else startRefreshTimer();                            |           |
|       tx_reserve();                                        |           |
|    } else if(QNI) {                                        |           |
|       if(RII) startResponseTimer();                        |           |
|       else startRefreshTimer();                            |           |
|       tx_reserve();                                        |           |
|    } else if(QNR) {                                        |           |
|       startStateTimer();                                   |           |
|    }                                                       |           |
| }                                                          |           |
|                                                            |           |
+------------------------------------------------------------+-----------+
```

Figure 17

A.2. State ST_WR

```
ST_WR::EV_RX_QUERY
+------------------------------------------------------------+-----------+
| Action                                                     | new State |
+------------------------------------------------------------+-----------+
|                                                            |           |
| sendrmf();                                                 |  ST_WR    |
|                                                            |           |
+------------------------------------------------------------+-----------+
```

Figure 18

```
ST_WR::EV_RX_NOTIFY
+------------------------------------------------------------+-----------+
| Action                                                     | new State |
+------------------------------------------------------------+-----------+
|                                                            |           |
| if(!QNI) tx_notify(Toggle=false);                          |  ST_WR    |
|                                                            |           |
+------------------------------------------------------------+-----------+
```

Figure 19

ST_WR::EV_RX_RESERVE

Action	new State

if(T-Flag && (!RII || QNR || STATELESS)) {
  send_rmf();
}
	ST_IDLE

if(T-Flag && RII && (STATEFULL && !QNR)) {
  startStateTimer(REFRESH_PERIOD);
  send_rmf();
}
	ST_WR

if(!T-Flag && QNR) {
  if(QNR) stopResponseTimer();
  startStateTimer();
  sendrmf();
}
	ST_WR

Figure 20

ST_WR::EV_RX_RESPONSE

Action	new State

if(QNE_Egress) {
  tx_response(Toggle=false, BYPASS_STATELESS_ID);
} else if(QNE ||
  (QNE_Ingress && nslp_id==BYPASS_STATELESS_ID)) {
  tx_response(Toggle=false, QOS_NSLP_ID);
}	

if(is_local(RII)==true && !outstanding_foreign_rii) {
  stopResponseTimer();
}
	ST_IDLE

if(is_local(RII)==true && outstanding_foreign_rii) {
  stopResponseTimer();
}
	ST_WR

if(is_foreign(RII)==true && !outstanding_local_rii)	ST_IDLE
if(is_foreign(RII)==true && outstanding_local_rii)	ST_WR

Figure 21

ST_WR::EV_TIMEOUT_RESPONSE

Action	new State
if(MAX_RETRY && !TIMER_PENDING)	ST_IDLE
if(MAX_RETRY && TIMER_PENDING)	ST_WR

if(!MAX_RETRY) {
  tx_query(Direction);
  restartResponseTimer();
}
	ST_WR

Figure 22

ST_WR::EV_TIMEOUT_STATELIFETIME

Action	new State
stopTimers();	ST_IDLE

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

Figure 23

```
ST_WR::EV_RX_RMF_MSG
+----------------------------------------------------------+-----------+
| Action                                                   | new State |
+----------------------------------------------------------+-----------+
|                                                          |           |
| if(rx_reserve && !T-Flag)                                |  ST_INST  |
|    if(QNE || QNE_Ingress || QNE_Egress) {                |           |
|       startStateTimer();                                 |           |
|       if(RII) startResponseTimer();                      |           |
|       else startRefreshTimer();                          |           |
|       tx_reserve();                                      |           |
|    } else if(QNI) {                                      |           |
|       if(RII) startResponseTimer();                      |           |
|       else startRefreshTimer();                          |           |
|       tx_reserve();                                      |           |
|    } else if(QNR) {                                      |           |
|       startStateTimer();                                 |           |
|    }                                                     |           |
| }                                                        |           |
|                                                          |           |
+----------------------------------------------------------+-----------+
```

Figure 24

A.3. State ST_INST

```
ST_INST::EV_RX_QUERY
+----------------------------------------------------------+-----------+
| Action                                                   | new State |
+----------------------------------------------------------+-----------+
|                                                          |           |
| sendrmf();                                               |  ST_INST  |
|                                                          |           |
+----------------------------------------------------------+-----------+
```

Figure 25

```
ST_INST::EV_RX_NOTIFY
+----------------------------------------------------------+-----------+
```

Action	new State

if(ErrorClass==ERRORCLASS_INFO) {
  if(ErrorCode==RR_NOT_SUPPORTED) {
  ReducedRefreshes = false;
  } else if(ErrorCode==RR_SUPPORTED) {
  ReducedRefreshes = true;
  }
}
	ST_INST

Figure 26

ST_INST::EV_RX_RESERVE

Action	new State

if(T-Flag && !RII) {
  if(rsn>peer_rsn) {
  update_rsn();
  sendrmf();
  }
}
	ST_IDLE

if(T-Flag && RII) {
  if(rsn>peer_rsn) {
  update_rsn();
  sendrmf();
  start_statetimer(REFRESH_RELOAD);
  }
}
	ST_WR

if(!T-Flag) {
  if(rsn>=peer_rsn) {
  if(rsn>peer_rsn) {
  update_rsn();
  sendrmf();
  }
  restart_statetimer();
  }
	ST_INST

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

Figure 27

ST_INST::EV_RX_RESPONSE

```
+------------------------------------------------------------+-----------+
| Action                                                     | new State |
+------------------------------------------------------------+-----------+
|                                                            |           |
| if(QNE_Egress) {                                           |           |
|    tx_response(toggle=false, BYPASS_STATELESS_ID);         |           |
| } else if(QNE ||                                           |           |
|       (QNE_Ingress && nslp_id==BYPASS_STATELESS_ID)) {     |           |
|    tx_response(toggle=false, QOS_NSLP_ID);                 |           |
| }                                                          |           |
| if(is_local_rii) stopResponseTimer();                      |           |
| startRefreshTimer();                                       |           |
|                                                            |           |
+------------------------------------------------------------+-----------+
|                                                            |           |
| if(INFO==SUCCESS)                                          |  ST_INST  |
|                                                            |           |
+------------------------------------------------------------+-----------+
|                                                            |           |
| if(INFO==FAILURE) {                                        |  ST_IDLE  |
|    sendrmf();                                              |           |
|    stop_timers();                                          |           |
| }                                                          |           |
|                                                            |           |
+------------------------------------------------------------+-----------+
```

Figure 28

ST_INST::EV_RX_RMF_MSG

```
+------------------------------------------------------------+-----------+
| Action                                                     | new State |
+------------------------------------------------------------+-----------+
|                                                            |           |
| if(rx_response && (fail || tear_successful)) {             |  ST_IDLE  |
|    tx_response();                                          |           |
| }                                                          |           |
|                                                            |           |
+------------------------------------------------------------+-----------+
```

if(rx_response && success) {
  tx_response();
}
	ST_INST

if(rx_reserve && !T-Flag)
  if(QNE || QNE_Ingress || QNE_Egress) {
  startStateTimer();
  if(RII) startResponseTimer();
  else startRefreshTimer();
  tx_reserve();
  } else if(QNI) {
  if(RII) startResponseTimer();
  else startRefreshTimer();
  tx_reserve();
  } else if(QNR) {
  startStateTimer();
  }
}
	ST_INST

Figure 29

ST_INST::EV_TIMEOUT_RESPONSE

Action	new State

if(MAX_RETRY) {
  sendrmf();
  stopTimers();
}
	ST_IDLE

if(!MAX_RETRY) {
  tx_reserve(Direction);
  restartResponseTimer()
}
	ST_INST

Figure 30

ST_INST::EV_TIMEOUT_REFRESH

Action	new State

if(ReducedRefreshes) {
  tx_reserve(RSN, Direction);
} else {
  tx_reserve(RSN, QSPEC, Direction);
}
restartRefreshTimer();
	ST_INST

Figure 31

ST_INST::EV_TIMEOUT_STATELIFETIME

Action	new State
stopTimers(); sendrmf();	ST_IDLE

Figure 32

ST_INST::EV_SII_CHANGED

Action	new State
tx_reserve(RII, RSN, QSPEC, SII);	ST_INST

Figure 33

ST_INST::EV_NETWORK_NOTIFICATION

Action	new State
tx_reserve(RII, RSN, QSPEC);	ST_INST

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

Figure 34

A.4 Commonly used functions:

A.4.1 update_rsn()

```
 1 void update_rsn() {
 2    if((STATEFULL && !QNE_Egress) ||
 3       (QNE_Egress && nslp_id==BYPASS_STATELESS_ID)) {
 4       peer_rsn=rsn;
 5    }
 6 }
 7
```

Figure 35

This function is used to update the rsn value only on statefull peers. At the egress node the rsn is updated when a message arrives with the nslp_id BYPASS_STATELESS_ID.

A.4.2 send_rmf()

```
 1 void sendrmf() {
 2
 3    switch(message_type) {
 4
 5       case QOS_MESSAGETYPE_RESERVE:
 6
 7          switch(nodepos) {
 8
 9             case QNI:
10
11                if(T-Flag) {
12                   free_qos();
13                   recv_rmf(reserve, DO_NOT_TOGGLE);
14
15                } else {
16                   qos_available = reserve_qos();
17                   if(qos_available) recv_rmf(reserve, DO_NOT_TOGGLE);
18                   else recv_rmf(response, INFOSPEC=FAIL, DO_TOGGLE);
```

```
 19                }
 20                break;
 21
 22             case QNE:
 23
 24                if(T-Flag) {
 25                   free_qos();
 26                   if(!S-Flag) recv_rmf(reserve, new_rsn, DO_NOT_TOGGLE);
 27                   else if(rii) recv_rmf(response, INFOSPEC=FAIL,
DO_TOGGLE);
 28
 29                } else {
 30                   qos_available = reserve_qos();
 31                   if(!qos_available) recv_rmf(response, INFOSPEC=FAIL,
DO_TOGGLE);
 32                   else if(!S-Flag) recv_rmf(reserve, DO_NOT_TOGGLE);
 33                   else recv_rmf(response, INFOSPEC=SUCCESS, DO_TOGGLE);
 34                   if(rr_supported) recv_rmf(notify, DO_TOGGLE);
 35                }
 36                break;
 37
 38             case QNE_Ingress:
 39
 40                if(T-Flag) {
 41                   free_qos();
 42                   recv_rmf(reserve, BYPASS_STATELESS_ID, DO_NOT_TOGGLE);
 43                   recv_rmf(reserve, new_rsn, new_rii, DO_NOT_TOGGLE);
 44                } else {
 45                   qos_available = reserve_qos();
 46                   if(!qos_available) recv_rmf(response, INFOSPEC=FAIL,
DO_TOGGLE);
 47                   else {
 48                      recv_rmf(reserve, BYPASS_STATELESS_ID, DO_NOT_TOGGLE);
 49                      recv_rmf(reserve, DO_NOT_TOGGLE);
 50                   }
 51                   if(rr_supported) recv_rmf, notify(DO_TOGGLE);
 52                }
 53                break;
 54
 55             case QNE_Interior:
 56
 57                if(T-Flag) {
 58                   free_qos();
 59                   recv_rmf(reserve(new_rsn, DO_NOT_TOGGLE));
 60
 61                } else {
 62                   qos_available = reserve_qos();
 63                   if(!qos_available) recv_rmf(response, INFOSPEC=FAIL,
```

```
DO_TOGGLE);
 64                    else recv_rmf(reserve, DO_NOT_TOGGLE);
 65                 }
 66                 break;
```

```
 67
 68             case QNE_Egress:
 69
 70                if(T-Flag) {
 71                   free_qos();
 72                   if(nslp_id==QOS_NSLP_ID && rii)
 73                      recv_rmf(response(new_rsn, new_rii, DO_NOT_TOGGLE));
 74                } else {
 75                   qos_available = reserve_qos();
 76                   if(!qos_available) recv_rmf(response, INFOSPEC=FAIL,
DO_TOGGLE);
 77                   if (nslp_id==QOS_NSLP_ID) {
 78                       if(rii) recv_rmf(response, INFOSPEC=SUCCESS,
QOS_NSLP_ID);
 79                       if(rr_supported) recv_rmf(notify, QOS_NSLP_ID);
 80                   } else if (nslp_id==BYPASS_STATELESS_NODE &&
rr_supported)
 81                       recv_rmf(notify, BYPASS_STATELESS_NODE);
 82                }
 83                break;
 84
 85             case QNR:
 86
 87                if(T-Flag) {
 88                   free_qos();
 89                   if(rii) recv_rmf(response(DO_TOGGLE));
 90                } else {
 91                   qos_available = reserve_qos();
 92                   if(!qos_available) recv_rmf(response, INFOSPEC=FAIL,
DO_TOGGLE);
 93                   if(rii) recv_rmf(response, INFOSPEC=SUCCESS, DO_TOGGLE);
 94                   if(rr_supported) recv_rmf(notify, INFOSPEC=RR_SUPPORTED,
DO_TOGGLE);
 95                }
 96                break;
 97
 98          }
 99          break;
100
101       case QOS_MESSAGETYPE_QUERY:
102
103          if (QNI && R-Flag) {
104             reserve_qos();
105             send_rmf(reserve, new_rii, new_rsn, new_qspec);
106
107          } else if (QNR && R-Flag) {
108
109             query_rmf();
```

```
110             send_rmf(query, R-Flag, PC, QSPEC, Direction=DOWNSTREAM);
111
112         } else {
113
114             query_rmf();
```

```
115             if(QNR) send_rmf(response);
116             else send_rmf(query);
117
118          }
119          break;
120
121       case QOS_MESSAGETYPE_RESPONSE:
122
123          break;
124
125       case QOS_MESSAGETYPE_NOTIFY:
126
127          break;
128
129    }
130
131 }
132
```

Figure 36

Message Type Reserve

At the QNI if the message has set the T-Flag the reservation is torn down and a new reserve message is triggered using the API-call recv_rmf(). If no T-Flag was set in the reserve message, reservation state is installed. If successful a new reserve message is triggered to be sent further along the path.

At QNE's the (tearing) reserve is sent further if the scoping flag is not set. Otherwise this node triggers a response if an RII object is present. If a reservation request fails a response is triggered to inform the initiator about the failure. If reduced refreshes are supported a notify message is sent back to the previous peer.

Ingress nodes trigger two new (tearing) reserve message, one with the old nslp_id and one with the BYPASS_STATELESS_ID to bypass stateless nodes. No special treatment of the messages is done at interior nodes.

The egress node triggers a response message, if a response was requested by including an RII object and if the nslp_id of the received message is QOS_NSLP_ID. If a requested reservation fails, a response is sent back in any case. Notify message for reduced refreshes are sent with the corresponding nslp_id.

The QNR as the destination of the message echoes a response if the reservation fails or if a response was requested by including an RII object. Also here a notify message is sent back to the previous node, if reduced refreshes are accepted.

Message Type QUERY

If a QUERY message with set R-Flag arrives at the QNI, the RMF triggers a new RESERVE message to be sent towards the QNR. Resource query messages are sent further after updating the QSPEC. The QNR triggers a response message containing the QSPEC.

Authors' Addresses

Xiaoming Fu
University of Goettingen
Telematics Group
Lotzestrasse 16-18
Goettingen 37083
Germany

Email: fu@cs.uni-goettingen.de

Hannes Tschofenig
Nokia Siemens Networks
Otto-Hahn-Ring 6
Munich, Bayern 81739
Germany

Email: Hannes.Tschofenig@nsn.com
URI: http://www.tschofenig.com

Tseno Tsenov
Nokia Siemens Networks
Otto-Hahn-Ring 6
Munich, Bayern 81739
Germany

Email: tseno.tsenov@mytum.de

Bernd Schloer
University of Goettingen
Telematics Group
Lotzestrasse 16-18
Goettingen 37083
Germany

Email: bschloer@cs.uni-goettingen.de

Full Copyright Statement

Intellectual Property

Acknowledgement

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