NSIS
Internet-Draft
Expires: May 11, 2005

X. Fu
Univ. Goettingen
H. Tschofenig
T. Tsenov
Siemens

November 10, 2004

QoS NSLP State Machine
draft-fu-nsis-qos-nslp-statemachine-00.txt

Status of this Memo

Copyright Notice

Abstract

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

Table of Contents

1. Introduction

This document describes the state machines for QoS NSLP [1], trying to show how QoS NSLP can be implemented to support its deployment. The state machines described in this document are illustrative of how the QoS NSLP protocol defined in [1] may be implemented for the first QoS NSLP node in the flow path, intermediate QoS NSLP nodes, and the last QoS NSLP node in the flow path. Where there are differences [1] are authoritative. The state machines 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

(which need to be incorporated into use scenarios when describing state machine. Note they are represented by way of certain objects/flags in Reserve and Query messages.)

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

We describe a set of state machines for different roles of entities running QoS NSLP to illustrate how QoS NSLP may be implemented.

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 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.

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(Toff): Transmit RESERVE message with 'Teardown' bit off
tx_RESERVE(Ton): Transmit RESERVE message with 'Teardown' bit on
tx_RESPONSE(): Transmit RESPONSE message
tx_QUERY(w/RII): Transmit QUERY message with Request Identification Information (RII) object
tx_QUERY(w/oRII): Transmit QUERY message without RII object
rx_RESPONSE(): Receive RESPONSE message
rx_QUERY(): Receive QUERY message
rx_RESERVE(): Receive RESERVE message
TIMEOUT_State: State lifetime timer expiration
TIMEOUT_Refresh: Refresh interval timer expiration
tg_QUERY: External trigger to send a QUERY message (typically triggered by the application).
tg_RESERVE: External trigger to send a RESERVE message.
tg_TEARDOWN: External trigger to clear previously established QoS state (typically triggered by the application). It is translated to a tx_RESERVE(Ton) message.
Set QoS state: establish the local QoS state.
Refresh QoS state: refresh the local QoS state.
Clear QoS state: delete the local QoS state.
Send info to Application: report information to the application.
RMF: Performs Resource Management Function and returns the following values{AVAIL, NO_AVAIL}.
SetRII: Sets the RII object of the messages e.g. the node requests explicit response to the message being sent. Returns values {0,1}.
CheckRII: Checks the RII object of received RESPONSE message if it is requested by current node or other upstream node. Returns values {LOCAL, NO_LOCAL}.
Result: Processes the information of the RESPONSE messages and provides information. It tells whether the reservation is successful or not (if it is a response to a reserve message), or the information carried in the response message (if it is a response to a query message), or an error has occurred. Returns values {INFO, SUCCESSFUL, ERROR}.

5.2 Common Variables

RII: Request Identification Information (RII) object. Logical variable representing if the RII is set or not. Takes values {0,1}.
SCOPING: Scoping flag of common message header. Takes values {"Next_hop","Whole_path"}.

5.3 Constants

6. State machine for first QoS NSLP node in the flow path

State: INIT

Condition	Action	State
UCT	initialize variables	IDLE

State: IDLE

Condition	Action	State	Note
(rx_QUERY) && (!RII) && (RMF="NO_AVAIL")	tx_NOTIFY(ERROR)	IDLE	1) 2)
(rx_QUERY) && (RII)	tx_RESPONSE(w/RII)	IDLE	
(tg_RESERVE) && (RMF="NO_AVAIL")	Send info to Application	IDLE	
(tg_QUERY) && (setRII)	tx_QUERY(w/RII)	WAITRESP1	
(tg_RESERVE) && (RII) && (RMF="AVAIL")	tx_RESERVE(w/RII)	WAITRESP2	
(rx_QUERY) && (!RII) && (setRII) && (RMF="AVAIL")	tx_RESERVE(w/RII)	WAITRESP2	2)

```
(rx_QUERY) && (!RII) && |tx_RESERVE(w/oRII),      |ESTABLISHED|2)
   (!setRII) &&         | Set QoS state,          |           |
   (RMF="AVAIL")        | Send info to Application|           |
                        |                         |           |
(tg_RESERVE) &&         |tx_RESERVE(w/oRII),      |ESTABLISHED|
   (!setRII) &&         | Set QoS state,          |           |
   (RMF="AVAIL")        | Send info to Application|           |
------------------------+-------------------------+-----------+---
```

Note: 1) tx_NOTIFY(ERROR) is transmitted when an ERROR event must be announced to other downstream nodes which do not expect a RESPONSE message for this action. E.g., there is no provided

RII which will be included in a RESPONSE message;
2) Relevant for Receiver-initiated reservation.

```
----------------
State: WAITRESP1
----------------

Condition               Action                    State
------------------------+-------------------------+------------
(TIMEOUT_Refresh) &&    |tx_RESERVE(w/RII)        |WAITRESP1
   (!MaxRetry)          |                         |
                        |                         |
(TIMEOUT_Refresh) &&    |Send info to Application |IDLE
   (MaxRetry)           |                         |
                        |                         |
rx_RESPONSE             |Send info to Application |IDLE
------------------------+-------------------------+------------


------------------
State: ESTABLISHED
------------------

Condition               Action                    State
------------------------+-------------------------+------------
TIMEOUT_Refresh         |tx_RESERVE               |ESTABLISHED
                        |                         |
tg_TEARDOWN             |tx_RESERVE(Ton),         |IDLE
```

	Clear QoS state	

State: WAITRESP2

Condition	Action	State
(TIMEOUT_Refresh) && (!MaxRetry)	tx_RESERVE(w/RII)	WAITRESP2
(TIMEOUT_Refresh) && (MaxRetry)	Send info to Application	IDLE
(rx_RESPONSE) && (Result="ERROR")	Send info to Application	IDLE
(rx_RESPONSE) && (Result="SUCCESS")&& (RMF="AVAIL")	Set QoS state, Send info to Application	ESTABLISHED

7. State machine for intermediate QoS NSLP nodes

```
-----------
State: INIT
-----------

Condition                 Action                      State
------------------------+--------------------------+------------
UCT                     | initialize variables     |IDLE
------------------------+--------------------------+------------


-----------
State: IDLE
```

```
-----------

Condition                 Action                       State      Note
------------------------+-------------------------+-----------+---
(rx_RESERVE)&& !((RII)&&|Set QoS state,           |ESTABLISHED|1a)
   (setRII)) &&         | tx_RESERVE(w/oRII)       |           |
   (RMF="AVAIL")        |                          |           |
                        |                          |           |
(rx_RESERVE) && (RII) &&|Set QoS state,           |ESTABLISHED|1b)
   (RMF="AVAIL") &&     | tx_RESPONSE(w/RII)       |           |
   (SCOPING="Next_hop") |                          |           |
                        |                          |           |
(rx_RESERVE) && (!RII)&&|Set QoS state            |ESTABLISHED|1b)
   (RMF="AVAIL") &&     |                          |           |
   (SCOPING="Next_hop") |                          |           |
                        |                          |           |
(rx_QUERY) && (!RII)    |tx_QUERY(w/oRII)          |IDLE       |2)
                        |                          |           |
(rx_QUERY) &&           |tx_QUERY(w/RII)           |IDLE       |
   (SCOPING="Next_hop") |                          |           |
                        |                          |           |
(rx_RESERVE) && (RII)&& |tx_RESPONSE(RII, ERROR)   |IDLE       |3)
   (RMF="NO_AVAIL")     |                          |           |
                        |                          |           |
(rx_RESERVE) && (!RII)&&|tx_NOTIFY(ERROR)          |IDLE       |3)
   (RMF="NO_AVAIL")     |                          |           |
                        |                          |           |
(rx_RESERVE) && ((RII)|||tx_RESERVE(w/RII)         |WAITRESP1  |4)
   (setRII)) &&         |                          |           |
   (RMF="AVAIL")        |                          |           |
                        |                          |           |
(rx_QUERY) && (RII)     |tx_QUERY(w/RII)           |WAITRESP2  |5)
                        |                          |           |
```



```
(tg_QUERY) && (setRII)  |tx_QUERY(w/RII)          |WAITRESP2  |5)
------------------------+-------------------------+-----------+---


------------------
State: ESTABLISHED
------------------
```

Condition	Action	State
rx_RESERVE(Ton)	tx_RESERVE(Ton), clear QoS state	IDLE
TIMEOUT_Refresh	Refresh QoS state; if state changes, tx_RESERVE(w/RII)	ESTABLISHED
TIMEOUT_State	Clear QoS state	IDLE

State: WAITRESP1

Condition	Action	State
(TIMEOUT_Refresh) && (!MaxRetry)	tx_RESERVE(w/RII) Send info to Application	WAITRESP1
(TIMEOUT_Refresh) && (MaxRetry) && (CheckRII="LOCAL")	tx_NOTIFY(ERROR), Send info to Application	IDLE
(TIMEOUT_Refresh) && (MaxRetry) && (CheckRII="NO_LOCAL")	tx_RESPONSE(w/RII,Result= "ERROR")	IDLE
(rx_RESPONSE) && (CheckRII="LOCAL")&& (Result="SUCCESS")	Set QoS state	ESTABLISHED
(rx_RESPONSE) && (CheckRII="NO_LOCAL") &&(Result="SUCCESS")	Set QoS state, tx_RESPONSE(RII)	ESTABLISHED
(rx_RESPONSE) && (CheckRII="LOCAL")&& (Result="ERROR")	tx_NOTIFY(ERROR), send info to Application	IDLE
(rx_RESPONSE) && (CheckRII="NO_LOCAL") &&(Result="ERROR")	tx_RESPONSE(w/RII)	IDLE

```
----------------
State: WAITRESP2
----------------

Condition                 Action                      State
------------------------+--------------------------+------------
(TIMEOUT_Refresh) &&    |tx_QUERY(w/RII)           |WAITRESP2
   (!MaxRetry)          |                          |
                        |                          |
(TIMEOUT_Refresh) &&    |Send info to Application  |IDLE
   (MaxRetry) &&        |                          |
   (CheckRII="LOCAL")   |                          |
                        |                          |
(TIMEOUT_Refresh) &&    |tx_RESPONSE(Result=       |IDLE
   (MaxRetry) &&        |  "ERROR")                |
   (CheckRII="NO_LOCAL")|                          |
                        |                          |
(rx_RESPONSE) &&        |Send info to Application  |IDLE
   (CheckRII="LOCAL")   |                          |
                        |                          |
(rx_RESPONSE) &&        |tx_RESPONSE(w/RII)        |IDLE
   (CheckRII="NO_LOCAL")|                          |
------------------------+--------------------------+-------------
```

Note: 1) Successful reservation with response request (1a) and with Scoping (1b);
2) Processing of Query msg for Receiver initiated reservation;
3) Unsuccessful reservation for Receiver initiated reservation, with/without request for response from the flow sender side. Tx_NOTIFY(ERROR) is sent to the upstream nodes to indicate failure of the reservation in the case when no RESPONSE is required by them;
4) Reservation requests with RII set in the upstream nodes or in this node;
5) Processing of Query message received from a neighboring node or triggered by the application layer.

8. State machine for last QoS NSLP node in the flow path

State: INIT

Condition	Action	State
UCT	initialize variables	IDLE

State: IDLE

Condition	Action	State	Note
(tg_QUERY) && (!setRII)	tx_QUERY(w/RII)	WAITRESV	1)
(rx_RESERVE) && (RII) && (RMF="AVAIL")	Set QoS state, tx_RESPONSE(w/RII)	ESTABLISHED	2a)
(rx_RESERVE) && (!RII)&& (RMF="AVAIL")	Set QoS state	ESTABLISHED	2b)
(tg_RESERVE) && (RMF="NO_AVAIL")	Send info to Application	IDLE	3)
(rx_RESPONSE) && (RII)&& (RMF="NO_AVAIL")	tx_RESPONSE(RII, ERROR)	IDLE	
(rx_QUERY) && (RII) &&	tx_QUERY(w/RII)	IDLE	

State: ESTABLISHED

Condition	Action	State
rx_RESERVE	Refresh QoS state	ESTABLISHED
TIMEOUT_State	Clear QoS state	IDLE

State: WAITRESV

Condition	Action	State	Note
(rx_RESPONSE) && (Result="ERROR")	Send info to Application	IDLE	
(rx_RESERVE) && (!RII)&& (RMF="AVAIL")	Set QoS state	ESTABLISHED	2)
(rx_RESERVE) && (RII)&& (RMF="AVAIL")	Set QoS state, tx_RESPONSE(w/RII)	ESTABLISHED	2)
(rx_RESERVE) && (RII) && (RMF="NO_AVAIL")	tx_NOTIFY(ERROR), Send info to Application	IDLE	4)
(rx_RESERVE) && (!RII)&& (RMF="NO_AVAIL")	tx_RESPONSE(RII,ERROR), Send info to Application	IDLE	4)

Note: 1) Initiation of receiver-side reservation;
2) Successful reservation with&without response request from sender side;
3) In case of no response requested (RII not present in RESERVE message), NOTIFY(ERROR) message is sent back to the upstream nodes in order to clear already established QoS state;
4) Unsuccessful reservation with&without response request from sender side;

9. Security Considerations

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

For the time being, the state machines described in this document do not consider the security aspect of QoS NSLP protocol itself. A future versions of this document will add security relevant states and state transitions.

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. For example, for receiver-initiated reservation, it is unclear who triggers a teardown; 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). Another example is, it is often ignored for the functionality of abort operation after a defined MaxRetry number of retries. Results of this type of transitions are dependent on the parameter RII (e.g., if it is locally set or not).

There are further unclear issues with processing rules and message definition, e.g., soft state handling and how to process notification messages, which will be described in more detail in a future version of this document.

To avoid confusions in state machines, instead of QNI, QNE and QNR, in this document we use the notations of "first QoS NSLP node in the flow path" (the closest one to the flow sender or the flow sender itself), "intermediate QoS NSLP nodes" and "last QoS NSLP node in the flow path" (the closest one to the flow receiver or the flow receiver itself).

Default rules and common state transitions in case of reception of certain messages as Notify, and Query(w/RII), will be described in a future version of this document.

11. Acknowledgments

The authors would like to thank Sven Van den Bosch for his feedback.

12. References

12.1 Normative References

[1] Bosch, S., Karagiannis, G. and A. McDonald, "NSLP for Quality-of-Service signaling", draft-ietf-nsis-qos-nslp-04 (work in progress), July 2004.

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

12.2 Informative References

[3] Vollbrecht, J., Eronen, P., Petroni, N. and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator", draft-ietf-eap-statemachine-05 (work in progress), September 2004.

[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/D9, January 2004.

[5] Ohba, Y., "State Machines for Protocol for Carrying Authentication for Network Access (PANA)", draft-ohba-pana-statemachine-00 (work in progress), July 2004.

[6] Tschofenig, H. and D. Kroeselberg, "Security Threats for NSIS", draft-ietf-nsis-threats-06 (work in progress), October 2004.

Authors' Addresses

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

EMail: fu@cs.uni-goettingen.de

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

EMail: Hannes.Tschofenig@siemens.com

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

EMail: tseno.tsenov@mytum.de

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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