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

Internet-Draft
Expiration Date: June 24, 2007

December 25, 2006

QoS NSLP State Machine
draft-fu-nsis-qos-nslp-statemachine-05.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 defi ned 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 Identifi cation 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.

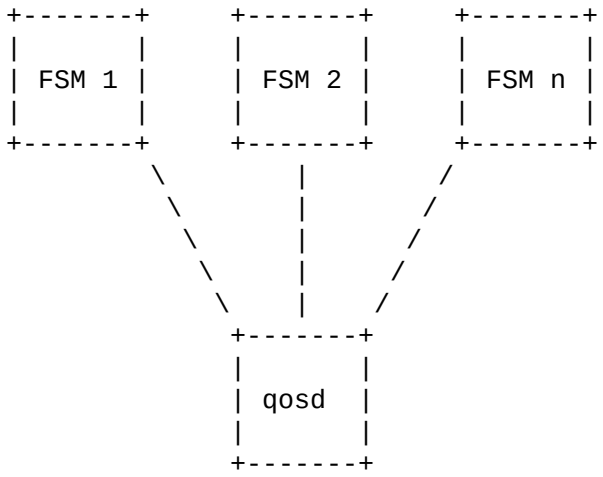

Figure 1

6.2 States, Events and Callback Functions

Three States are defi ned: 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_TG_QUERY	idle__tg_query
ST_IDLE	EV_RX_QUERY	idle__rx_query
ST_IDLE	EV_TG_RESERVE	idle__tg_reserve
ST_IDLE	EV_RX_RESERVE	idle__rx_reserve
ST_IDLE	EV_RX_RESPONSE	idle__rx_response
ST_WR	EV_TG_QUERY	wr__tg_query
ST_WR	EV_RX_QUERY	wr__rx_query
ST_WR	EV_TG_RESERVE	wr__tg_reserve
ST_WR	EV_RX_RESERVE	wr__rx_reserve
ST_WR	EV_RX_RESPONSE	wr__rx_response
ST_WR	EV_TIMEOUT_WAITRESP	wr__timeout_waitresp
ST_INST	EV_TG_QUERY	inst__tg_query
ST_INST	EV_RX_QUERY	inst__rx_query
ST_INST	EV_RX_NOTIFY	inst__rx_notify
ST_INST	EV_TG_RESERVE	inst__tg_reserve
ST_INST	EV_RX_RESERVE	inst__rx_reserve
ST_INST	EV_RX_RESPONSE	inst__rx_response
ST_INST	EV_TIMEOUT_WAITRESP	inst__timeout_waitresp
ST_INST	EV_TIMEOUT_REFRESH	inst__timeout_refresh
ST_INST	EV_TIMEOUT_STATELIFETIME	inst__timeout_statelifetime

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 confi rmation is expected. On arrival the StateLife Timer is restarted.

If the confi rmation 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.

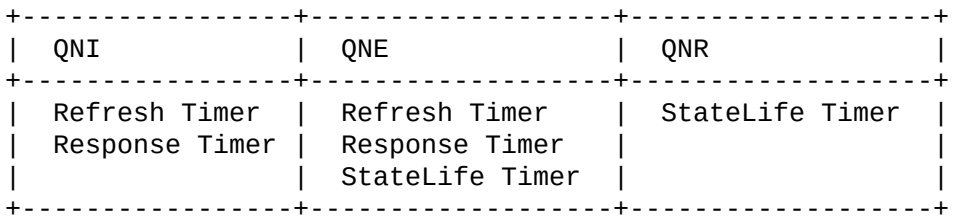

Figure 3

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

7.1 State ST_IDLE

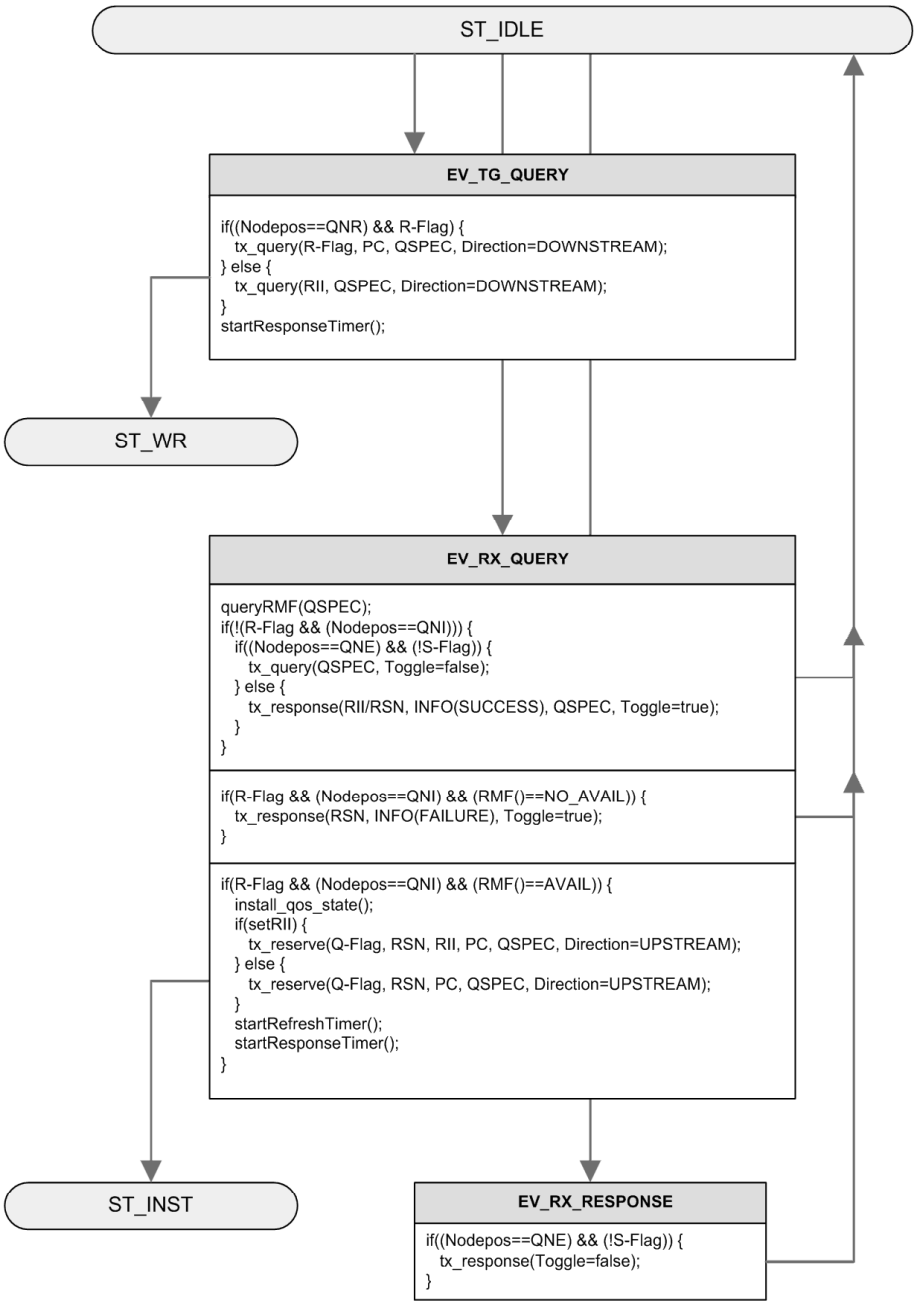

Figure 4

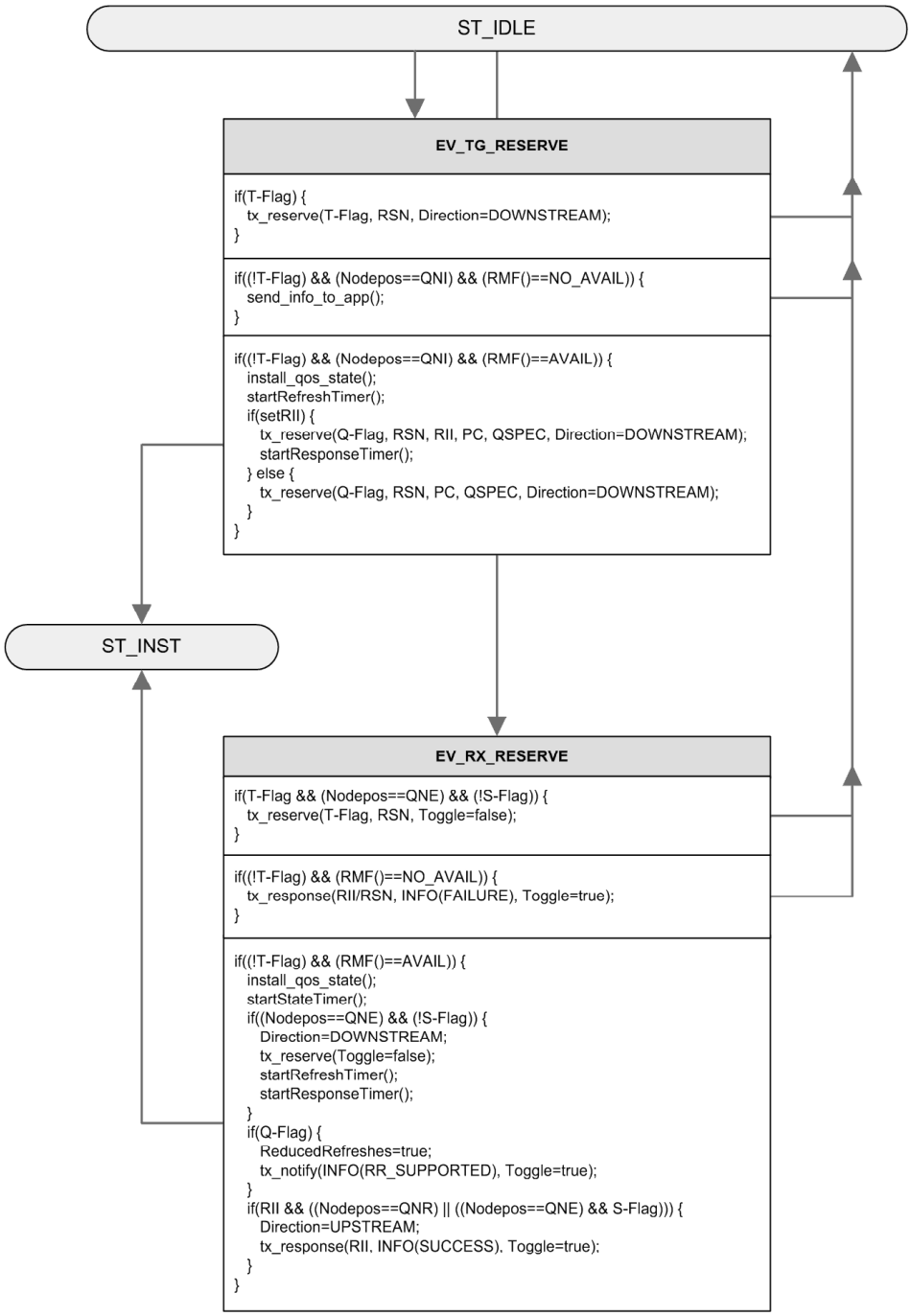
ST_IDLE
EV_TG_RESERVE
if(T-Flag) {
tx_reserve(T-Flag, RSN, Direction=DOWNSTREAM);
}
if((!T-Flag) && (Nodepos==QNI) && (RMF()==NO_AVAIL)) {
send_info_to_app();
}
if((!T-Flag) && (Nodepos==QNI) && (RMF()==AVAIL)) {
install_qos_state();
startRefreshTimer();
if(setRII) {
tx_reserve(Q-Flag, RSN, RII, PC, QSPEC, Direction=DOWNSTREAM);
startResponseTimer();
} else {
tx_reserve(Q-Flag, RSN, PC, QSPEC, Direction=DOWNSTREAM);
}
}
ST_INST
EV_RX_RESERVE
if(T-Flag && (Nodepos==QNE) && (!S-Flag)) {
tx_reserve(T-Flag, RSN, Toggle=false);
}
if((!T-Flag) && (RMF()==NO_AVAIL)) {
tx_response(RII/RSN, INFO(FAILURE), Toggle=true);
}
if((!T-Flag) && (RMF()==AVAIL)) {
install_qos_state();
startStateTimer();
if((Nodepos==QNE) && (!S-Flag)) {
Direction=DOWNSTREAM;
tx_reserve(Toggle=false);
startRefreshTimer();
startResponseTimer();
}
if(Q-Flag) {
ReducedRefreshes=true;
tx_notify(INFO(RR_SUPPORTED), Toggle=true);
}
if(RII && ((Nodepos==QNR) || ((Nodepos==QNE) && S-Flag))) {
Direction=UPSTREAM;
tx_response(RII, INFO(SUCCESS), Toggle=true);
}
}

Figure 5

7.2 State ST_WR

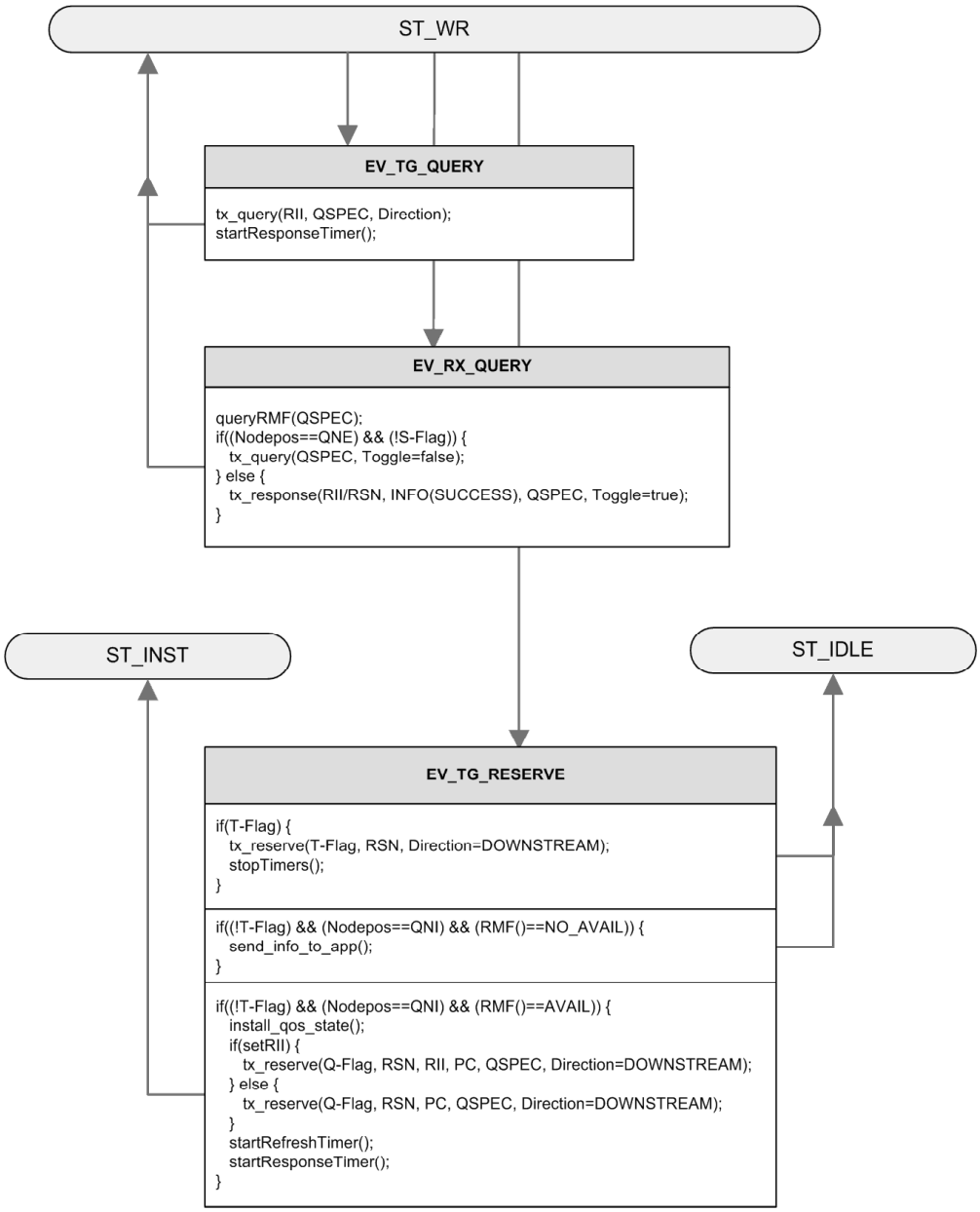

Figure 6

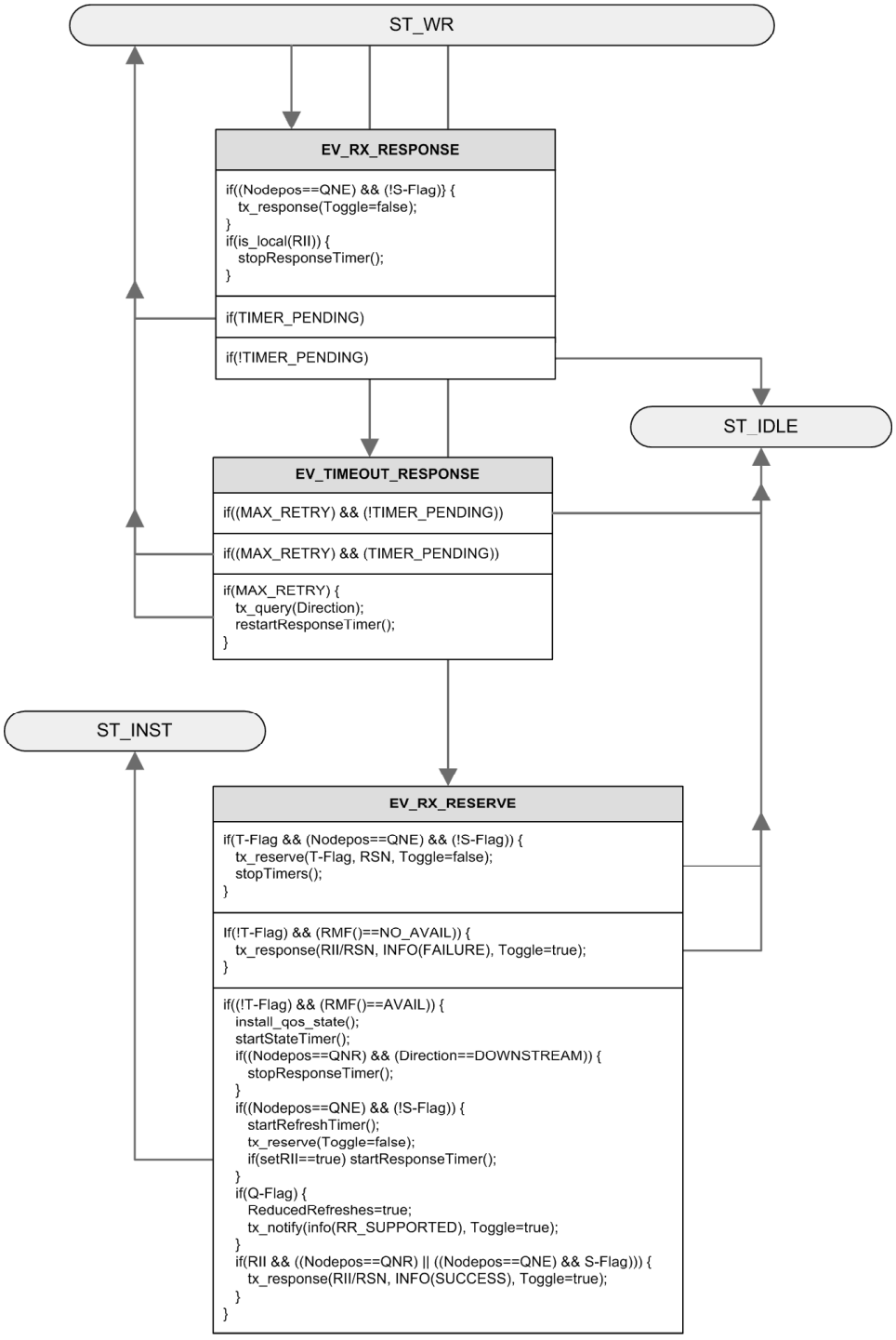
ST_WR
EV_RX_RESPONSE
if((Nodepos==QNE) && (!S-Flag)} {
tx_response(Toggle=false);
}
if(is_local(RII)) {
stopResponseTimer();
}
if(TIMER_PENDING)
if(!TIMER_PENDING)
ST_IDLE
EV_TIMEOUT_RESPONSE
if((MAX_RETRY) && (!TIMER_PENDING))
if((MAX_RETRY) && (TIMER_PENDING))
if(MAX_RETRY) {
tx_query(Direction);
restartResponseTimer();
}
ST_INST
EV_RX_RESERVE
if(T-Flag && (Nodepos==QNE) && (!S-Flag)) {
tx_reserve(T-Flag, RSN, Toggle=false);
stopTimers();
}
If(!T-Flag) && (RMF()==NO_AVAIL)) {
tx_response(RII/RSN, INFO(FAILURE), Toggle=true);
}
if((!T-Flag) && (RMF()==AVAIL)) {
install_qos_state();
startStateTimer();
if((Nodepos==QNR) && (Direction==DOWNSTREAM)) {
stopResponseTimer();
}
if((Nodepos==QNE) && (!S-Flag)) {
startRefreshTimer();
tx_reserve(Toggle=false);
if(setRII==true) startResponseTimer();
}
if(Q-Flag) {
ReducedRefreshes=true;
tx_notify(info(RR_SUPPORTED), Toggle=true);
}
if(RII && ((Nodepos==QNR) || ((Nodepos==QNE) && S-Flag))) {
tx_response(RII/RSN, INFO(SUCCESS), Toggle=true);
}
}

Figure 7

7.3 State ST_INST

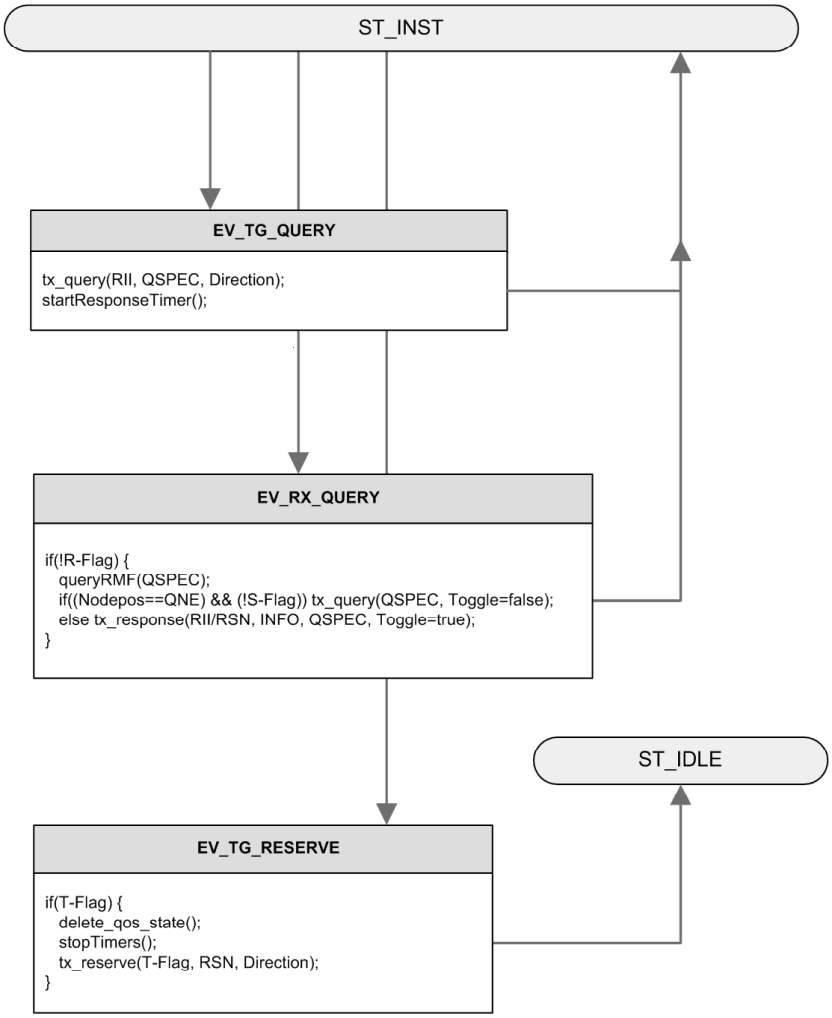

Figure 8

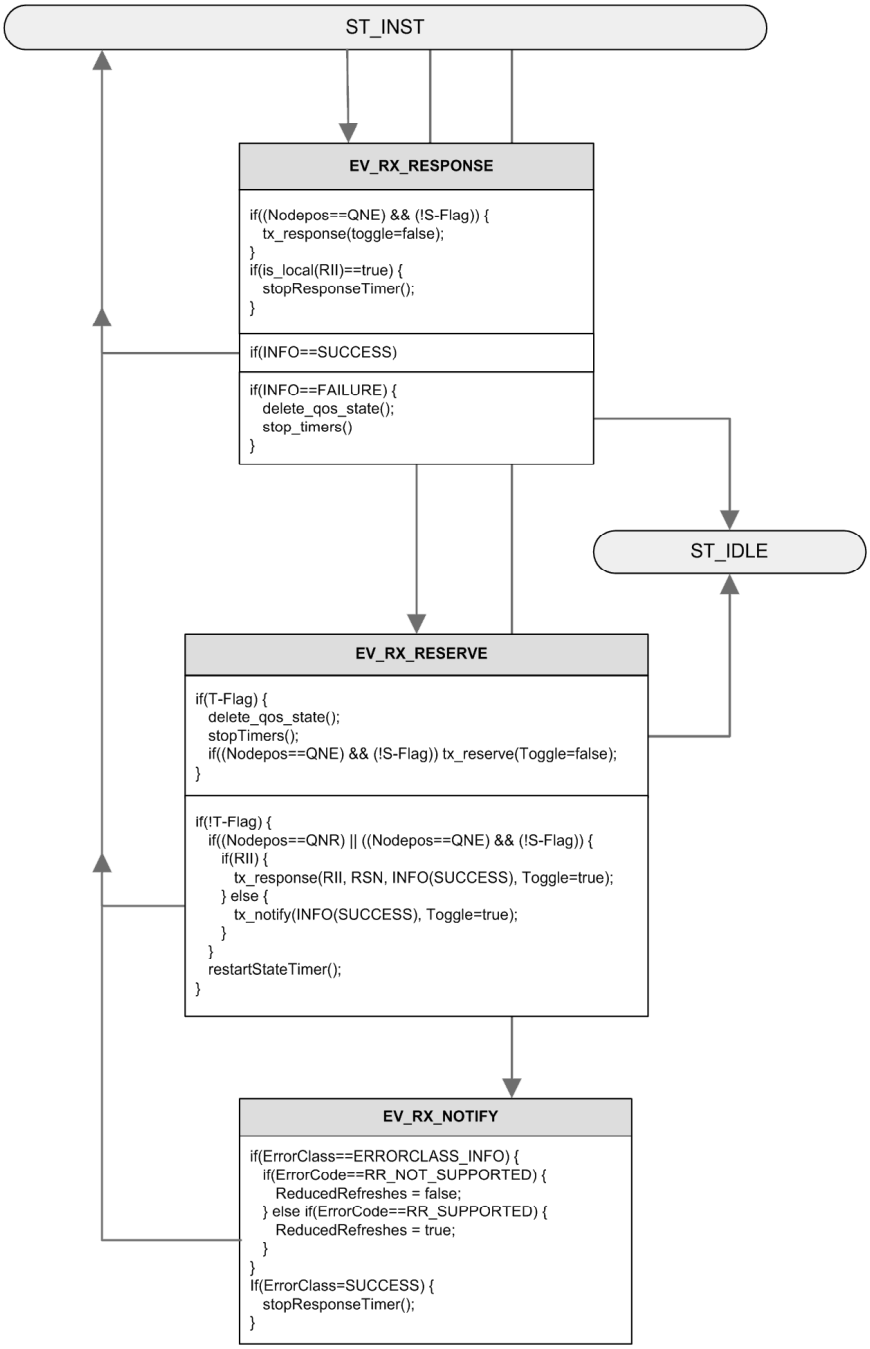
ST_INST
EV_RX_RESPONSE
if((Nodepos==QNE) && (!S-Flag)) {
tx_response(toggle=false);
}
if(is_local(RII)==true) {
stopResponseTimer();
}
if(INFO==SUCCESS)
if(INFO==FAILURE) {
delete_qos_state();
stop_timers()
}
ST_IDLE
EV_RX_RESERVE
if(T-Flag) {
delete_qos_state();
stopTimers();
if((Nodepos==QNE) && (!S-Flag)) tx_reserve(Toggle=false);
}
if(!T-Flag) {
if((Nodepos==QNR) || ((Nodepos==QNE) && (!S-Flag)) {
if(RII) {
tx_response(RII, RSN, INFO(SUCCESS), Toggle=true);
} else {
tx_notify(INFO(SUCCESS), Toggle=true);
}
}
restartStateTimer();
}
EV_RX_NOTIFY
if(ErrorClass==ERRORCLASS_INFO) {
if(ErrorCode==RR_NOT_SUPPORTED) {
ReducedRefreshes = false;
} else if(ErrorCode==RR_SUPPORTED) {
ReducedRefreshes = true;
}
}
If(ErrorClass=SUCCESS) {
stopResponseTimer();
}

Figure 9

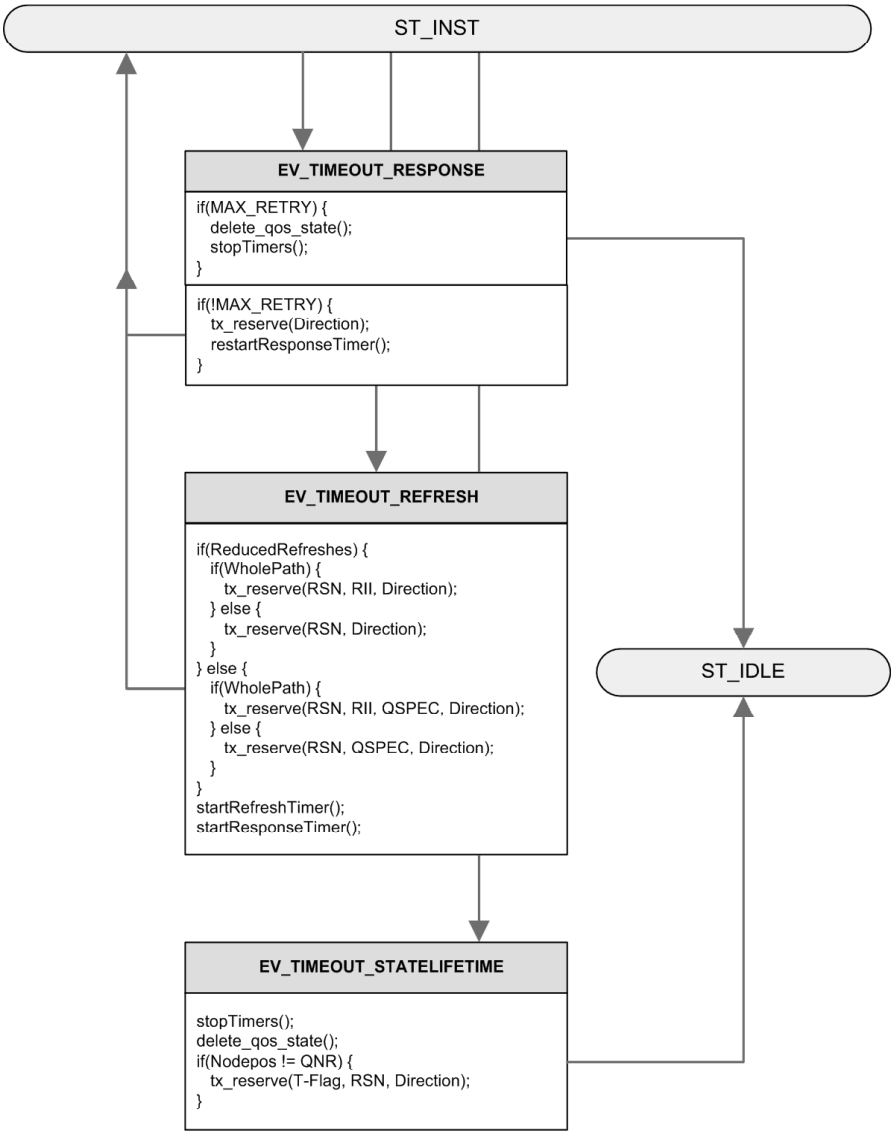
ST_INST
EV_TIMEOUT_RESPONSE
if(MAX_RETRY) {
delete_qos_state();
stopTimers();
}
if(!MAX_RETRY) {
tx_reserve(Direction);
restartResponseTimer();
}
EV_TIMEOUT_REFRESH
if(ReducedRefreshes) {
if(WholePath) {
tx_reserve(RSN, RII, Direction);
} else {
tx_reserve(RSN, Direction);
}
} else {
if(WholePath) {
tx_reserve(RSN, RII, QSPEC, Direction);
} else {
tx_reserve(RSN, QSPEC, Direction);
}
}
startRefreshTimer();
startResponseTimer();
ST_IDLE
EV_TIMEOUT_STATELIFETIME
stopTimers();
delete_qos_state();
if(Nodepos != QNR) {
tx_reserve(T-Flag, RSN, Direction);
}

Figure 10

8. Actions and Transitions

This chapter describes the operation of the FSM.

8.1 State ST_IDLE

8.1.1 idle__tg_query

The local application has triggered a QUERY in ST_IDLE. If the node position is QNR and a Receiver Initiated Reservation is requested, then a QUERY message is constructed with set R-Flag together with a PC and a QSPEC. No RII object is included because a RESPONSE message does not need to be triggered. In case of a normal QUERY a RII object is included to match back the response. It also contains a QSPEC to perform a query along the path. In both cases the variable Direction is set to DOWNSTREAM and the Response Timer is started. The message is sent downstream and transition is made to ST_WR.

8.1.2 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 QSPEC object is present, then it is passed to the RMF. If the R-Flag is not set and the node position is QNE and the Scoping Flag is not set, then a new QUERY message with the returned QSPEC from the RMF is constructed and sent further along the path. If the Scoping Flag is set or the node position is QNI or QNR, then a RESPONSE message with the received RII object, an INFO object with Error Class SUCCESS and the QSPEC object is sent back with set Toggle Flag. Transition is made to ST_IDLE.

If the R-Flag is set and the node position is QNI and the RMF returns that no resources are available, then a RESPONSE with a RSN object and a INFO object with Error Class Failure is sent back. Transition is done to ST_IDLE.

In the other case where resources are available, the RMF performs resource reservation. A new RESERVE message is created, the Q-Flag is set, if Reduced Refreshes were requested, a RSN object, a PC and the returned QSPEC is added. If a response is requested, then a RII object is added additionally and the response timer is started. Direction is set to UPSTREAM and the message is sent upstream. The refresh timer is started. Transition is made to ST_INST.

8.1.3 idle__tg_reserve

The local application has triggered a RESERVE message to be sent. If the T-Flag is set, then a RESERVE message is sent downstream with set T-Flag and with a RSN object. Transition is made to ST_IDLE.

If the node position is QNI and the RMF returns that no resources are available, then a FAILURE is sent back to the local application. Transition is made to ST_IDLE.

If resources are available, then the RMF performs resource reservation, a new RESERVE message is created, the Q-Flag is set if reduced refreshes are requested, a RSN object, a PC and the returned QSPEC is added. If a response is requested, then a RII object is added additionally and the response timer is started. Direction is set to DOWNSTREAM and the message is sent downstream. The refresh timer is started. Transition is made to ST_INST.

8.1.4 idle__rx_reserve

A RESERVE from the network in ST_IDLE has arrived. If the T-Flag is set, if the node position is QNE and if the Scoping Flag is not set, then the message is forwarded further along the path with toggle=false. Transition is made to ST_IDLE.

If the T-Flag is not set and the RMF returns that no resources are available, then a RESPONSE with a RII (if available, else with RSN object) and a INFO object with Error Class Failure is sent back. Transition is done to ST_IDLE.

If resources are available, then the RMF performs resource reservation, the State Timer is started. If the node position is QNE and the Scoping Flag is not set, then a new RESERVE message is created out of the old one and sent further downstream. The Response Timer is started additionally to the Refresh Timer. If the Q-Flag was set in the received message, the variable ReducedRefreshes is set to true and a NOTIFY with the corresponding INFO object is sent back with toggle=true. If the received message contained a RII object and the node is QNR or QNE and the Scoping Flag is set, then a RESPONSE with this RII object and INFO=SUCCESS is sent back. Transition is made to ST_INST.

8.1.5 idle__rx_response

If the node position is QNE and the Scoping Flag is not set, then the RESPONSE message is forwarded further along the path. Transition is made to ST_IDLE.

8.2 State ST_WR

8.2.1 wr__tg_query

The local application has triggered a QUERY in ST_WR. The value of the RII object is stored and the message is sent to the stored direction together with a QSPEC object. The Response Timer is started. Transition is made to ST_WR.

8.2.2 wr__rx_query

This function is executed when a QUERY message has arrived over the network from GIST and the FSM is in State ST_WR. If a QSPEC object is present, then it is passed to the RMF. If the R-Flag is not set and the node position is QNE and the Scoping Flag is not set, then a new QUERY message with the returned QSPEC from the RMF is constructed and sent further along the path. If the Scoping Flag is set or the node position is QNI or QNR, then a RESPONSE message with the received RII object, an INFO object with Error Class Success and the QSPEC object is sent back with set Toggle Flag. Transition is made to ST_WR.

8.2.3 wr__tg_reserve

See idle__tg_reserve.

8.2.4 wr__rx_reserve

See idle__rx_reserve. If the node position is QNR and the stored Direction is DOWNSTREAM, then this is the requested RESERVE message for the Receiver Initiated Reservation. The Response Timer is stopped.

Transition is made to ST_INST.

8.2.5 wr__rx_response

In ST_WR 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 object is stored to match back a response, then the Response Timer is stopped. If no more Response Timer are pending, transition is made to ST_IDLE, otherwise to ST_WR.

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, then the Response Timer is stopped. If no more Response Timer are pending, transition is done to ST_IDLE, otherwise to ST_WR. If the maximum number of retransmissions has not been reached, then the stored message is resent and transition is made to ST_WR.

8.3 State ST_INST

8.3.1 inst__tg_query

See wr__tg_query. Transition is made to ST_INST.

8.3.2 inst__rx_query

See wr__rx_query. Transition is made to ST_INST.

8.3.3 inst__rx_notify

If Reduced Refreshes were requested, then the Error Code is checked whether the next peer has accepted them. After receiving the RII object Reduced Refreshes can be used as refreshing RESERVEs. Transition is made to ST_INST.

8.3.4 inst__tg_reserve

If the T-Flag is set, then all pending timers are stopped, the reservation is torn down and the message is sent further along the path into the stored direction with toggle=false. Transition is done to ST_IDLE.

8.3.5 inst__rx_reserve

A RESERVE message in ST_INST has arrived. If the T-Flag is set, then all pending timer are stopped, the reservation is torn down. If the node position is QNE and the S-Flag is not set, then the RESERVE is forwarded with toggle=false. Transition is done to ST_IDLE.

If the T-Flag is not set, then this is a refreshing RESERVE. A RESPONSE with the received RII and INFO=SUCCESS is sent back with Toggle=false, if the node position is QNR or QNE and the S-Flag is set. The Statelife Timer is being restarted.

8.3.6 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 object is stored to match back a response, 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 timer are stopped. Transition is made to ST_IDLE.

8.3.7 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. Transition is made to ST_IDLE. If maximum numbers of retransmissions has not been reached, then the refreshing RESERVE is resent and the Response Timer is restarted. Transition is made to ST_INST.

8.3.8 inst__timeout_refresh

A RESERVE message is created. If Reduced Refreshes are accepted, no QSPEC object is added. If the refresh is sent along the whole path, then a RII object is added. The Response and the Refresh Timer is restarted. Transition is made to ST_INST.

8.3.9 inst__timeout_statelifetime

The Statelife Timer timed out in ST_INST. All active timer are stopped and existing reservations are removed. If the node position is not QNR, then a RESERVE message with T-Flag set is created and sent into the stored direction. Transition is done to ST_IDLE.

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. A future version 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.

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, QSPEQ, 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

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-11 (work in progress), June 2006.

[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", draft-ietf-eap-statemachine-06 (work in progress), December 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/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", draft-ietf-nsis-threats-06 (work in progress), October 2004.

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_TG_RESERVE

Action	new State

if(T-Flag) {
  tx_reserve(T-Flag, RSN, Direction=DOWNSTREAM);
}
	ST_IDLE

if((!T-Flag) && (Nodepos==QNI) && (RMF()==NO_AVAIL)) {
  send_info_to_app();
}
	ST_IDLE

if((!T-Flag) && (Nodepos==QNI) && (RMF()==AVAIL)) {
  install_qos_state();
  if(setRII) {
  tx_reserve(Q-Flag, RSN, RII, PC, QSPEC,
  Direction=DOWNSTREAM);
  } else {
  tx_reserve(Q-Flag, RSN, PC, QSPEC,
  Direction=DOWNSTREAM);
  }
  startRefreshTimer();
  startResponseTimer();
}
	ST_INST

Figure 11

```
ST_IDLE::EV_RX_RESERVE
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(T-Flag && (Nodepos==QNE) && (!S-Flag)) {               |  ST_IDLE  |
|    tx_reserve(T-Flag, RSN, Toggle=false);                 |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((!T-Flag) && (RMF()==NO_AVAIL)) {                      |  ST_IDLE  |
|    tx_response(RII/RSN, INFO(FAILURE), Toggle=true)       |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((!T-Flag) && (RMF()==AVAIL)) {                         |  ST_INST  |
|    install_qos_state();                                   |           |
|    startStateTimer();                                     |           |
|    if((Nodepos==QNE) && (!S-Flag)) {                      |           |
|       Direction=DOWNSTREAM;                               |           |
|       tx_reserve(Toggle=false);                           |           |
|       startRefreshTimer();                                |           |
|       startResponseTimer();                               |           |
|    }                                                      |           |
|    if(Q-Flag) {                                           |           |
|       ReducedRefreshes=true;                              |           |
|       tx_notify(INFO(RR_SUPPORTED), Toggle=true);         |           |
|    }                                                      |           |
|    if(RII && ((Nodepos==QNR) ||                           |           |
|       ((Nodepos==QNE) && S-Flag))) {                      |           |
|       Direction=UPSTREAM;                                 |           |
|       tx_response(RII, INFO(SUCCESS), Toggle=true);       |           |
|    }                                                      |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 12

```
ST_IDLE::EV_TG_QUERY
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((Nodepos==QNR) && R-Flag) {                            |  ST_WR    |
```

```
|    tx_query(R-Flag, PC, QSPEC, Direction=DOWNSTREAM);   |           |
| } else {                                                |           |
|    tx_query(RII, QSPEC, Direction=DOWNSTREAM);          |           |
| }                                                       |           |
| startResponseTimer();                                   |           |
|                                                         |           |
+---------------------------------------------------------+-----------+
```

Figure 13

```
ST_IDLE::EV_RX_QUERY
+---------------------------------------------------------+-----------+
| Action                                                  | new State |
+---------------------------------------------------------+-----------+
|                                                         |           |
| queryRMF(QSPEC);                                        |  ST_IDLE  |
| if(!(R-Flag && (Nodepos==QNI))) {                       |           |
|    if((Nodepos==QNE) && (!S-Flag)) {                    |           |
|       tx_query(QSPEC, Toggle=false);                    |           |
|    } else {                                             |           |
|       tx_response(RII/RSN, INFO(SUCCESS),               |           |
|          QSPEC, Toggle=true);                           |           |
|    }                                                    |           |
| }                                                       |           |
|                                                         |           |
+---------------------------------------------------------+-----------+
|                                                         |           |
| if(R-Flag && (Nodepos==QNI) && (RMF()==NO_AVAIL)) {     |  ST_IDLE  |
|    tx_response(RSN, INFO(FAILURE), Toggle=true);        |           |
| }                                                       |           |
|                                                         |           |
+---------------------------------------------------------+-----------+
|                                                         |           |
| if(R-Flag && (Nodepos==QNI) && (RMF()==AVAIL)) {        |  ST_INST  |
|    install_qos_state();                                 |           |
|    if(setRII) {                                         |           |
|       tx_reserve(Q-Flag, RSN, RII, PC, QSPEC,           |           |
|          Direction=UPSTREAM);                           |           |
|    } else {                                             |           |
|       tx_reserve(Q-Flag, RSN, PC, QSPEC,                |           |
|          Direction=UPSTREAM);                           |           |
|    }                                                    |           |
|    startRefreshTimer();                                 |           |
|    startResponseTimer();                                |           |
| }                                                       |           |
|                                                         |           |
+---------------------------------------------------------+-----------+
```

Figure 14

```
ST_IDLE::EV_RX_RESPONSE
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((Nodepos==QNE) && (!S-Flag)) {                         |  ST_IDLE  |
|    tx_response(Toggle=false);                             |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 15

A.2. State ST_WR

```
ST_WR::EV_RX_RESPONSE
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((Nodepos==QNE) && (!S-Flag)) {                         |           |
|    tx_response(Toggle=false);                             |           |
| }                                                         |           |
| if(is_local(RII)==true) {                                 |           |
|    stopResponseTimer();                                   |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(TIMER_PENDING==true)                                   |  ST_WR    |
|                                                           |           |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(TIMER_PENDING==false)                                  |  ST_IDLE  |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 16

```
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 17

ST_WR::EV_RX_RESERVE

Action	new State

if(T-Flag && (Nodepos==QNE) && (!S-Flag)) {
  tx_reserve(T-Flag, RSN, Toggle=false);
  stopTimers();
}
	ST_IDLE

if((!T-Flag) && (RMF()==NO_AVAIL)) {
  tx_response(RII/RSN, INFO(FAILURE), Toggle=true);
}
	ST_IDLE

if((!T-Flag) && (RMF()==AVAIL)) {
  install_qos_state();
  startStateTimer();
  if((Nodepos==QNR) && (Direction==DOWNSTREAM)) {
  stopResponseTimer();
  }
  if((Nodepos==QNE) && (!S-Flag)) {
  tx_reserve(Toggle=false);
  startRefreshTimer();
  startResponseTimer();
  }
	ST_INST

```
|    if(Q-Flag) {                                          |           |
|       ReducedRefreshes=true;                             |           |
|       tx_notify(info(RR_SUPPORTED), Toggle=true);        |           |
|    }                                                     |           |
|    if(RII && ((Nodepos==QNR) ||                          |           |
|       ((Nodepos==QNE) && S-Flag))) {                     |           |
|       tx_response(RII, INFO(SUCCESS), Toggle=true);      |           |
|    }                                                     |           |
| }                                                        |           |
|                                                          |           |
+----------------------------------------------------------+-----------+
```

Figure 18

```
ST_WR::EV_TG_QUERY
+----------------------------------------------------------+-----------+
| Action                                                   | new State |
+----------------------------------------------------------+-----------+
|                                                          |           |
| tx_query(RII, QSPEC, Direction);                         |  ST_WR    |
| startResponseTimer();                                    |           |
|                                                          |           |
+----------------------------------------------------------+-----------+
```

Figure 19

```
ST_WR::EV_RX_QUERY
+----------------------------------------------------------+-----------+
| Action                                                   | new State |
+----------------------------------------------------------+-----------+
|                                                          |           |
| queryRMF(QSPEC);                                         |  ST_WR    |
| if((Nodepos==QNE) && (!S-Flag)) {                        |           |
|    tx_query(QSPEC, Toggle=false);                        |           |
| } else {                                                 |           |
|    tx_response(RII/RSN, INFO(SUCCESS), QSPEC,            |           |
|       Toggle=true);                                      |           |
| }                                                        |           |
|                                                          |           |
+----------------------------------------------------------+-----------+
```

Figure 20

```
ST_WR::EV_TG_RESERVE
```

```
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(T-Flag) {                                              |  ST_IDLE  |
|    tx_reserve(T-Flag, RSN, Direction=DOWNSTREAM);         |           |
|    stopTimers();                                          |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((!T-Flag) && (Nodepos==QNI) && (RMF()==NO_AVAIL)) {    |  ST_IDLE  |
|    send_info_to_app();                                    |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((!T-Flag) && (Nodepos==QNI) && (RMF()==AVAIL)) {       |  ST_INST  |
|    install_qos_state();                                   |           |
|    if(setRII) {                                           |           |
|       tx_reserve(Q-Flag, RSN, RII, PC, QSPEC,            |           |
|          Direction=DOWNSTREAM);                           |           |
|    } else {                                               |           |
|       tx_reserve(Q-Flag, RSN, PC, QSPEC,                  |           |
|          Direction=DOWNSTREAM);                           |           |
|    }                                                      |           |
|    startRefreshTimer();                                   |           |
|    startResponseTimer();                                  |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 21

A.3. State ST_INST

```
ST_INST::EV_RX_RESPONSE
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if((Nodepos==QNE) && (!S-Flag)) {                         |           |
|    tx_response(toggle=false);                             |           |
| }                                                         |           |
| if(is_local(RII)==true) {                                 |           |
|    stopResponseTimer();                                   |           |
| }                                                         |           |
|                                                           |           |
```

if(INFO==SUCCESS)	ST_INST

if(INFO==FAILURE) {
delete_qos_state();
stop_timers();
}
	ST_IDLE

Figure 22

ST_INST::EV_RX_RESERVE

Action	new State

if(T-Flag) {
delete_qos_state();
stopTimers();
if((Nodepos==QNE) && (!S-Flag)) {
tx_reserve(Toggle=false);
}
}
	ST_IDLE

if(!T-Flag) {
if((Nodepos==QNR) || ((Nodepos==QNE)&&(!S-Flag))) {
if(RII) {
tx_response(RII, RSN, INFO(SUCCESS),
Toggle=true);
} else {
tx_notify(INFO(SUCCESS), Toggle=true);
}
}
restartStateTimer();
}
	ST_INST

Figure 23

```
ST_INST::EV_RX_NOTIFY
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(ErrorClass==ERRORCLASS_INFO) {                         |  ST_INST  |
|    if(ErrorCode==RR_NOT_SUPPORTED) {                      |           |
|       ReducedRefreshes = false;                           |           |
|    } else if(ErrorCode==RR_SUPPORTED) {                   |           |
|       ReducedRefreshes = true;                            |           |
|    }                                                      |           |
| }                                                         |           |
| if(ErrorClass==SUCCESS) {                                 |           |
|    stopResponseTimer();                                   |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 24

```
ST_INST::EV_TG_QUERY
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| tx_query(RII, QSPEC, Direction);                          |  ST_INST  |
| startResponseTimer();                                     |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 25

```
ST_INST::EV_RX_QUERY
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(!R-Flag) {                                             |  ST_INST  |
|    queryRMF(QSPEC);                                       |           |
|    if((Nodepos==QNE) && (!S-Flag))                        |           |
|       tx_query(QSPEC, Toggle=false);                      |           |
|    } else {                                               |           |
|       tx_response(RII/RSN, INFO, QSPEC, Toggle=true);     |           |
|    }                                                      |           |
| }                                                         |           |
|                                                           |           |
```

```
+-----------------------------------------------------------+-----------+
```

Figure 26

```
ST_INST::EV_TG_RESERVE
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(T-Flag) {                                              |  ST_INST  |
|    delete_qos_state();                                    |           |
|    stopTimers();                                          |           |
|    tx_reserve(T-Flag, RSN, Direction);                    |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 27

```
ST_INST::EV_TIMEOUT_RESPONSE
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(MAX_RETRY) {                                           |  ST_IDLE  |
|    delete_qos_state();                                    |           |
|    stopTimers();                                          |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| if(!MAX_RETRY) {                                          |  ST_INST  |
|    tx_reserve(Direction);                                 |           |
|    restartResponseTimer()                                 |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 28

```
ST_INST::EV_TIMEOUT_REFRESH
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
```

```
| if(ReducedRefreshes) {                                    |  ST_INST  |
|    if(WholePath) {                                        |           |
|       tx_reserve(RSN, RII, Direction);                    |           |
|    } else {                                               |           |
|       tx_reserve(RSN, Direction);                         |           |
|    }                                                      |           |
| } else {                                                  |           |
|    if(WholePath) {                                        |           |
|       tx_reserve(RSN, RII, QSPEC, Direction);             |           |
|    } else {                                               |           |
|       tx_reserve(RSN, QSPEC, Direction);                  |           |
|    }                                                      |           |
| }                                                         |           |
| startResponseTimer();                                     |           |
| restartRefreshTimer();                                    |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 29

```
ST_INST::EV_TIMEOUT_STATELIFETIME
+-----------------------------------------------------------+-----------+
| Action                                                    | new State |
+-----------------------------------------------------------+-----------+
|                                                           |           |
| stopTimers();                                             |  ST_IDLE  |
| delete_qos_state();                                       |           |
| if(Nodepos != QNR) {                                      |           |
|    tx_reserve(T-Flag, RSN, Direction);                    |           |
| }                                                         |           |
|                                                           |           |
+-----------------------------------------------------------+-----------+
```

Figure 30

Authors' Addresses

Xiaoming Fu
University of Goettingen
Telematics Group
Lotzestrasse 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

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

Email: bschloer@cs.uni-goettingen.de

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgement

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