NSIS T . Tsenov
Internet-Draft H. Tschofenig
Intended status: Informational Nokia Siemens Networks
Expires: October 24, 2010 X. Fu (Ed.)
Univ. Goettingen
C. Aoun
E. Davies
Folly Consulting
April 24, 2010

GIST State Machine
draft-ietf-nsis-ntlp-statemachine-10.txt

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet- Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 10, 2010.

Copyright Notice

Abstract

This document describes state machines for the General Internet Signaling Transport (GIST). The states of GIST nodes for a given f ow and their transitions are presented in order to illustrate how GIST may be implemented.

Table of Contents

1. Introduction

The state machines described in this document are illustrative of how the GIST protocol defined in [1] ma be implemented for the GIST nodes in different locations of a fl w path. Where there are differences - [1] is authoritative. The state machines are informative only. Implementations may achieve the same results using different methods.

There are two types of possible entities for GIST signaling:

- GIST querying node - GIST node that initiates the discovery of the next peer;

- GIST responding node - GIST node that is the discovered next peer;

We d escribe a set of state machines for these entities to illustrate how GIST 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 giv en 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 specifi 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 a y) 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 xecuting 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 xcept 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 alue.

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 documen 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 th 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 = 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 Quer , Response, Confirm

State machine represents handling of GIST messages that match a Message Routing State´s MRI, NSLPID and SID and with no protocol errors. Separate parallel instances of the state machines should handle messages for different Message Routing States.

The state machine states represent the upstream/downstream peers states of the Message Routing State.

For simplification not all objects included in a message are sh wn. Only those that are significant for th case are shown. State machines do not present handling of messages that are not significant for managemen of the states.

The state machines presented in this document do not cover all functions of a GIST node. Functionality of message forwarding, transmission of NSLP data without MRS establishment and providing of the received messages to the appropriate MRS, we refer as "Lower level pre-processing" step. Pre-processing provides to the appropriate MRS state machine only the messages which are matched against waiting Query/Response cookies, or established MRS MRI+NSLPID+SID primary key. This is presented by "rx_*" events in the state machines.

Management of Messaging Associations (MA) is considered in the document via procedures, events and variables, which describe MA interaction with the MRS state machines. A state machine for MA management is not explicitly presented.

5.1 Common Procedures

Tx_Query:
Transmit of Query message

Tx_Response:
Transmit of Response message

Tx_Confirm
Transmit of Confirm messag

Tx_Data:
Transmit of Data message

Tg_MessageStatus:
NSLP/GIST API message informing NSLP application for unsuccessful delivery of a message

Tg_RecvMsg:
NSLP/GIST API message that provides received message to the NSLP application

Tg_NetworkNotification
NSLP/GIST API message that informs NSLP application for change in MRS

Install downstream/upstream MRS:
Install new Message Routing State and save the corespoding peer state info (IP address and UDP port or pointer to the used MA) for the current Message Routing State or update the coresponding peer state info.

Delete MRS:
Delete installed downstream/upstream peer´s info for the current Message Routing State and delete the Message Routing State if required.

Refresh MRS:
Refreshes installed MRS.

Queue NSLP info:
Save NLSP messages in a queue until conditions for their sending are present, e.g. a required MA

association is established.

CheckPeerInfo:
The sender of the received data message is matched against the installed peer info in the MRS.

Delete MA:
Delete/disconnect used MA.

Stop using shared MA:
Stop using shared MA. If the shared MA is no more used by any other MRSs, it depends on the local policy whether it is deleted or kept.

Tg_Establish_MA:
Trigers establishment of a new MA.

Start/Restart a timer variable (Section 5.3):
Start/Restart of a certain timer.

Install/Update/Delete UpstreamPeerInfo variable (Section 5.3):
Management of upstream peer info in responding node state machine.

5.2 Common Events

Rx_Query:
Receive of Query message

Rx_Response:
Receive of Response message

Rx_Confirm
Receive of Confirm messag

Rx_Data:
Receive of Data message

Tg_SendMsg:
NSLP/GIST API message from NSLP application that requests transmission of a NSLP message.

Tg_SetStateLifetime(time_period):
NSLP/GIST API message providing info for the lifetime of a Routing State (RS), required by the application. "Time_period = 0" represents the cancellation of established RSs/MAs, invoked by the NSLP application.

Tg_InvalidRoutingState:
NSLP/GIST API notification from NSLP application for path change

Tg_ERROR:
General Error event / system level error.

Tg_MA_Established:
A new MA has been successfully established.

Tg_MA_Error:
Error event with used MA.

Timeout a timer variable (Section 5.3):
Timeout of a certain timer.

5.3 Common Variables

Variables listed in this section are defined as

- Specific information carried in the rece ved messages.

- Conditions that are results of processes not defined in the state machine model

State machine logic is based on these general conditions and message parameters.

The type of mode and destination info is determined by NSLP application parameters and local GIST policy. Here it is represented by the common variables Dmode, Cmode and MAinfo.

Cmode:
The message MUST be transmitted in Cmode. This is specified by "Message transfer attri utes" set by NSLP application to any of the following values:

"Reliability" is set to TRUE.

"Security" is set to values that request secure handling of a message.

"Local processing" is set to values that require services offered by Cmode (e.g., congestion control). [1]

Dmode:
The message MUST be transmitted in Dmode. This is specified by local poli y rules and in case that the "Message transfer attributes" are not set by NSLP application to any of the following values:

"Reliability" is set to TRUE.

"Security" is set to values that request special security handling of a message.

"Local processing" is set to values that require services offered by Cmode [1]

MAinfo:
GIST message parameters describing the required MA or proposed MA, e.g., "Stack-proposal" and "Stack-Configuration-Data" [1]

NSLPdata:

NSLP application data.

RespCookie:
Responder Cookie that is being sent by the responding node with the Response message in case that its local policy requires a confirmation from the querying node

ConfirmRequired
Indicator that a Confirm message is required by the local poli y rule for installation of a new MRS.

NewPeer:
Indicator that a Response message is received from new responding peer.

MAexist:
Indicator that an existing MA will be reused in data transfer between peers.

UpstreamPeerInfo:
Upstream peer info that is saved in an established MRS.

T_Inactive_QNode:
Message Routing State lifetime timer in querying node

T_Expired_RNode:
Message Routing State lifetime timer in responding node

T_Refresh_QNode:
Message Routing State refresh timer in querying node

T_No_Response:
Timer for the waiting period for Response message in querying node

T_No_Confirm
Timer for the waiting period for Confirm message in responding nod

No_MRS_Installed:
Data sent by responding node via a Response message that indicates loss of Confirm message

6. State machines

The following section presents the state machine diagrams of GIST peers. The document is issued in two formats - .pdf and .txt.

In the .pdf document, the state machine diagrams are depicted in details in this section. All state machine information (triggering event, action taken and variables status) is visualized on the diagrams.

In the .txt document, state machine diagrams depict only transition numbers. Following each diagram is a list of state transition descriptions. Complete transition details (triggering event, action taken and variables status) are given in state transition tables in Appendix A.

Please use the .pdf version whenever possible. It is the clearer representation of the state machine. In case of a difference between the two documents, please refer to the .pdf version.

6.1 Diagram notations

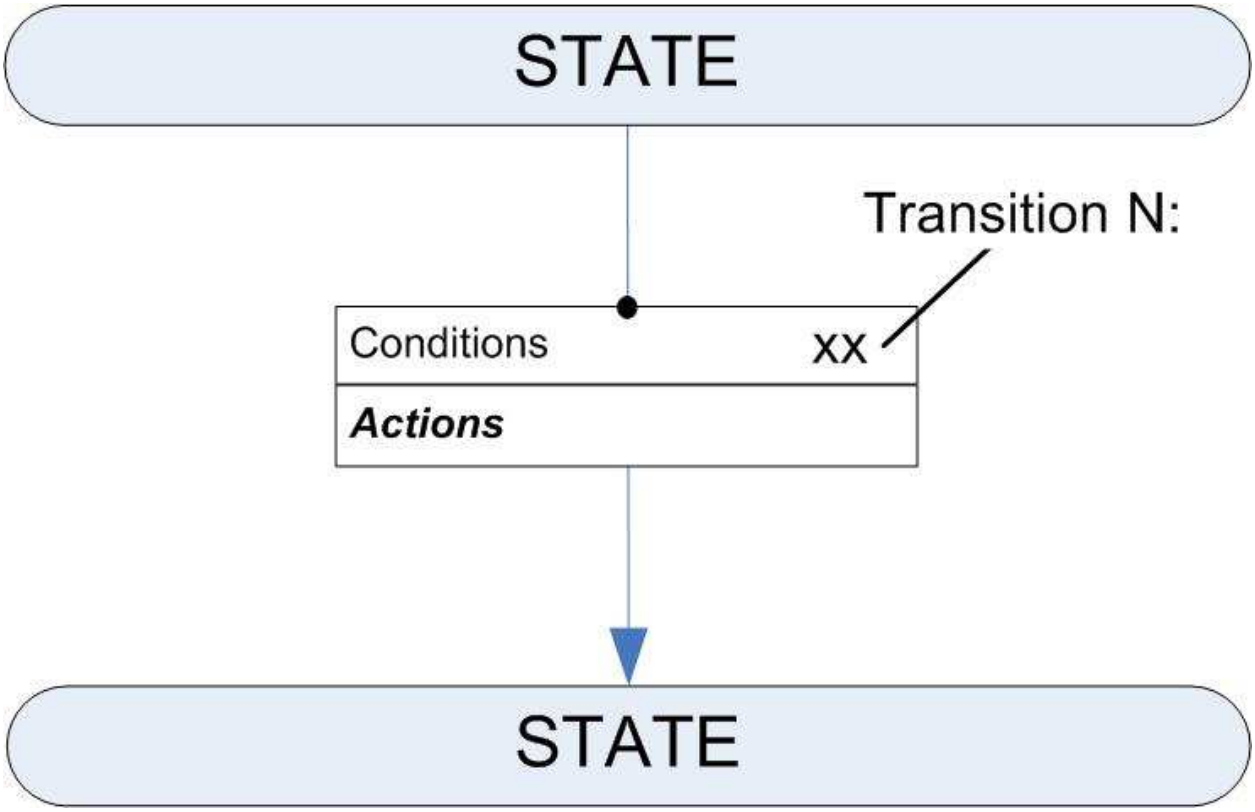


Figure 1: Diagram notations

6.2 State machine for GIST querying node

GIST querying node state machine diagram is depicted below. Transition descriptions follow.

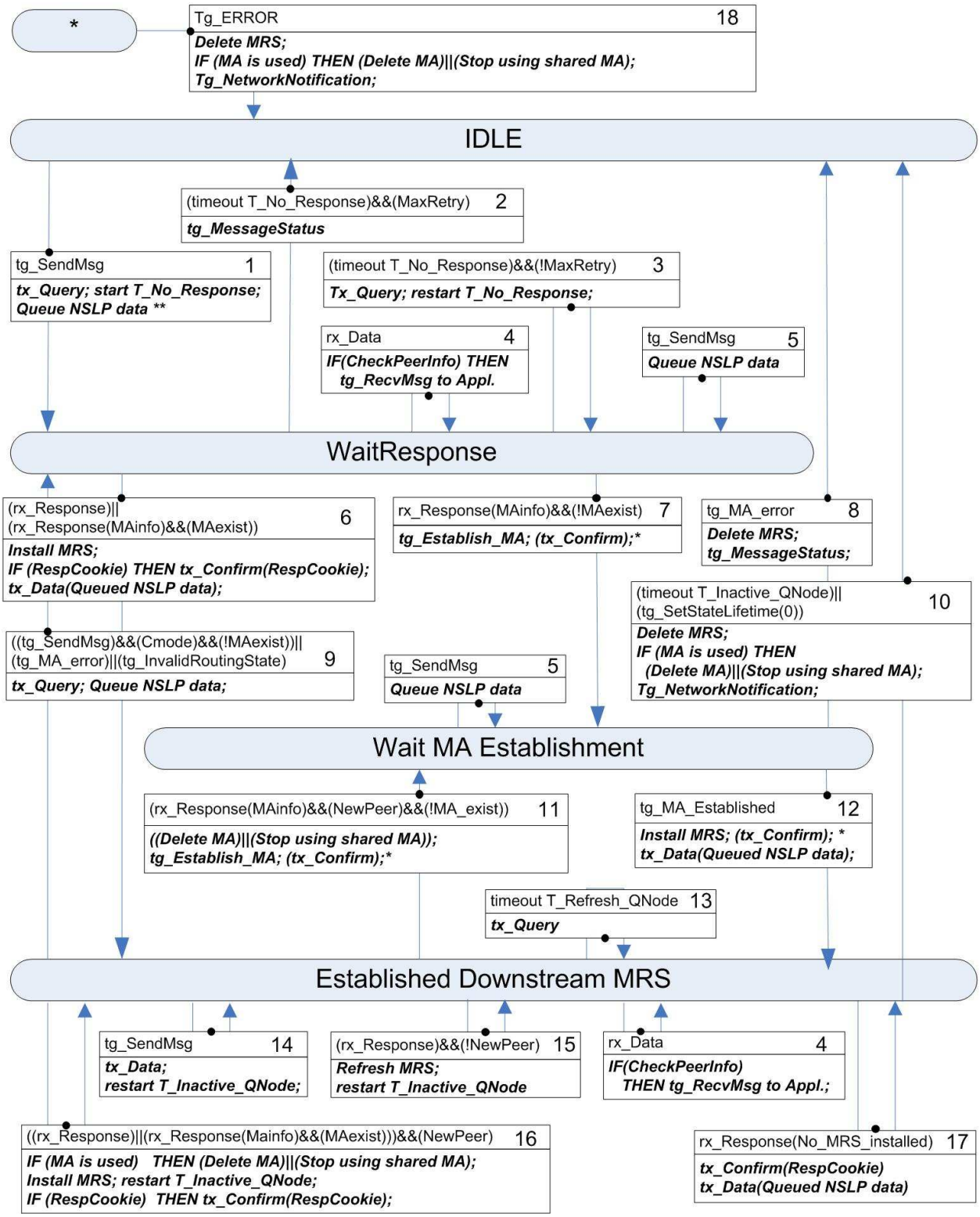

*
Tg_ERROR 18
Delete MRS;
IF (MA is used) THEN (Delete MA)||(Stop using shared MA);
Tg_NetworkNotification;
IDLE
(timeout T_No_Response)&&(MaxRetry) 2
tg_MessageStatus
tg_SendMsg 1
tx_Query; start T_No_Response;
Queue NSLP data **
(timeout T_No_Response)&&(!MaxRetry) 3
Tx_Query; restart T_No_Response;
rx_Data 4
IF(CheckPeerInfo) THEN
tg_RecvMsg to Appl.
tg_SendMsg 5
Queue NSLP data
WaitResponse
(rx_Response)||
(rx_Response(MAinfo)&&(MAexist)) 6
Install MRS;
IF (RespCookie) THEN tx_Confirm(RespCookie);
tx_Data(Queued NSLP data);
rx_Response(MAinfo)&&(!MAexist) 7
tg_Establish_MA; (tx_Confirm);*
tg_MA_error 8
Delete MRS;
tg_MessageStatus;
(timeout T_Inactive_QNode)||
(tg_SetStateLifetime(0)) 10
Delete MRS;
IF (MA is used) THEN
(Delete MA)||(Stop using shared MA);
Tg_NetworkNotification;
((tg_SendMsg)&&(Cmode)&&(!MAexist))||
(tg_MA_error)||(tg_InvalidRoutingState) 9
tx_Query; Queue NSLP data;
tg_SendMsg 5
Queue NSLP data
Wait MA Establishment
(rx_Response(MAinfo)&&(NewPeer)&&(!MA_exist)) 11
((Delete MA)||(Stop using shared MA));
tg_Establish_MA; (tx_Confirm);*
tg_MA_Established 12
Install MRS; (tx_Confirm); *
tx_Data(Queued NSLP data);
timeout T_Refresh_QNode 13
tx_Query
Established Downstream MRS
tg_SendMsg 14
tx_Data;
restart T_Inactive_QNode;
(rx_Response)&&(!NewPeer) 15
Refresh MRS;
restart T_Inactive_QNode
rx_Data 4
IF(CheckPeerInfo)
THEN tg_RecvMsg to Appl.;
((rx_Response)||(rx_Response(Mainfo)&&(MAexist)))&&(NewPeer) 16
IF (MA is used) THEN (Delete MA)||(Stop using shared MA);
Install MRS; restart T_Inactive_QNode;
IF (RespCookie) THEN tx_Confirm(RespCookie);
rx_Response(No_MRS_installed) 17
tx_Confirm(RespCookie)
tx_Data(Queued NSLP data)

Figure 2: GIST Querying Node State Machine

1**) Initial request from NSLP application is received, which triggers Query messages requesting either Dmode or Cmode. Depending on nodes local policy NSLP data might be piggybacked in the Query requesting Dmode. Query may carry MAinfo if Cmode transport is needed.
2) T_No_Response timer expires and maximum number of retries has been reached. NSLP application is notified for the GIST peer disc very failure.
3) T_No_Response timer expires. Query is resent.
4) Data message is received. It is checked if its sender matches the installed downstream peer info in the MRS and then processed. In WaitResponse state, this event might happen in the process of MA upgrade, when the downstream peer is still not aware of establishment of the new MA.
5) NSLP application provides data for sending. NSLP data is queued, because downstream peer is not discovered or required MA is still not established.
6) Response message is received. If Dmode connection is requested or available MA can be reused for requested Cmode, the MRS is established.
7*) Response message is received. If Cmode connection must be established and there is no available MA to be reused, MA establishment is initiated and waited to be completed.
8) A MA establishment fails. NSLP application is notified for unsuccessful message del very.
9) NSLP application provides data for sending and requested transport parameters require upgrade of established MRS from Dmode/Cmode to Cmode. Or NSLP application notifies GIST for pat change. As a result downstream GIST peer discovery is initiated.
10) MRS lifetime expires or NSLP application notifies that MRS is no longer needed. MRS is deleted. If no needed, MA is deleted, too. NSLP application is notified for MRS change
11*) Path change detected as a Response message from a new downstream GIST peer is received. A new MA must be established for requested Cmode.
12*) A new MA is established. MRS is installed. Queued NSLP data is sent.
13) T_Refresh_QNode timer expires. Query message is sent.
14) NSLP application provides data for sending. It is sent via Data message towards downstream GIST peer.
15) Response message from the downstream GIST peer is received. The peer is not changed. MRS is refreshed (T_Refresh_QNode timer is restarted).
16) Path change detected as a Response message from a new downstream GIST peer is received. Dmode is requested or existing MA can be reused for requested Cmode.
17) Responding peer indicates that it has not received a Confirm message and it has no established upstrea MRS. Confirm message is resent
18) General error or system level error occurs. MRS is deleted. If not needed, MA is deleted, too. NSLP application is notified for the MRS change

Remarks:
*) Response and Confirm messages might be sent either in Dmode or Cmode, before or after M establishment depending on nodes local 3-way handshake policy and the availability of MAs to be reused. See [1] for details.
**) Depending on GIST local policy, NSLPdata might be send as payload of Query and Confirm message (piggybacking).

6.3 State machine for GIST responding node

GIST responding node state machine diagram is depicted below. Transition descriptions follow.

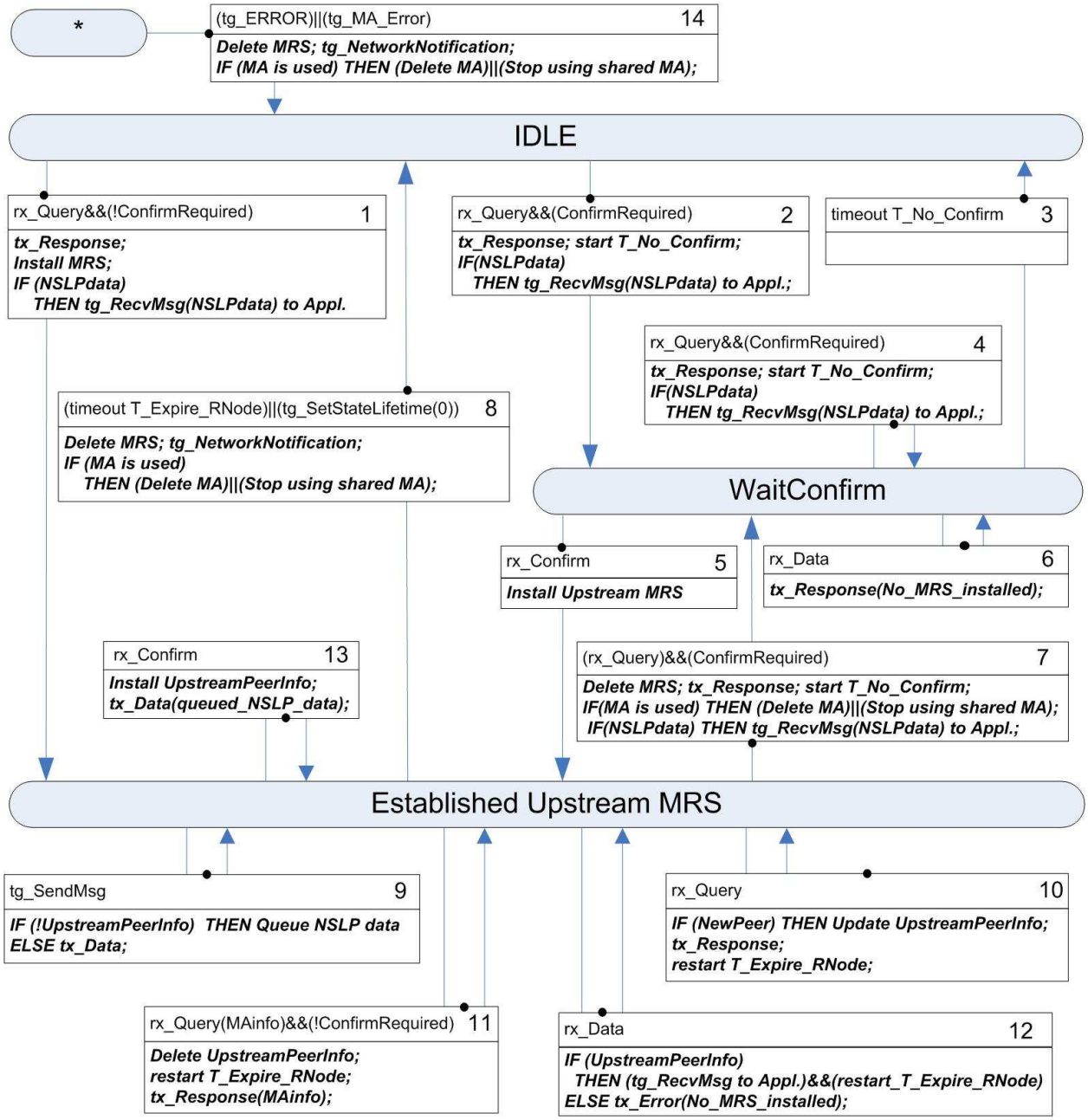


Figure 3: GIST Responding Node State Machine

1) A Query message is received. MRS is installed immediately, because local policy permits it. Query message might carry piggybacked NSLP data which is provided to the NSLP application.
2) A Query message is received. Local policy requires explicit Confirm message for MRS installation. Quer message might carry piggybacked NSLP data which is provided to the NSLP application.
3) T_No_Confirm timer xpires. Note that all cases of lost handshake GIST messages are handled only by GIST querying node via resend of Query message.
4) A Query message is received again. This means that sent Response message has not been received by upstream GIST peer. Response message is resent.
5) A Confirm message is rece ved which causes installation of the upstream MRS.
6) In case of lost Confirm message, data messages might be rece ved from the upstream GIST node (it is unaware of the lost Confirm message). Response indicating the loss of the Confirm is sent back to t upstream GIST node.
7) A Query message is received with request for change of the used connection mode (from Dmode/Cmode to better Cmode) or from new upstream GIST node. Local policy requires explicit Confirm messag for MRS installation.
8) MRS lifetime expires or NSLP application notifies that MRS is no longer needed. MRS is deleted. If use and not needed, MA is deleted, too. NSLP application is notified for MRS change
9) NSLP application provides data for sending. NSLP data is sent if discovery process is successfully accomplished or it is queued if Confirm message is still xpected to confirm establishment of a MA
10) A Query message is received. If it is sent from a new upstream GIST node then there is a path change. Local policy does not need explicit Confirm message for MRS installation. MRS data is updated
11) A Query message is received with request for change of the used connection mode (from Dmode/Cmode to better Cmode). Local policy does not need explicit Confirm message for MRS installation. MR data is updated.
12) A Data message is received. Data messages are accepted only if complete MRS is installed, e.g., there is installed upstream peer info. If not, then Confirm message is xpected and Data message is not accepted. Response indicating the loss of the Confirm is sent back to the upstream GIST node
13) A Confirm message is rece ved. It accomplishes assignment of an existing MA (or establishment of a new MA) needed for data transferring between peers. The information for the used MA is installed as upstream peer info.
14) General error or system level error occurs. MRS is deleted. If not needed, MA is deleted, too. NSLP application is notified for MRS change

7. Security Considerations

This document does not raise new security considerations. Security considerations are addressed in GIST specification [1] and in [6]

8. IANA Considerations

This document has no actions for IANA.

9. Acknowledgments

The authors would like to thank Christian Dickmann who contributed to refining of the state machine

The authors would like to thank Robert Hancock, Ingo Juchem, Andreas Westermaier, Alexander Zrim,

Julien Abeille Youssef Abidi and Bernd Schloer for their insightful comments.

10. References

10.1. Normative References

[1] Schulzrinne, H., "GIST: General Internet Signaling Transport", draft-ietf-nsis-ntlp-20 (work in progress), December 2009.

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

10.2. Informative References

[3] V ollbrecht, J., Eronen, P., Petroni, N., and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator", RFC4137, August 2005.

[4] Institute of Electrical and Electronics Engineers, "Standard for Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE 802-1X-2004, December 2004.

[5] F ajardo, V., Ohba, Y. and R. Marin-Lopez, "State Machines for Protocol for Carrying Authentication for Network Access (PANA)", RFC 5609, August 2009.

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

Appendix A. State transition tables

State transition tables given below represent the state diagrams in ASCII format. Please use the .pdf version whenever possible. It is the clearer representation of the state machine.

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 transition tables for GIST querying node

Please refer to state machine diagram on Figure 2.

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

+Transition
|  |Condition               |Action                    |State
V--+------------------------+--------------------------+-----------
1) |tg_SendMsg              |tx_Query                  |Wait
** |                        |start T_No_Response       |Response
   |                        |Queue NSLP data           |
   |                        |                          |
18)|Tg_ERROR                |Delete MRS                |IDLE
   |                        |IF (MA is used)           |
   |                        |  ((Delete MA)||          |
   |                        |  (Stop using shared MA))|
   |                        |Tg_NetworkNotification    |
   |                        |                          |
---+------------------------+--------------------------+-----------


-----------
State: WaitResponse
-----------

+Transition
|  |Condition               |Action                    |State
V--+------------------------+--------------------------+-----------
2) |(timeout T_No_Response) |tg_MessageStatus          |IDLE
   |&&(MaxRetry)            |                          |
   |                        |                          |
3) |(timeout T_No_Response) |Tx_Query                  |Wait
   |&&(!MaxRetry)           |restart T_No_Response     |Response
   |                        |                          |
4) |rx_Data                 |IF(CheckPeerInfo)         |Wait
```

```
   |                        |      tg_RecvMsg to Appl.|Response
   |                        |                         |
5) |tg_SendMsg              |Queue NSLP data          |Wait
   |                        |                         |Response
   |                        |                         |
6) |rx_Response)||          |Install MRS              |Established
   |(rx_Response(MAinfo)&&  |IF (RespCookie)          |Downstream
   |(MAexist))              |   tx_Confirm(RespCookie)|MRS
   |                        |tx_Data(Queued NSLP data)|
   |                        |                         |
7) |rx_Response(MAinfo)&&   |tg_Establish_MA          |Wait MA
*  |(!MAexist)              |(tx_Confirm)             |Establish.
   |                        |                         |
   |                        |                         |
18)|Tg_ERROR                |(Delete MRS)             |IDLE
   |                        |IF (MA is used)          |
   |                        |  ((Delete MA)||         |
   |                        |  (Stop using shared MA))|
   |                        |Tg_NetworkNotification   |
   |                        |                         |
---+------------------------+-------------------------+-----------
```

```
-----------
State: Established Downstream MRS
-----------

+Transition
|  |Condition               |Action                   |State
V--+------------------------+-------------------------+-----------
4) |rx_Data                 |IF(CheckPeerInfo)        |Established
   |                        |      tg_RecvMsg to Appl.|Downstream
   |                        |                         |MRS
   |                        |                         |
9) |((tg_SendMsg)&&(Cmode)&&|tx_Query                 |Wait
   |(!MAexist))||           |Queue NSLP data          |Response
   |(tg_MA_error)||         |                         |
   |(tg_InvalidRoutingState)|                         |
   |                        |                         |
10)|(timeout T_Inactive_    |Delete MRS               |IDLE
   |                QNode)|||IF (MA is used)          |
   |(tg_SetStateLifetime(0))|   (Delete MA)||         |
   |                        |   (Stop using shared MA)|
   |                        |Tg_NetworkNotification   |
   |                        |                         |
11)|(rx_Response(MAinfo)&&  |((Delete MA)||           |Wait MA
```

*	(NewPeer)&&(!MA_exist))	(Stop using shared MA)) tg_Establish_MA (tx_Confirm)	Establish.
13)	timeout T_Refresh_QNode	tx_Query	Established Downstream MRS
14)	tg_SendMsg	tx_Data restart T_Inactive_QNode	Established Downstream MRS
15)	(rx_Response)&& (!NewPeer)	Refresh MRS restart T_Inactive_QNode	Established Downstream MRS

16)	(rx_Response)||
	(rx_Response(Mainfo)&&
	(MAexist)))&&(NewPeer)
		IF (MA is used)
		(Delete MA)||
		(Stop using shared MA)
		Install MRS
		restart T_Inactive_QNode
		IF (RespCookie)
		tx_Confirm(RespCookie)
			Established
			Downstream
			MRS

17)	rx_Response(No_MRS_ installed)	tx_Confirm(RespCookie) tx_Data(Queued NSLP data)	Established Downstream MRS
18)	Tg_ERROR	(Delete MRS) IF (MA is used) ((Delete MA)|| (Stop using shared MA)) Tg_NetworkNotification	IDLE

State: Wait MA Establishment

+Transition V	Condition	Action	State
5)	tg_SendMsg	Queue NSLP data	Wait MA Establish.
8)	tg_MA_error	Delete MRS	IDLE

```
   |                        |tg_MessageStatus         |
   |                        |                         |
12)|tg_MA_Established       |Install MRS              |Established
*  |                        |(tx_Confirm)             |Downstream
   |                        |tx_Data(Queued NSLP data)|MRS
   |                        |                         |
18)|Tg_ERROR                |Delete MRS               |IDLE
   |                        |IF (MA is used)          |
   |                        |  ((Delete MA)||         |
   |                        |  (Stop using shared MA))|
   |                        |Tg_NetworkNotification   |
   |                        |                         |
---+------------------------+-------------------------+-----------
```

A.2. State transition tables for GIST responding node

Please refer to state machine diagram on Figure 3.

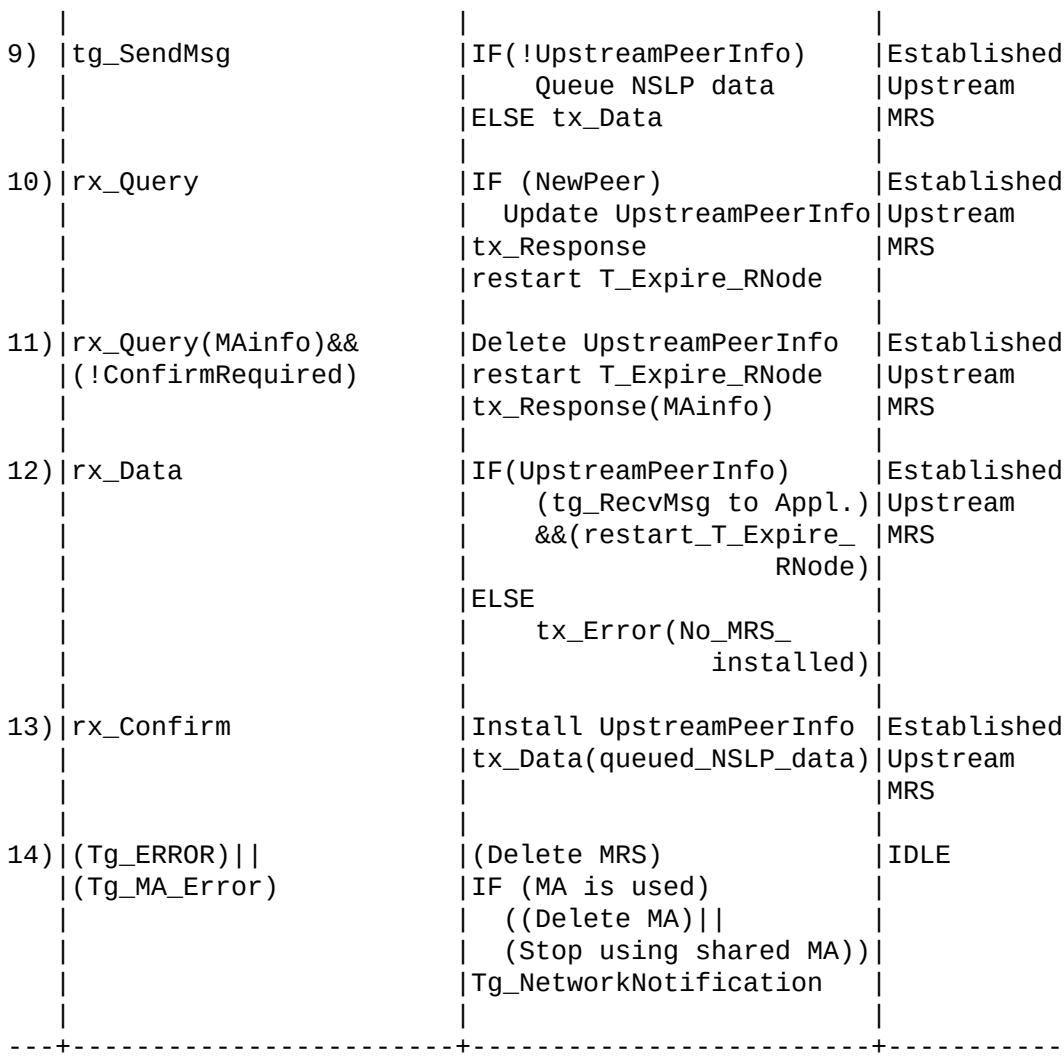
```
-----------
State: IDLE
-----------

+Transition
|  |Condition               |Action                   |State
v--+------------------------+-------------------------+-----------
1) |rx_Query&&              |tx_Response              |Established
   |(!ConfirmRequired)      |Install MRS              |Upstream
   |                        |IF(NSLPdata)             |MRS
   |                        |     tg_RecvMsg(NSLPdata)|
   |                        |                 to Appl.|
   |                        |                         |
2) |rx_Query&&              |tx_Response              |Wait
   |(ConfirmRequired)       |start T_No_Confirm       |Confirm
   |                        |IF(NSLPdata)             |
   |                        |     tg_RecvMsg(NSLPdata)|
   |                        |                 to Appl.|
   |                        |                         |
---+------------------------+-------------------------+-----------



-----------
State: WAIT CONFIRM
-----------

+Transition
|  |Condition               |Action                   |State
```

3)	timeout T_No_Confirm		IDLE
4)	rx_Query&& (ConfirmRequired)	tx_Response start T_No_Confirm IF(NSLPdata) tg_RecvMsg(NSLPdata) to Appl.	Wait Confirm
5)	rx_Confirm	Install Upstream MRS	Established Upstream MRS
6)	rx_Data	tx_Response(No_MRS_ installed)	Wait Confirm
14)	(Tg_ERROR)|| (Tg_MA_Error)	(Delete MRS) IF (MA is used) ((Delete MA)|| (Stop using shared MA)) Tg_NetworkNotification	IDLE

State: Established Upstream MRS

---

7)	(rx_Query)&& (ConfirmRequired)	Delete MRS tx_Response start T_No_Confirm IF(MA is used) (Delete MA)|| (Stop using shared MA) IF(NSLPdata) tg_RecvMsg(NSLPdata) to Appl.	Wait Confirm
8)	(timeout T_Expire_RNode)||| (tg_SetStateLifetime(0))	Delete MRS tg_NetworkNotification IF(MA is used) (Delete MA)|| (Stop using shared MA)	IDLE


9) tg_SendMsg
IF(!UpstreamPeerInfo)
Queue NSLP data
ELSE tx_Data
Established Upstream MRS
10) rx_Query
IF (NewPeer)
Update UpstreamPeerInfo
tx_Response
restart T_Expire_RNode
Established Upstream MRS
11) rx_Query(MAinfo)&& (!ConfirmRequired)
Delete UpstreamPeerInfo
restart T_Expire_RNode
tx_Response(MAinfo)
Established Upstream MRS
12) rx_Data
IF(UpstreamPeerInfo)
(tg_RecvMsg to Appl.)
&&(restart_T_Expire_ RNode)
ELSE
tx_Error(No_MRS_ installed)
Established Upstream MRS
13) rx_Confirm
Install UpstreamPeerInfo
tx_Data(queued_NSLP_data)
Established Upstream MRS
14) (Tg_ERROR)|| (Tg_MA_Error)
(Delete MRS)
IF (MA is used)
((Delete MA)||
(Stop using shared MA))
Tg_NetworkNotification
IDLE

Authors´ Addresses

Tseno Tsenov
Sofia, Bul aria

Email: tseno.tsenov@mytum.de

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Email: Hannes.Tschofenig@nsn.com

Xiaoming Fu (editor)
University of Goettingen
Computer Networks Group
Goldschmidtstr. 7
Goettingen 37077
Germany

Email: fu@cs.uni-goettingen.de

Cedric Aoun
Paris, France

Email: cedric@caoun.net

Elwyn B. Davies
Folly Consulting
Soham, Cambs, UK

Phone: +44 7889 488 335
Email: elwynd@dial.pipex.com