NSIS
Internet-Draft
Expires: January 9, 2008

T. Tsenov
H. Tschofenig
Nokia Siemens Networks
X. Fu
Univ. Goettingen
C. Aoun
E. Davies
Folly Consulting
July 8, 2007

GIST State Machine
draft-ietf-nsis-ntlp-statemachine-04.txt

Status of this Memo

Copyright Notice

Abstract

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

Table of Contents

1. Introduction

This document describes the state machines for GIST [1], trying to show how GIST can be implemented to support its deployment. The state machines described in this document are illustrative of how the GIST protocol defined in [1] may be implemented for the GIST nodes in different locations of a flow path. Where there are differences [1] are 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 describe 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 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

MA
: Messaging Association

Upstream/Downstream MRS
: Message Routing State with upstream/downstream peer state info

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

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

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

!
: Logical NOT operator.

&&
: Logical AND operator.

||
: Logical OR operator.

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

{ statement 1, ... statement N }
: Compound statement. Braces are used to group statements that are executed together as if they were a single statement.

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

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

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

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

++
: Increment the preceding integer operator by 1.

+
: Arithmetic addition operator.

&
: Bitwise AND operator.

5. Common Rules

Throughout the document we use terms defined in the [1], such as Query, 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 shown. Only those that are significant for the case are shown. State machines do not present handling of messages that are not significant for management of the states.

Presented in this document state machines do not cover all functions of a GIST node. Functionality of message forwarding, ROA processing, 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. The interaction of this step with the presented here state machines is defined as follows:

Pre-processing provides to the appropriate MRS FSM only the messages which are matched against waiting Query/Response cookies, or established MRS MRI+NSLPID primary key. This is presented by "rx_*" events in the state machines.

5.1 Common Procedures

Tg_SendMsg:
NSLP/GIST API message that request transmission of a NSLP message.

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

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

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

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

Tx_Query:
Transmit of Query message in Dmode

Tx_Response_Dmode:
Transmit of Response message in Dmode

Tx_Confirm_Dmode:
Transmit of Confirm message in Dmode

Rx_Query_Dmode:
Receive of Query message in Dmode

Rx_Response_Dmode:
Receive of Response message in Dmode

Rx_Confirm_Cmode:
Receive of Confirm message in Dmode

Tx_Response_Cmode:
Transmit of Response message in Cmode (via MA)

Tx_Confirm_Cmode:
Transmit of Confirm message in Cmode (via MA)

Rx_Response_Cmode:
Receive of Response message in Cmode (via MA)

Rx_Confirm_Cmode:
Receive of Confirm message in Cmode (via MA)

Queue NSLP msg info:
Save NLSP messages in a queue until a required MA association is established

Tx_Msg_Cmode:
Transmit message in Cmode (via MA)

Rx_Msg_Cmode:
Receive message in Cmode (via MA)

Tx_Msg_Dmode:
Transmit message in Dmode

Rx_Msg_Dmode:
Receive message in Dmode

TIMEOUT_MRSlifetime:
Expiration of the lifetime timer of the upstream/downstream peer state info of the Message Routing State.

TIMEOUT_Refresh:
Refresh interval timer expiration

TIMEOUT_WaitResponse:
Expiration of Timer for the waiting period for Response message.

TIMEOUT_WaitConfirm:
Expiration of Timer for the waiting period for Confirm message.

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.

Establish MA:
Establish Message Association (MA) between current node and its downstream peer

Established MA:
A Message Association (MA) is established between the current node and its upstream peer. The initiator for the establishment is the upstream peer. Re-use existing MA: An existing MA between the current node and its peer is re-used.

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.

REFRESH MRS:
Refreshes installed MRS.

Tg_MA_Error:
Error event with used MA.

Tg_PathChange:
External event for Path change detected.

Tg_Establish_MA:
Trigers establishment of MA.

Tg_MA_Established:
MA has been successfully established.

Tg_ERROR:
General Error event / system level error.

No_MRS_Installed:
Error response, send by the Responding node indicating lost Confirm message.

5.2 Common Variables

It is assumed that the type of mode and destination info (which need to be taken from the application parameters and local GIST policy)is provided. This is represented by the common variables Dmode, Cmode, MAinfo, MApresent and Refresh.

Cmode:
The message MUST be transmitted in Cmode. This is specified by "Message transfer attributes" set 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 policy rules and in case that the "Message transfer attributes" are not set 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 "Node-addressing". This list of GIST parameters is not complete. A full mapping is left for future version of the document.

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.

Refresh:
This variable specifies that the message is for refresh purposes.

ConfirmRequired:
TConfirm message is required by the local policy rule for installation of the new MRS.

NewPeer:
Response message is received from new responding peer.

MAexist:
Existing MA will be reused.

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

UpstreamPeerInstalled:
Upstream peer info is installed in the MRS.

5.3 Constants

6. State machines

The following section presents the state machine diagrams of GIST peers.

6.1 Diagram notations

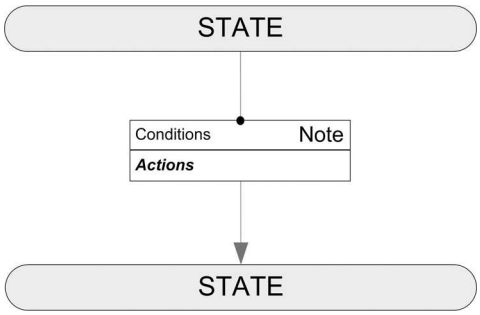


Figure 1: Diagram notations

6.2 State machine for GIST querying node

The following is a diagram of the GIST querying node state machine. Also included is clarification of notation.

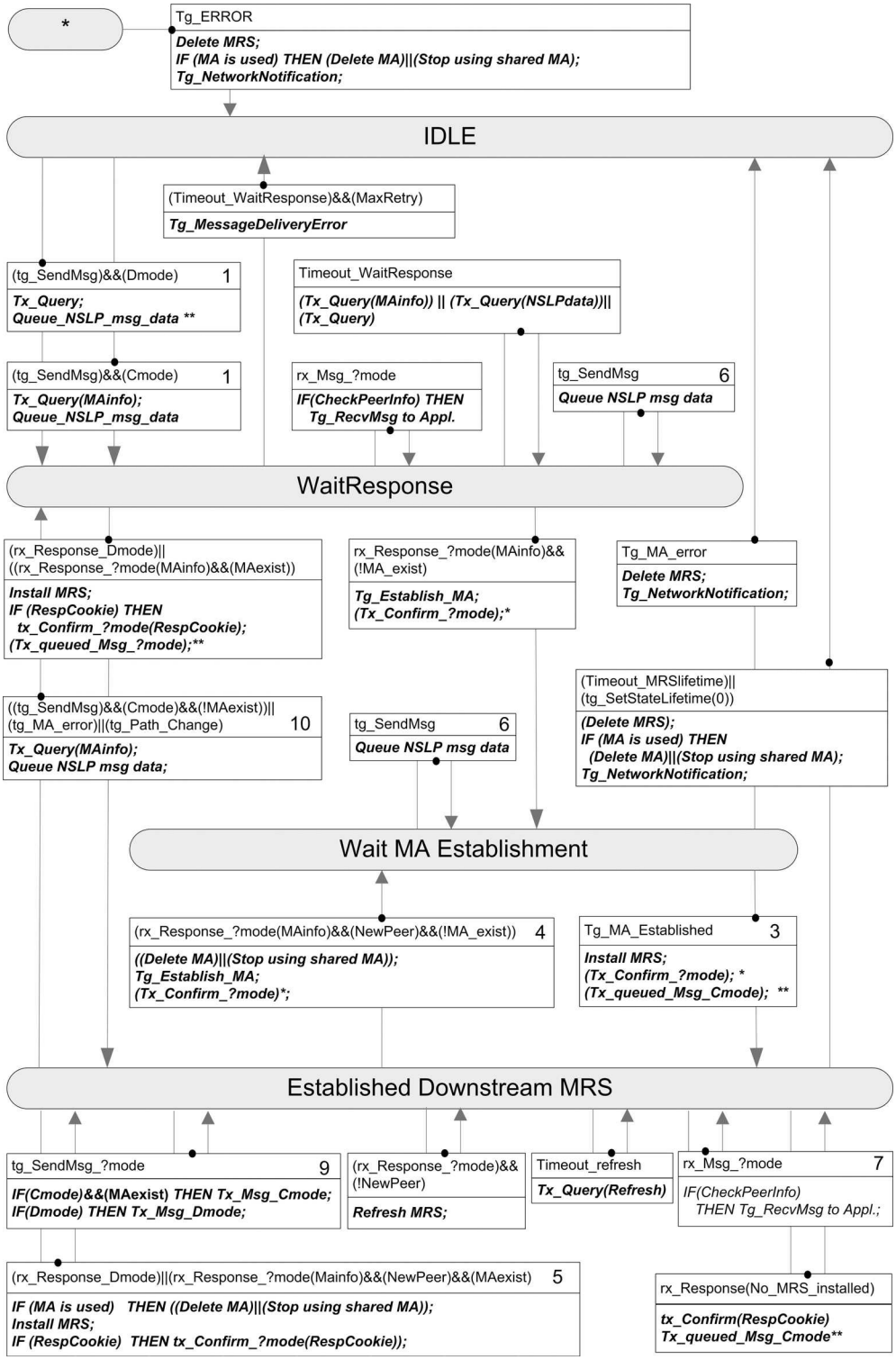


Figure 1: GIST Querying Node State Machine

*) Response and Confirm messages might be send either in Dmode or Cmode, before or after MA establishment depending on node's local 3-way handshake policy and the availability of MAs to be reused. See draft for details.

**) Depending on the local policy NSLPdata might be send as payload of Query and Confirm messages. (piggybacking)

1) Initial request from NSLP are received, which triggers Query messages requesting either D_mode or C_mode. Dependign on local policy of the node, NSLP data might be piggybacked in the Query requesting D_mode.

2) Response message is received. If C_mode connections must be established and there is no available MA to be reused, MA establishment is initiated and waited to be completed. If D_mode connection is requested or available MA can be reused if C_mode is requested the MRS is established.

3) New MA is successfully established and MRS, which will use it, is installed.

4) Path change detected events - local recovery procedure, where new MA must be established for requested C_mode connection. THIS IS VALID ONLY IF THE NODE IS CROSSOVER NODE.

5) Path change detected events - local recovery procedure, where D_mode or C_mode with available MA must be established. THIS IS VALID ONLY IF THE NODE IS CROSSOVER NODE.

6) NSLP data is queued, because downstream peer is not discovered or required MA is still not established.

7) Received Data messages are checked if their sender matches the installed downstream peer info in the MRS and then processed.

8) Received Data messages are checked if their 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.

9) Depending on the requested transport from NSLP and currently established D_mode or C_mode, NSLP message is sent D_mode if D_mode is requested and C_mode if the features of the used MA covers the required transport. (e.g. used MA is reliable and NSLP request reliable but not secure transport)

10) External event notifies for Path Change and discovery procedures is restarted. THIS IS VALID ONLY IF THE NODE IS CROSSOVER NODE OR NSLP requests C_mode transport that is not covered by currently used D_mode or MA (case of MA upgrade) and discovery procedure is restared but current downstream peer info is kept in order to be able to receive messages from it during the upgrade process.

6.3 State machine for GIST responding node

The following is a diagram of the GIST responding node state machine. Also included is clarification of notation.

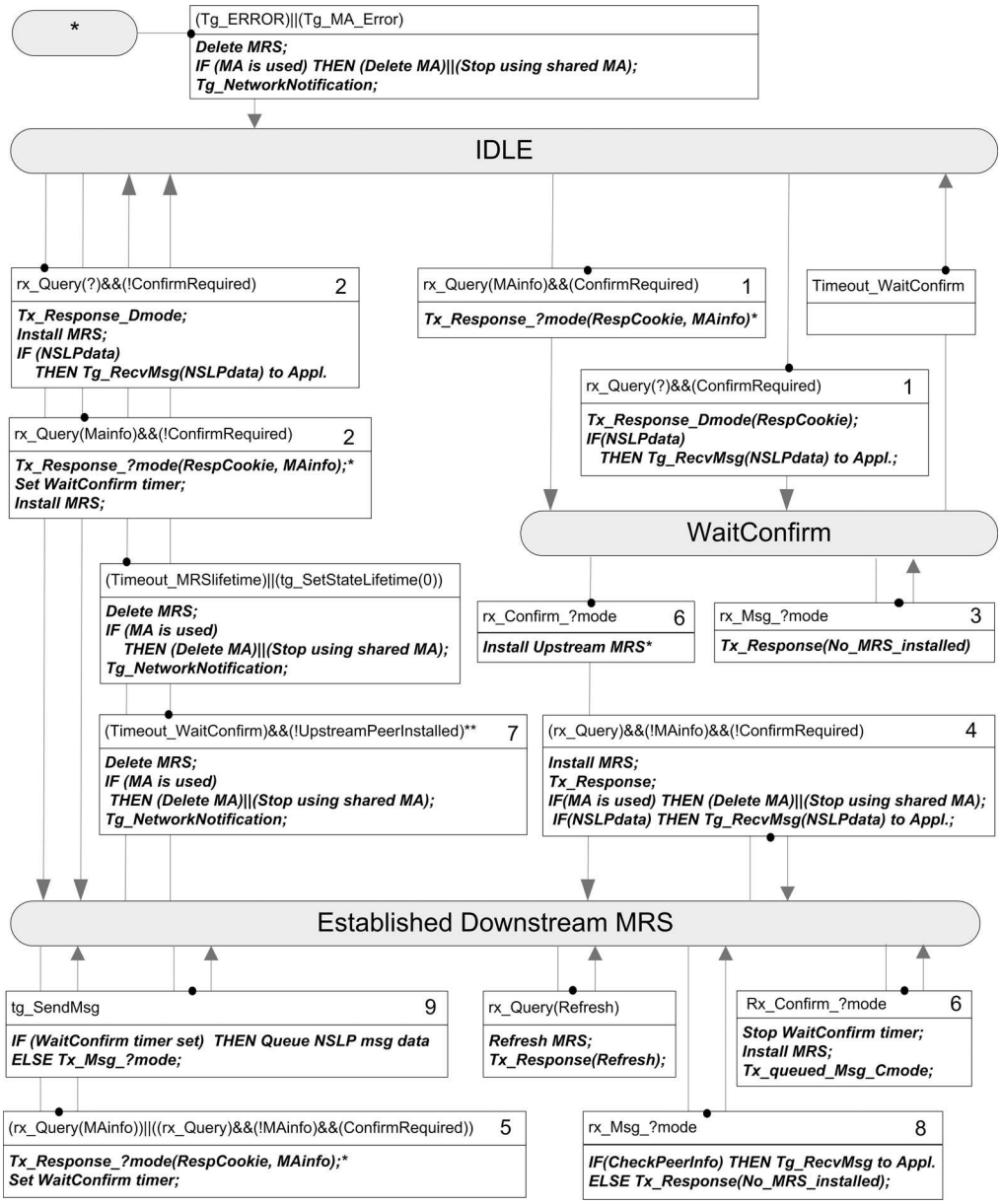


Figure 3: GIST Responding Node State Machine

*) Response and Confirm messages might be send either in Dmode or Cmode depending on node's local 3-way handshake policy and the availability of MAs to be reused. See draft for details.

**) Differentiation between WaitConfirm timer expiration of initial MRS event or MA_upgrade event is

based on the presence of installed peer info in the MRS. If no peer info is installed this is initial MRS establishment.

1) Initial Query messages requesting either D_mode or C_mode connection. In both cases, explicit Confirm message is required for MRS installation, based on the local policy. Query requesting D_mode might carry piggybacked NSLP data.
2) Initial Query messages requesting either D_mode or C_mode connection. In both cases, MRS is installed immediately, based on the local policy. Query requesting D_mode might carry piggybacked NSLP data. In the case of C_mode request, Confirm message is required to confirm the establishment of the used MA.
3) In case of lost Confirm message, data messages might be received from the upstream node (it is unaware of the lost Confirm message). Response indicating the loss of the Confirm is sent back to the upstream node.
4) Event of change of the upstream peer (e.g., path change) with request of D_mode and non-paranoid local policy.
5) Event of request of change of the used connection mode (from D_mode/C_mode to better C_mode), event of change of the upstream peer (e.g., path change) with request for C_mode or D_mode connection and paranoid local policy.
6) Confirm message is received which causes installation of the complete MRS or just installation of the used MA as a upstream peer info.
7) Differentiation between WaitConfirm timer expiration of initial MRS event or MA upgrade/change event is based on the presence of installed peer info in the MRS. If no peer info is installed this is initial MRS establishment and installed MRS must be deleted.
8) Data messages are accepted only if complete MRS is installed, e.g., there is installed upstream peer info. If not then Confirm message is expected and data message will not be accepted but Response indicating the loss of the Confirm is sent back to the upstream node.
9) NSLP message can't be sent upstream if Confirm message is not received and MA is not installed as upstream peer info. They are queued.

8. Security Considerations

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

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

9. Open Issues

We have left for further version of the document the following issues:

1. The FSM that handles the management of a MA is considered in the document (e.g., tg_Establish_MA, tg_MA_established events), but it is not currently explicitely presented. It is left for future version of the document.
2. Functionality of, as referred in the document "Lower level pre- processing" (Section 5), namely message forwarding, RAO processing, transmission of NSLP data without MRS establishment and providing of the received messages to the appropriate MRS is left for future version of the document.
3. Currently we use WaitConfirm state in the Responding node FSM, but following the DoS prevention

approaches for no state installation in the Responding node before receiving of Confirm message, we consider possible removing of this state. This issue requires further investigations.

10. Contributors

Christian Dickmann contributed to refining of the state machine since 01 version.

11. Acknowledgments

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.

12. References

12.1. Normative References

[1] Schulzrinne, H., "GIST: General Internet Signaling Transport", draft-ietf-nsis-ntlp-08 (work in progress), February 2005.

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

11.2. 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.

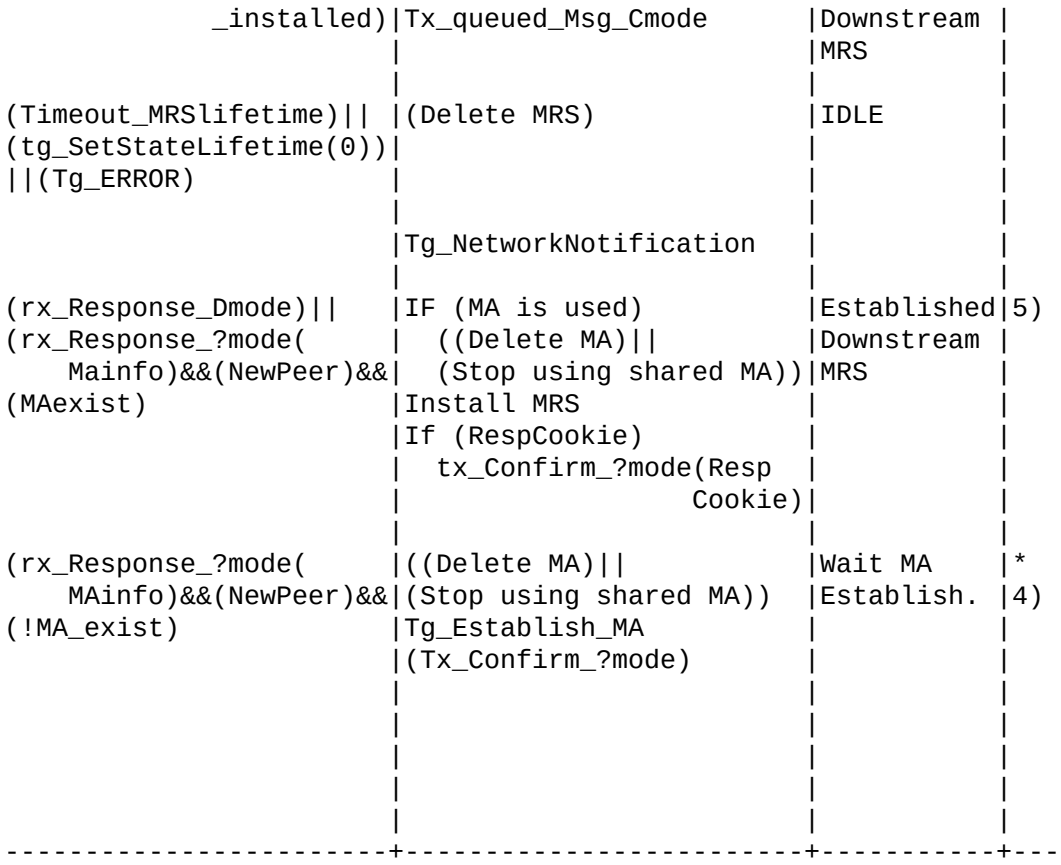
A.1. State machine for GIST querying node (Figure 2)

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

Condition                Action                     State      Note
-------------------------+--------------------------+-----------+---
(tg_SendMsg)&&(Dmode)    |Tx_Query                  |Wait       |1)
                         |Queue_NSLP_msg_data       |Response   |**
                         |                          |           |
(tg_SendMsg)&&(Cmode)    |Tx_Query(MAinfo)          |Wait       |1)
                         |Queue_NSLP_msg_data       |Response   |
                         |                          |           |
-------------------------+--------------------------+-----------+---



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

Condition                Action                     State      Note
-------------------------+--------------------------+-----------+---
(rx_Response_Dmode)||    |Install MRS               |Established|**
((rx_Response_?mode(     |If (RespCookie)           |Downstream |2)
     MAinfo)&&(MAexist))|  tx_Confirm_?mode(Resp   |MRS        |
                         |                   Cookie)|           |
                         |Tx_queued_Msg_?mode       |           |
                         |                          |           |
(rx_Response_?mode(      |Tg_Establish_MA           |Wait MA    |*
   MAinfo)&&(!MA_exist))|(Tx_Confirm_?mode)        |Establish. |2)
                         |                          |           |
rx_Msg_?mode             |IF(CheckPeerInfo)         |Wait       |8)
                         |      Tg_RecvMsg to Appl.|Response   |
                         |                          |           |
tg_SendMsg               |Queue NSLP msg data       |Wait       |6)
                         |                          |Response   |
```

Timeout_WaitResponse	(Tx_Query(MAinfo)) || (Tx_Query(NSLPdata))|| (Tx_Query)	Wait Response	
(Timeout_WaitResponse) &&(MaxRetry)	Tg_MessageDeliveryError	IDLE	
Tg_ERROR	(Delete MRS) IF (MA is used) ((Delete MA)|| (Stop using shared MA)) Tg_NetworkNotification	IDLE	

State: Established Downstream MRS

Condition	Action	State	Note
rx_Msg_?mode	IF(CheckPeerInfo) Tg_RecvMsg to Appl.	Established Downstream MRS	7)
tg_SendMsg_?mode	IF(Cmode)&&(MAexist) Tx_Msg_Cmode IF(Dmode) Tx_Msg_Dmode	Established Downstream MRS	9)
((tg_SendMsg)&&(Cmode)&& (!MAexist))|| (tg_MA_error)|| (tg_Path_Change)	Tx_Query(MAinfo) Queue NSLP msg data	Wait Response	10)
Timeout_refresh	Tx_Query(Refresh)	Established Downstream MRS	
(rx_Response)&& (!NewPeer)	Refresh MRS	Established Downstream MRS	
rx_Response(No_MRS	tx_Confirm(RespCookie)	Established	




```
-----------
State: Wait MA Establishment
-----------

Condition                Action                    State      Note
-----------------------+-------------------------+-----------+---
tg_SendMsg               |Queue NSLP msg data      |Wait MA    |6)
                         |                         |Establish. |
                         |                         |           |
Tg_MA_Established        |Install MRS              |Established|3)
                         |(Tx_Confirm_?mode)       |Downstream |*
                         |(Tx_queued_Msg_Cmode)    |MRS        |**
                         |                         |           |
Tg_MA_error              |Delete MRS               |IDLE       |
                         |Tg_NetworkNotification   |           |
                         |                         |           |
Tg_ERROR                 |(Delete MRS)             |IDLE       |
                         |IF (MA is used)          |           |
```

```
                        |  ((Delete MA)||         |           |
                        |  (Stop using shared MA))|           |
                        |Tg_NetworkNotification   |           |
                        |                         |           |
------------------------+-------------------------+-----------+---
```

Figure 4

A.2. State Machine for GIST responding node (Figure 3)

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

Condition                Action                     State      Note
-------------------------+--------------------------+-----------+---
rx_Query(?)              |Tx_Response_Dmode(Resp    |Wait       |1)
&&(ConfirmRequired)      |                  Cookie)|Confirm    |
                         |IF(NSLPdata)              |           |
                         |Tg_RecvMsg(NSLPdata)      |           |
                         |                 to Appl.|           |
                         |                          |           |
rx_Query(?)              |Tx_Response_Dmode         |Established|2)
&&(!ConfirmRequired)     |Install MRS               |Upstream   |
                         |IF(NSLPdata)              |MRS        |
                         |Tg_RecvMsg(NSLPdata)      |           |
                         |                 to Appl.|           |
                         |                          |           |
rx_Query(MAinfo)         |Tx_Response_?mode(Resp    |Wait       |*
&&(ConfirmRequired)      |          Cookie, MAinfo)|Confirm    |1)
                         |                          |           |
rx_Query(Mainfo)         |Tx_Response_?mode(Resp    |Established|*
&&(!ConfirmRequired)     |          Cookie, MAinfo)|Upstream   |2)
                         |Set WaitConfirm timer     |MRS        |
                         |Install MRS               |           |
                         |                          |           |
-------------------------+--------------------------+-----------+---



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

Condition                Action                     State      Note
-------------------------+--------------------------+-----------+---
rx_Msg_?mode             |Tx_Response(No_MRS_       |Wait       |3)
                         |               installed)|Confirm    |
```

Rx_Confirm_?mode	Install Upstream MRS	Established Upstream MRS	* 6)
Timeout_WaitConfirm		IDLE	

State: Established Upstream MRS

Condition	Action	State	Note
Rx_Confirm_?mode	Stop WaitConfirm timer Install MRS Tx_queued_Msg_Cmode	Established Upstream MRS	6)
(Timeout_WaitConfirm)&& (!UpstreamPeerInstalled)	(Delete MRS) IF (MA is used) ((Delete MA)|| (Stop using shared MA)) Tg_NetworkNotification	IDLE	7) **

rx_Msg_?mode	IF(CheckPeerInfo)
	Tg_RecvMsg to Appl.
	ELSE
	Tx_Response(No_MRS_
	installed)
		Established Upstream MRS	8)

tg_SendMsg	IF(WaitConfirm timer set) Queue NSLP msg data ELSE Tx_Msg_?mode	Established Upstream MRS	9)
(Timeout_MRSlifetime)|| (tg_SetStateLifetime(0))	(Delete MRS)&& IF (MA is used) ((Delete MA)|| (Stop using shared MA)) Tg_NetworkNotification	IDLE	
rx_Query(Refresh)	Refresh MRS Tx_Response(Refresh)	Established Upstream MRS	
(rx_Query(MAinfo))||	Tx_Response_?mode(Resp	Established	*

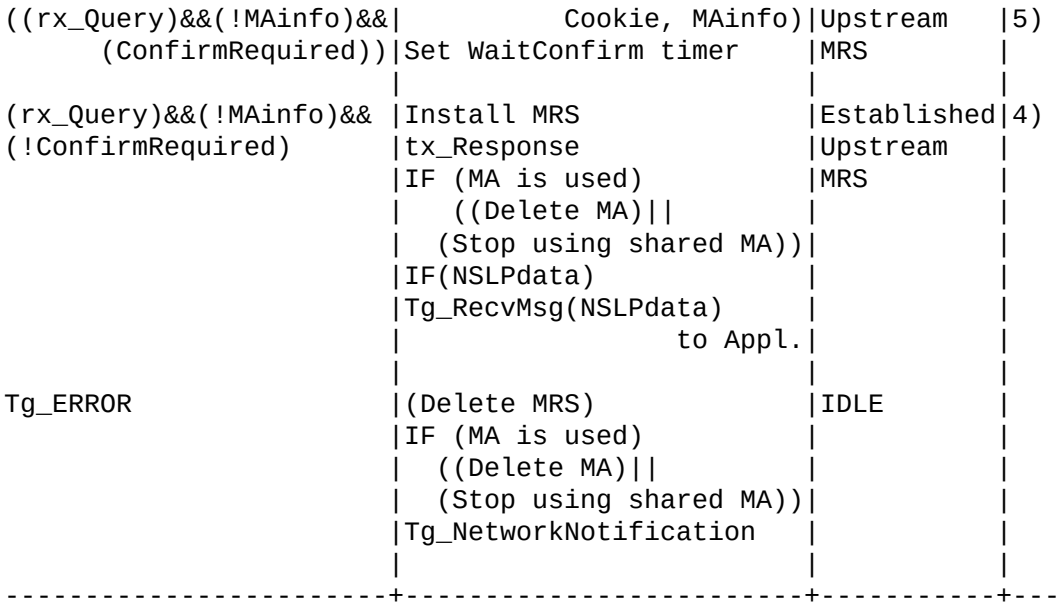

((rx_Query)&&(!MAinfo)&&
(ConfirmRequired))
Cookie, MAinfo)
Set WaitConfirm timer
Upstream MRS
5)
(rx_Query)&&(!MAinfo)&&
(!ConfirmRequired)
Install MRS
tx_Response
IF (MA is used)
((Delete MA)||
(Stop using shared MA))
IF(NSLPdata)
Tg_RecvMsg(NSLPdata)
to Appl.
Established
Upstream MRS
4)
Tg_ERROR
(Delete MRS)
IF (MA is used)
((Delete MA)||
(Stop using shared MA))
Tg_NetworkNotification
IDLE

Figure 5

Authors' Addresses

Tseno Tsenov
Sofia,
Bulgaria

Email: tseno.tsenov@mytum.de

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

Email: Hannes.Tschofenig@nsn.com

Xiaoming Fu
University of Goettingen
Computer Networks Group
Lotzestr. 16-18
Goettingen 37083
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

Full Copyright Statement

Intellectual Property

Acknowledgment

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