NSI Process Coordination and Message Transport

# Message Transport

With the flexibility of the NSI CSv2.0 to instantiate tree workflows of arbitrary complexity, it was essential to formalize the concept of a Message Transport Layer (MTL) that could to provide the necessary message delivery abstractions to the NSI layer to simplify its operation.

The MTL is responsible for end-to-end communications between NSAs and has two primary requirements:

* Sending and receiving of messages; The MTL is responsible for encapsulating the datagram with all the necessary information (e.g. source/destination, port, protocol, etc) for delivery, and removing transport information when a datagram is received prior to passing it to the Coordinator.
* Verifying if a message was received by the intended destination NSA; To do this, the MTL utilizes message receipt acknowledgement and timeouts to determine if a packet was or was not successfully delivered.

It should be noted that there is no inherent requirement for the MTL to be reliable or ensure delivery order because these functions can be accomplished by the higher level processes. ***For NSI CSv2.0, SOAP (ref ?) is used as the MTL.***

Figure 1. Coordinator and MTL in an NSA

# Message and Process Coordination

As the MTL only defines basic message transport capabilities, the NSA requires more intelligent message and process coordination to function. These capabilities are defined in a logical entity called the coordinator. Even though both the MTL and Coordinator are part of the NSA, the Coordinator is integral to the NSI Stack, where as the MTL is functionally distinct and can be readily substituted.

Figure 2. Example workflow tree of a NSI reservation request

Logically, the Coordinator has several responsibilities, mainly:

* Coordinating, tracking, and aggregating (if necessary) of message requests, replies, and notifications
* Processing or forwarding of notifications as necessary
* Servicing query requests

## Communications

Reliable communications is a vital requirement of the NSA. With the MTL providing only basic message transport capabilities, it is the responsibility of the Coordinator to keep track of message states and make decisions accordingly. To do this, the Coordinator must maintain the following information on a per NSI request message basis:

* Who was the (NSI request) message sent to
* Was the message received (i.e. ack’ed) or not (i.e. MTL timeout)
* Which NSA has sent back an NSI reply (e.g. \*.confirm, \*.fail, \*.not\_applicable) for the initial NSI request (e.g. \*.request)

### Per Request Information Elements

For each NSI request/reply interaction, the Coordinator must maintain several items of information that are associated with those messages. This is particularly important for an Aggregator NSAs (AG) since it must keep track of the message status for each of its children in the request workflow. This information includes:

* NSA IDs: A list of NSA that the messages were sent to.
* Connection ID: The name that uniquely identifies the connection request/reservation (see “ogf\_nsi\_connection\_types\_v2\_0.xsd” for more detail).
* Correlation ID: The label that identifies messages associated to a unique NSI request/reply interaction. This is used to associate NSI replies to requests, and also to identify messages for re-delivery (i.e. message retries).
* Message status: This provides the message state for each of the NSI requests sent to the various NSAs to reflect the current status, such as; MTL sent, MTL receipt acknowledged, MTL timeout, and Coordinator timeout.

In addition to the detailed information of the status for each child NSA NSI request (see “*request\_segment\_list(Conn\_ID, NSA)*” in Figure 4.), the Coordinator must also maintain an aggregate message status indicating if the messages were delivered successfully to all the children (see “*request\_list(Conn\_ID)*” Figure 4.).

### Timeouts

To identify communication failures, both the MTL and Coordinator have defined timeouts to detect breakdowns in certain aspects of the communication channel. The characteristics of these timeouts are outlined below:

* MTL Timeout
  + Symptoms
    - No acknowledgement of message receipt after a pre-determined time period after the message was sent.
  + Causes
    - Failure in end-to-end communication between NSAs.
* Coordinator Timeout
  + Symptoms
    - No NSI reply after a pre-determined time period after the NSI request was sent.
  + Causes
    - Failure in the MTL such that the NSI reply (from the PA) could not be delivered to the requesting NSA (the RA).
    - The NSA processing the request (e.g. PA) was unable to reply due to incapacitation.
    - The NSA processing the request (Aggr) was blocked waiting for NSI replies from downstream NSAs. *(This scenario can be resolved by adjusting the Coordinator timeout value of the requester.)*

As both the MTL and Coordinator timeouts are distinct and can be set exclusively, it is important to understand the interplay between the MTL and Coordinator timeouts in order to mitigate artificial “failures”.

Figure 3. Potential MH/MTL timeout sequences

In the event of an MTL or Coordinator timeout, the Coordinator will generate a message delivery failure notification and send it up the workflow tree (towards the uRA).

### Failure Recover

In NSI CSv2.0, there is no inherent expectation for any (interim) NSAs except for the ultimate requester NSA (uRA) to make a decision and take action when it receives a message delivery failure notification. Any interim (aggregator) NSA that receives the delivery failure notification MUST forward it up the workflow tree if it does not want to or cannot resolve the issue. On receiving the message delivery failure notification, the uRA ultimate has two choices:

1. Terminate the reservation; this is done by sending down a terminate request (“term.rq”) through the workflow tree.
2. Request redelivery of the original message; this is done by resending down the original message through the workflow tree.

When the original message is resent down the workflow tree, it will contain the original Correlation ID. AGs receiving the duplicate request SHOULD only attempt redelivery of the message to children that it did not receive an acknowledgement for (i.e. MTL timeout) or reply to (i.e. Coordinator timeout) the original message. If the message sent with the original Correlation ID does not match the original message (e.g. different message parameters/content), the message is rejected and an error returned.

## Information

While per request information (see “Per Request Information Elements”) will only persist for the duration of the NSI request/reply interaction, the Coordinator must also store information associated with the entire reservation.

Figure 4. Information maintained by Coordinator for each Connection Reservation and NSI Request

### Per Reservation Information Elements

To support the recursive query function in NSI CS v2.0, a (AG)Coordinator must track the current states (i.e. RSM, PSM, LSM) of all its children as well as the condition of the data plane status. This information is persistent but updated over the lifetime of the reservation (see “*connection\_segment\_list(Conn\_ID, NSA)*” in Figure 4.).

* NSA IDs: A list of NSA that are part of the connection request workflow tree.
* Connection IDs: Connection IDs associated with each NSA in the workflow tree.
* Source and Destination STPs: The source and destination STP of each NSA segment composing the end-to-end circuit
* Reservation Parameters: A list of reservation parameters (e.g. start/end time, bandwidth, etc) associated with each NSA segment
* RSM States: State of children’s Reservation State Machine and current committed reservation version number
* PSM States: State of children’s Provision State Machine
* LSM States: State of Children’s Lifecycle State Machine
* Data plane states: The status of the children’s data plane (i.e. in-service/out-of-service), and the version of the reservation instantiated in the data plane if it is in-service (see “Data Plane Status Information” section for more details).

### Reservation Versioning Information

To support the modification of reservations, the notion of versioning has been introduced to identify the instance of a reservation over its lifetime. The properties and characteristics of the versioning is as follows:

* Version numbers are integer values ≥ 0 (zero)
* Version numbers are assigned by the RA when a reservation request (i.e. NSI\_rsv.rq) is made to a PA
* An integer ≥ 0 can be assigned by the RA for the initial request, however subsequent modifications to the request MUST use monotonically increasing version numbers (although they need not be sequential)
* If a version number is not specified in an NSI\_rsv.rq, it is assumed to be 0 (zero) regardless if it is the initial or subsequent requests
* An NSI\_rsv.rq with a version number ≤ the (highest) current committed reservation version number will result in a failed request and an appropriate error
* A uPA must keep track of
  + Version number of currently committed reservation
  + Version number of pending modification request (if any)
  + Version number of reservation instantiated in the data plane by the NRM
* An Aggregator must keep track of
  + Version numbers of currently committed reservations in each child segment
  + Version number of pending modification request (only one modify can be outstanding at any time)
  + Version numbers of reservations instantiated in the data plane in each child segment (see “Data Plane Status Information”)
* Version numbers of failed (e.g. timed-out) or aborted modifications are not stored, and therefore can be reused. For example:

1. Successful initial NSI\_req.rq(ver = 2) results in Reservation(v2)
2. Successful modify NSI\_req.rq(ver = 5) results in Reservation(v5)
3. Failed modify NSI\_req.rq(ver = 6) retains Reservation(v5)
4. Subsequent successful modify NSI\_req.rq(ver = 6) results in Reservation(v6)

### Data Plane Status Information

To reflect the state of the data plane, a Coordinator will maintain three flags:

* Activate (boolean): To indicate whether the data plane is in-service or out-of-service
  + uPA:
    - True => data plane is in-service
    - False => data plane is out-of-service
  + AG:
    - True => all children’s data planes are in-service
    - False => one or more children’s data plane is out-of-service
* Version (int): The version of the committed reservation instantiated in the data plane. *NB: This field is only valid when ActivateFlag is true.*
  + uPA: Version number of the committed reservation
  + AG: Largest version number of the committed reservation among the children
* VersionConsistent (boolean): Reflects if the “Version” numbers are consistent
  + uPA: This is always True
  + AG:
    - True => all children’s “Version” numbers are the same
    - False => all children’s “Version” numbers are not the same

When there is a change in the data plane status (i.e. uPA is notified by its NRM, or AG notified by one or more of its children), the Coordinator will send up the workflow tree a “DataPlaneStateChange.nt” notification with the updated Activate, Version, and VersionConsistent values.

For the AG, reporting the aggregate data plane state of its children requires some processing. The following pseudo-code describes this behavior:

if all of ChildrenDataPlaneStatus[1..n].Active are true then

{

DataPlaneStatus.Active = true

DataPlaneStatus.Version =

maximum\_of(ChildrenDataPlaneStatus[1..n].Version)

If all ChildrenDataPlaneStatus[1..n].Version are the same, and

all of ChildrenDataPlaneStatus[1..n].VersionCosistent are true then

{

DataPlaneStatus.VersionConsistent = true

}

else

{

DataPlaneStatus.VersionConsistent = false

}

}

## Process Coordination

The following is an attempt to describe the behavior of the Coordinator in relation to the processing of requests and interactions with the various state machines in the NSA. Due to the slight difference in behavior between an AG and uPA, they are describe separately

### Aggregator NSA

#### Processing of NSI Requests

**NSI\_rsv.rq(Conn\_ID, Corr\_ID, Ver)** /\* ***from parent NSA*** \*/

if (new Conn\_ID) then

{

create state machines RSM(Conn\_ID), PSM(Conn\_ID), LSM(Conn\_ID)

do path finding -> create entry for all children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

}

send res.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_rsvcommit.rq(Conn\_ID, Corr\_ID, ver)** /\* ***from parent NSA*** \*/

send rsvcommit.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_rsvabort.rq(Conn\_ID, Corr\_ID, ver)** /\* ***from parent NSA*** \*/

send rsvabort.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_prov.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send prov.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_rel.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send rel.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_term.rq** /\* ***from parent NSA*** \*/

send term.rq(Corr\_ID) to LSM(Conn\_ID)

send term.rq to RSM(Conn\_ID), PSM(Conn\_ID) /\* if RSM and PSM exist \*/

**NSI\_rsv.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send res.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsv.fl(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

if request\_list(Conn\_ID, Corr\_ID).Status != fail then

{

set request\_list(Conn\_ID, Corr\_ID).Status = fail

send res.fl(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvcommit.cf(Conn\_ID, Corr\_ID, Ver)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rsvcommit.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvcommit.fl(Conn\_ID, Corr\_ID, Ver)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rsvcommit.fl(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvabort.cf(Conn\_ID, Corr\_ID, Ver)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rsvabort.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_prov.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send prov.cf(Corr\_ID) to PSM(Conn\_ID)

}

**NSI\_rel.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rel.cf(Corr\_ID) to PSM(Conn\_ID)

}

**NSI\_term.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send term.cf(Corr\_ID) to LSM(Conn\_ID)

}

#### Requests from State Machines

**rsv.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_rsv.rq(Conn\_ID, Corr\_ID, Ver) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rsvcommit.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_rsvcommit.rq(Conn\_ID, Corr\_ID, Ver) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rsvabort.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_rsvabort.rq(Conn\_ID, Corr\_ID, Ver) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rsv.cf(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsv.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsv.fl(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsv.fl(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvcommit.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsvcommit.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvcommit.fl(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsvcommit.fl(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvabort.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsvabort.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**prov.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_prov.rq(Conn\_ID, Corr\_ID) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rel.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_prov.rq(Conn\_ID, Corr\_ID) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**prov.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_prov.cf(Conn\_ID, Corr\_ID) to the parent

**rel.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_rel.cf(Conn\_ID, Corr\_ID) to the parent

**term.rq(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_term.rq(Conn\_ID, Corr\_ID) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**term.cf(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

clean up everything related to Conn\_ID

send NSI\_term.cf(Conn\_ID, Corr\_ID) to the parent

### Ultimate Provider NSA

#### Processing of NSI Requests

**NSI\_rsv.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

if (new Conn\_ID) then

{

create state machines RSM(Conn\_ID), PSM(Conn\_ID), LSM(Conn\_ID)

}

send res.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

if reservation is made by checking the Reservation DB then

{

send res.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

else

{

send res.fl(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvcommit.rq(Conn\_ID, Corr\_ID, Ver)** /\* ***from parent NSA*** \*/

send rsvcommit.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_rsvabort.rq(Conn\_ID, Corr\_ID, Ver)** /\* ***from parent NSA*** \*/

send rsvabort.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_prov.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send prov.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_rel.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send rel.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_term.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send term.rq(Corr\_ID) to LSM(Conn\_ID)

send term.rq to RSM(Conn\_ID), PSM(Conn\_ID), ASM(Conn\_ID)

/\* if RSM, PSM and ASM exist \*/

#### Requests from State Machines

**rsv.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

ignore

**rsvcommit.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

ignore

**rsvabort.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

ignore

**rsv.cf(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

set REPLIED(Corr\_ID)

send NSI\_rsv.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsv.fl(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

set REPLIED(Corr\_ID)

send NSI\_rsv.fl(Conn\_ID, Corr\_ID) to the parent

**rsvcommit.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

commit the reservation(Conn\_ID, Ver)

set REPLIED(Corr\_ID)

send NSI\_rsvcommit.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvcommit.fl(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

commit the reservation(Conn\_ID, Ver)

set REPLIED(Corr\_ID)

send NSI\_rsvcommit.fl(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvabort.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

abort the reservation(Conn\_ID, Ver)

set REPLIED(Corr\_ID)

send NSI\_rsvabort.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**prov.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

set prov\_flag(Conn\_ID)

if in\_period\_flag is set then

{

activate data plane according to the latest reservation

send prov.cf(Corr\_ID) to PSM(Conn\_ID)

}

**rel.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

reset prov\_flag(Conn\_ID)

deactivate data plane

send rel.cf(Corr\_ID) to PSM(Conn\_ID)

**prov.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_prov.cf(Conn\_ID, Corr\_ID) to the parent

**rel.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_rel.cf(Conn\_ID, Corr\_ID) to the parent

**term.rq(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

ignore

**term.cf(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

clean up everything related to Conn\_ID

send NSI\_term.cf(Conn\_ID, Corr\_ID) to the parent