**NSI Connection Service Protocol v1.1**

Status of This Document

This document provides information to the Grid community on the NSI Connection Service that operates on the interface between a requesting software agent and the provider software agent. It describes the protocol, architecture and associated processes and environment in which software agents interact to deliver the Connection Service. Distribution is unlimited.

Copyright Notice

Copyright © Open Grid Forum (2008-2012). All Rights Reserved.

Trademark

OGSA is a registered trademark and service mark of the Open Grid Forum.

Abstract

This document describes the Connection Service Protocol for the Network Service Interface (NSI). The Connection Service is used to manage connection oriented circuits that transit network providers. The Network Service Interface (NSI) is defined to be the set of protocols and parameters that are used between a software agent requesting a network service and the software agent providing that Network Service. The Connection Service is intended to operate within the Network Service Framework (NSF, GFD.173).

Contents

[1. Overview 2](#_Toc312162314)

[2. Connection Service architecture 3](#_Toc312162315)

[3. Connection Service lifecycle 4](#_Toc312162316)

[3.1 Connection Service primitives 4](#_Toc312162317)

[3.2 Connection Service state-machine 6](#_Toc312162318)

[3.3 Connection Service message handling 7](#_Toc312162319)

[3.4 Requesting a Reservation 11](#_Toc312162320)

[3.5 Provisioning a Connection 11](#_Toc312162321)

[3.6 Releasing the Provisioning state 12](#_Toc312162322)

[3.7 Terminating a Connection 12](#_Toc312162323)

[3.8 Forced end 12](#_Toc312162324)

[3.9 Querying a Connection 12](#_Toc312162325)

[4. Connection Service primitives and attributes 12](#_Toc312162326)

[4.1 Connection Service primitives 12](#_Toc312162327)

[4.2 Common types and attributes 13](#_Toc312162328)

[4.2.1 Generic types 13](#_Toc312162329)

[4.2.2 Common transport envelope and attributes 14](#_Toc312162330)

[4.2.3 Common message attributes 14](#_Toc312162331)

[4.3 Message types 15](#_Toc312162332)

[4.3.1 ReserveRequest 15](#_Toc312162333)

[4.3.2 ReserveConfirmed 17](#_Toc312162334)

[4.3.3 ReserveFailed 18](#_Toc312162335)

[4.4 Terminating, Provisioning and Releasing messages and attributes 18](#_Toc312162336)

[4.5 Query messages and attributes 20](#_Toc312162337)

[4.5.1 QueryRequest 20](#_Toc312162338)

[4.5.2 QueryConfirm 21](#_Toc312162339)

[4.5.3 QueryFailed 22](#_Toc312162340)

[5. Connections: Transport and Service planes 22](#_Toc312162341)

[6. Inter-domain vs intra-domain pathfinding 23](#_Toc312162342)

[7. STPs and Paths 24](#_Toc312162343)

[7.1 STP syntax 24](#_Toc312162344)

[7.2 Under-specified STPs 24](#_Toc312162345)

[7.3 Path 24](#_Toc312162346)

[7.4 Path example in an NSA chain 24](#_Toc312162347)

[7.5 Path example in an NSA tree 25](#_Toc312162348)

[8. Authentication and Authorization 26](#_Toc312162349)

[8.1 Security Requirements 26](#_Toc312162350)

[8.2 Message Security 26](#_Toc312162351)

[8.3 Authorization 27](#_Toc312162352)

[9. Failures and exceptions 27](#_Toc312162353)

[9.1 Service plane failures 27](#_Toc312162354)

[9.2 Transport plane failures 27](#_Toc312162355)

[10. Appendix A: Best Practices for NSA implementation 28](#_Toc312162356)

[10.1 Message transport error handling 28](#_Toc312162357)

[10.2 ACK handling 29](#_Toc312162358)

[10.3 Guidelines on timeouts: 29](#_Toc312162359)

[3. The requester NSA may choose to send queries to check the status of a request rather than terminating at timeout. 29](#_Toc312162360)

[10.4 Parallel processing of messages: 29](#_Toc312162361)

[The provider NSA should respond to queries even if still working on a response to a request. 29](#_Toc312162362)

[10.5 NTP servers 29](#_Toc312162363)

[11. Appendix B: Service Definitions for Connection Services 30](#_Toc312162364)

[12. Contributors 33](#_Toc312162365)

[13. Glossary 33](#_Toc312162366)

[14. Intellectual Property Statement 35](#_Toc312162367)

[15. Disclaimer 35](#_Toc312162368)

[16. Full Copyright Notice 35](#_Toc312162369)

[17. References 36](#_Toc312162370)

# Overview

1.1 Summary

The NSI protocol is defined by a suite of documents. This recommendation describes the NSI Connection Service Protocol and should be read in conjunction with the NSI Network Services Framework (NSF, GFD.173).

The Network Service Framework (NSF) defines several key architectural elements: a Network, a Network Service, a Network Service Agent (NSA), a Network Service Interface (NSI), and a NSI Protocol. These elements exist in a notional Network Service Plane. The framework describes an environment within which network resources are treated as explicitly manageable objects. Within the framework, these network resources can be selected, allocated, interrogated, and manipulated by software agents on behalf of requesting users.

Network resources and capabilities are presented to the consumer through a set of ‘Network Services’. The Network Services Framework presents a unified model for interacting with these services. Network Services include the ability to create connections, to share topology information, and to do other services needed by a set of federated NSAs.

This document defines a Connection Service protocol to support the reservation, creation, management and removal of Connections.

Where capitalized words are used in this document, these have a formal definition; see the glossary for details.

1.1 Context

Multi-protocol environments

Traditional models of circuit services and control planes adopt a single very tightly defined Data Plane technology, and then hard code the service attributes (signaling parameters) into the control plane protocols. Emerging multiprotocol services will need to leverage many Data Plane technologies and will need to recognize a wider array of service attributes. The NSI supports an abstracted notion of a Connection, and the NSI Service Definition provides a mechanism for specifying service specific constraints or limits on that connection as realized in different Networks. It is up to the NSA path finders and/or the NRM pathfinders to select a path that meets the Service Request and is consistent with the Service Definitions along that path. NSI allows a single Service Plane protocol suite to present different services and different transport capabilities to the user.

Multi-Provider environments

Traditional models of circuit services and control planes assume operation within a single operator environment. NSI is designed to support the creation of Connections that transit several operators. This creates specific requirements for authorization and authentication which are addressed by NSI.

# Connection Service architecture

The Connection Service is one of the services supported by the Network Service Framework (NSF). The Connection Service communicates using messaging between Requester Agent (RA) and Provider Agent (PA) state-machine pairs; this RA/PA relationship conforms to the usage defined in the NSF. Figure 1 shows an example of the use of Connection Service and RA/PA pairs.

The NSI messages exchanged between RA (blue) and PA (red) and the associated state-machine make up the NSI CS protocol definition. The NSA functionality and associated internal NSA messages, while mentioned in this document, do not form a normative part of the NSI CS protocol.

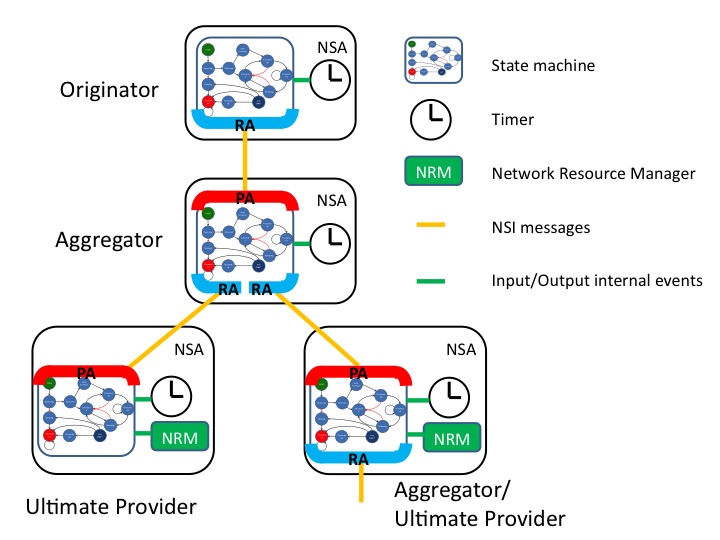


Figure : Example of NSA roles

Any NSA can act as an aggregator and initiate connection service requests to multiple PA’s. The relationship between the RA and PA can be represented as a parent-child relationship where the initiating RA is the parent and the PA is the child. A NSA aggregator implementation should include a PA and multiple RA logical functions. The Network Resource Manager (NRM) is the function that performs the local control of the network resources and is also considered to be a child of the aggregator or ultimate provider.

The Connection Service’s behavior is dependent on the presence of an NRM and children NSAs. In general, 4 combinations of RA, PA and NRM are identified:

* An NSA with the originating RA in the hierarchy is referred to as a CS originator.
* An NSA with a PA and one or more RAs is referred to as a CS aggregator.
* An NSA with a PA and an NRM only is referred to as a CS ultimate provider.
* An NSA with a PA and an NRM and one or more RAs is referred to as a CS aggregator and ultimate provider.

# Connection Service lifecycle

## Connection Service primitives

The Connection Service (CS) protocol is a message based command-response protocol that operates between an RA and a PA.

The NSI CS protocol defines a set of six primitives that provide the control necessary to manage Connections; these are described in Table 1.

|  |  |
| --- | --- |
| **Primitive** | **Description** |
| ***reserve*** *(Request/Confirm/Failed)* | The RA requests the PA to Reserve network resources for a Connection between two STP’s constrained by certain service parameters. |
| ***provision*** *(Request/Confirm/Failed)* | The RA requests the PA to Provision a Reservation (associated with a previous reservation message). |
| ***release*** *(Request/Confirm/Failed)* | The RA request for the PA to de-provision resources without removing the Reservation. |
| ***terminate*** *(Request/Confirm/Failed)* | The RA request for the PA to release the Provisioned resources and terminate the Reservation. |
| ***forcedEnd***  *(Request only)* | This is reported by the PA to the RA to notify that the PA has forced a termination of the Reservation. |
| ***query***  *(Request only)* | Mechanism for either RA or PA to query the other NSA for a set of connection service instances between the RA-PA pair. This message can be used as a status polling mechanism. |

Table : List of primitives

The 3 possible extensions for the message primitives are:

* ***Request*** – The RA sends the request to the PA, for example *reserveRequest*.
* ***Confirm*** - A PA sends this positive operation response message (such as *reserveConfirm*) to the Requester NSA that issued the original request message (*reserveRequest*).
* ***Failed*** - A Provider NSA sends this negative operation response message (such as *reserveFailed*) to the Requester NSA that issued the original request message (*reserveRequest*).

The following figure shows to examples of how message primitives are used to first Reserve and then Provision a Connection. Two modes of Provisioning are supported: Manual and Automatic.

Z:\OGF_NSI\NSI _protocol\CS lifecycle.emf

Figure : Connection Service lifecycle

For brevity of this diagram and the state-machine, the NSI messages are abbreviated as follows:

rsv reserve

prov provision

rel release

term terminate

fcd\_end forcedEnd

rq request

cf confirm

fl failed

These Connection Service message primitives are used to initiate, manage and remove a Connection (the connection life cycle). A single state-machine for both RA and PA describes the state changes and their relationship to messages. The primitives, timing events and NRM responses form inputs and outputs to the state-machine. The connection state can be any of the following:

|  |  |
| --- | --- |
| **State** | **Description** |
| Initial (pseudo state) | the state-machine is not created until a *reserveRequest* is received |
| Reserving | A *reserveRequest* has either been sent or received and the processing is in progress. |
| Reserved | The *reserveRequest* has succeeded and a connection schedule has been created. |
| AutoProvision | A *provisionRequest* has been sent or received before the requested connection *startTime*. The NSA will now automatically provision the Connection at *startTime*. |
| Scheduled | The Reservation *startTime* has been reached and the Connection is ready to be provisioned as soon as the *provisionRequest* message is received. |
| Provisioning | The *provisionRequest* has been sent or received and the Provisioning is in progress. |
| Provisioned | The Connection has been successfully provisioned |
| Releasing | A *releaseRequest* has been sent or received and a release of the connection is in progress |
| Cleaning | A *terminateRequest* has been initiated by this state-machine and awaiting confirmation from the children. |
| Terminating | a *terminateRequest* message has been received from the parent and waiting for connection termination by all children |
| Terminated | Nothing exists any longer, a terminate request has been successful. The current state-machine still receives messages from the children and passes it up to the parent |

Table : List of states

## Connection Service state-machine

The Connection Service state-machine is shown in Figure 3. This is a generic state-machine which incorporates all of the possible operational modes: originator, aggregation and ultimate provider.

Z:\OGF_NSI\NSI _protocol\NSI-SM-Single-Diagram-July 22 2011.emf

Figure : NSI connection service state-machine

The text boxes show the messages associated with transitions between states. These are color coded as follows:

*black*: an internal event – either a timer or result of an operation within the NSI protocol.

*red* : an incoming NSI message – this may be from either a parent or child NSA

*blue*: an output event which is a NSI message – this may go to either a parent or a child NSA

NSI messages have an associated direction; upstream messages are forwarded the by a PA to an RA, downstream messages are sent from a RA to a PA. These directions are indicated with the use of the following symbol prefixes:

**>**: Downstream input/output

**<**: Upstream input/output

Z:\OGF_NSI\NSI _protocol\SM_key.emf

The following internal messages are possible. Note that these are a non-normative part of the NSI specification.

SM 🡪 NRM

***reserve***: reserve the requested resources on the local NRM.

***provision***: Provisioning of resources within local NRM

***release:*** release of resources within the local NRM

***terminate:*** terminate a Reservation within local NRM

NRM 🡪 SM (reply)

***\*\_cf***: the requested action in the local NRM was completed successfully.

***\*\_fl:*** the requested action failed in local NRM.

NRM 🡪 SM (asynchronous event)

***forced\_end:*** resources allocated to the schedule were (administratively) released by the local NRM

Timer 🡪 SM

***start\_time:*** start time of the Reservation has been reached

***end\_time:*** end time of the Reservation has been reached

## Connection Service message handling

Each message is uniquely handled depending on the current state of the Connection Service state-machine. Table 3 shows a full matrix of all messages and all states, and the actions to be taken. For example if the current state is the initial state and a legitimate *reserveRequest* is received, then the next state is Reserving and the output event is to perform the Reservation. If necessary, a *reserveRequest* is forwarded to any children present.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| State | Incoming message | | | | |
|  |  |  |  |  | start\_time |
|  | >rsv.rq | >prov.rq | >rel.rq | >term.rq |  |
| Initial | Reserving | N/A\*1 | N/A\*1 | N/A\*1 | N/A\*1 |
| reservation |
| >rsv.rq |
| Reserving | Reserving | Reserving | Reserving | Terminating | Hold\*2 |
|  |  |  | release terminate |
| <rsv.fl | <prov.fl | <rel.fl | >term.rq |
| Reserved | Reserved | Auto Provision | Reserved | Terminating | Scheduled |
|  |  |  | release terminate |  |
| <rsv.fl | >prov.rq | <rel.fl | >term.rq |  |
| Auto Provision | Auto Provision | Auto Provision | Auto Provision | Terminating | Provisioning |
|  |  |  | release terminate | provision |
| <rsv.fl | <prov.fl | <rel.fl | >term.rq |  |
| Scheduled | Scheduled | Provisioning | Releasing | Terminating | Scheduled |
|  | provision | release | release terminate |  |
| <rsv.fl | >prov.rq | >rel.rq | >term.rq |  |
| Provisioning | Provisioning | Provisioning | Provisioning | Terminating | Provisioning |
|  |  |  | release terminate |  |
| <rsv.fl | <prov.fl | <rel.fl | >term.rq |  |
| Provisioned | Provisioned | Provisioned | Releasing | Terminating | Provisioned |
|  |  | release | release terminate |  |
| <rsv.fl | <prov.cf | >rel.rq | >term.rq |  |
| Releasing | Releasing | Releasing | Releasing | Terminating | Releasing |
|  |  |  | release terminate |  |
| <rsv.fl | <prov.fl | <rel.fl | >term.rq |  |
| Cleaning | Cleaning | Cleaning | Cleaning | Cleaning | Cleaning |
|  |  |  |  |  |
| <rsv.fl | <prov.fl | <rel.fl | <term.fl |  |
| Terminating | Terminating | Terminating | Terminating | Terminating | Terminating |
|  |  |  |  |  |
| <rsv.fl | <prov.fl | <rel.fl | <term.fl |  |
| Terminated | Terminated | Terminated | Terminated | Terminated | Terminated |
|  |  |  |  |  |
| <rsv.fl | <prov.fl | <rel.fl | <term.fl |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| State | Incoming message | | | | |
|  | end\_time | reserve\_cf | reserve\_fl | provision\_cf | provision\_fl |
|  |  | <rsv.cf | <rsv.fl | <prov.cf | <prov.fl |
| Initial | N/A\*1 | N/A\*1 | N/A\*1 | N/A\*1 | N/A\*1 |
|
|
| Reserving | Terminated | Reserved | Cleaning | Reserving | Reserving |
| release terminate |  | terminate |  |  |
|  | <rsv.cf | <rsv.fl >term.rq |  |  |
| Reserved | Terminated | Reserved | Reserved | Reserved | Reserved |
| release terminate |  |  |  |  |
|  |  |  |  |  |
| Auto Provision | Terminated | Auto Provision | Auto Provision | Auto Provision | Auto Provision |
| release terminate |  |  |  |  |
|  |  |  |  |  |
| Scheduled | Terminated | Scheduled | Scheduled | Scheduled | Scheduled |
| release terminate |  |  |  |  |
|  |  |  |  |  |
| Provisioning | Terminated | Provisioning | Provisioning | Provisioned | Scheduled |
| release terminate |  |  |  |  |
|  |  |  | <prov.cf | <prov.fl |
| Provisioned | Terminated | Provisioned | Provisioned | Provisioned | Provisioned |
| release terminate |  |  |  |  |
|  |  |  |  |  |
| Releasing | Terminated | Releasing | Releasing | Releasing | Releasing |
| release terminate |  |  |  |  |
|  |  |  |  |  |
| Cleaning | Cleaning | Hold\*2 | Hold\*2 | Hold\*2 | Hold\*2 |
|  |
|  |
| Terminating | Terminating | Hold\*2 | Hold\*2 | Hold\*2 | Hold\*2 |
|  |
|  |
| Terminated | Terminated | Terminated | Terminated | Terminated | Terminated |
|  |  |  |  |  |
|  | <rsv.cf | <rsv.fl | <prov.cf | <prov.fl |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| State | Incoming message | | | | |
|  | release\_cf | release\_fl | terminate\_cf | terminate\_fl | forced\_end |
|  | <rel.cf | <rel.fl | <term.cf | <term.fl | <fcd\_end |
| Initial | N/A\*1 | N/A\*1 | N/A\*1 | N/A\*1 | N/A\*1 |
|
|
| Reserving | Reserving | Reserving | Reserving | Reserving | Cleaning |
|  |  |  |  | release terminate |
|  |  |  |  | <fcd\_end >term.rq |
| Reserved | Reserved | Reserved | Reserved | Reserved | Cleaning |
|  |  |  |  | release terminate |
|  |  |  |  | <fcd\_end >term.rq |
| Auto Provision | Auto Provision | Auto Provision | Auto Provision | Auto Provision | Cleaning |
|  |  |  |  | release terminate |
|  |  |  |  | <fcd\_end >term.rq |
| Scheduled | Scheduled | Scheduled | Scheduled | Scheduled | Cleaning |
|  |  |  |  | release terminate |
|  |  |  |  | <fcd\_end >term.rq |
| Provisioning | Provisioning | Provisioning | Provisioning | Provisioning | Cleaning |
|  |  |  |  | release terminate |
|  |  |  |  | <fcd\_end >term.rq |
| Provisioned | Provisioned | Provisioned | Provisioned | Provisioned | Cleaning |
|  |  |  |  | release terminate |
|  |  |  |  | <fcd\_end >term.rq |
| Releasing | Scheduled | Scheduled | Releasing | Releasing | Cleaning |
|  |  |  |  | release terminate |
| <rel.cf | <rel.fl |  |  | <fcd\_end >term.rq |
| Cleaning | Hold\*2 | Hold\*2 | Terminated | Terminated | Cleaning |
|  |  |  |
|  |  |  |
| Terminating | Hold\*2 | Hold\*2 | Terminated | Terminated | Terminating |
|  |  |  |
| <term.cf | <term.fl |  |
| Terminated | Terminated | Terminated | Terminated | Terminated | Terminated |
|  |  |  |  |  |
| <rel.cf | <rel.fl | <term.cf |  |  |

Table : message handling

Key to Table 3: message handling:

|  |  |
| --- | --- |
| black strings: | states |
| red strings: | NSI messages |
| blue strings: | internal events |

|  |  |
| --- | --- |
|  | Input event |
|  | Input message |
| Current State | Next State |
| Output event |
| Output message |

|  |
| --- |
| Unexpected input/event |
| (recommended to log |
| and notify to the admin of the NSA) |

|  |  |
| --- | --- |
| reserve\_cf | When messages and event are received from all children |
| <rsv.cf |
|  |  |
| reserve\_fl | When messages/event (either cf or fl) are received from all children, and one or more of them is fl |
| <rsv.fl |
|  |  |
| forced\_end | When one or more message or event is received |
| <fcd\_end |

\*1 "Initial" is a pseudo state. An SM is generated when >rsv.rq is received. Therefore, these cases never occur.

\*2 The input event/message should be held (not consumed) and will be processed in a future state. No transition occurs.

## Requesting a Reservation

A *reserveRequest* is sent by the RA to the PA to initiate the lifecycle of the Connection. Once sent, both the RA and PA state-machines transition from Initial state to Reserving state. The PA reserves resources requested using path computation and depending on the results may choose to contact other NSAs to complete the reservation process. If the Reserving process completes successfully (also in all child NSAs) a *reserveConfirm* message is sent to the RA and the Connection moves into the Reserved state.

If the reservation process fails locally or in any of the child NSAs (due for example to the failure of path computation), then the PA issues a *reserveFailed* message to the RA, and the Connection moves to the Cleaning state in both the RA and PA and issues a *terminateRequest* to all child NSAs and a terminate instruction to the local NRM..

Once the local NSA responds with a *terminate\_ok* message and any child NSAs come back with a *terminateConfirm* message, the state-machine is terminated.

## Provisioning a Connection

When the Connection is in the Reserved state the RA can send a *provisionRequest* message. This request will be treated in two possible ways depending on the arrival of the request in relation to the startTime specified in the *reserveRequest* message:

* **Manual Start:** Where the *startTime* has already passed (according to the PA local timer), receipt of the *provisionRequest* message moves to Provisioning state.
* **Auto Start:** Were the *startTime* has not yet arrived (according to the PA local timer), the Connection moves to the *AutoProvision* state and waits until the *startTime* is reached. The Connection state then moves to Provisioning without further action by the RA at the commencement of the *startTime*.

When the local NRM or child NSAs indicate that the Provisioning has been completed, the PA issues a provisionConfirm message to the RA and the Connection moves to the Provisioned state. If the Provisioning fails, a *provisionFail* message is issued by the PA to the RA. No further action is taken – the Reservation moves to Scheduled state.

Connection lifecycle remains in the Scheduled state until a) a *provisionRequest* is re-tried, b) a *terminateRequest* is received, or the Reservation expires (end\_time is reached) and is automatically terminated by the PA.

A *provisionRequest* for a Connection already Provisioned is allowed and does not affect the service instance. A *provisionConfirmed* is returned. This action is specified in order to easily recover and synchronize connections that may have otherwise had piecewise interruptions to children.

## Releasing the Provisioning state

When a Connection is in the Provisioned state, the RA can send a *releaseRequest*. When a PA receives this request the Connection moves from the Provisioned state to the Releasing state and the local NRM and child NSAs are notified to de-provision the data plane resources associated with this connection. De-provision means that the data plane is no longer operational, but the resources remain reserved for the Connection in question. When the local NRM and all child NSAs indicate that the de-provisioning has been completed, the PA issues a *releaseConfirm* to the RA and the Connection moves to the Scheduled state.

If the local NRM or any child NSA fails to release completely, a *releaseFailed* message is issued to the RA and the connection moves to Scheduled state.

A *releaseRequest* for a connection already released is allowed but no action is performed.

## Terminating a Connection

In any state the RA may send a *terminateRequest* message to the PA. The Connection will then immediately move to the Terminating state, initiate a removal of the local Reservation and forward the *terminateRequest* to all child NSAs. Once the Reservation has been removed (and if necessary the Provisioning cancelled), both locally and on all child NSAs, the Connection moves to the Terminated state and the *terminateConfirm* message is sent to the RA.

## Forced end

The PA may force the end of a Reservation. In this case a *forcedEndRequest* message is sent upstream and the state changes to Cleaning state. In the cleaning state the RA removes the Reservation.

## Querying a Connection

The RA may send a query to the PA and the PA may send a query to the RA to find the state of a Connection. The PA returns the information about all service attributes associated with the connection as resolved in satisfying the *reserveRequest*. A query can request information about one or more Connections. Two query types may be requested, a summary or a detailed query. A summary query will return local information only, i.e it will not walk the NSA tree. A detailed request will instigate an attempt to walk the NSA tree and collect information about all of the children relating to the nominated Connection.

# Connection Service primitives and attributes

## Connection Service primitives

The Connection Service primitives and their attributes are described in detail in this section. The supported Connection Service Primitives and their transport layer ACKs are shown in Figure 4. The NSI messages shaded in green for the transport part of the message and as such are a non-normative part of this specification, the remaining message format is independent of the transport layer used.



Figure 4: CS primitives

## Common types and attributes

This section describes the types and attributes which are common to all NSI messages.

### Generic types

uuidType

Universally Unique Identifier (UUID) is a URN that should be used in accordance with ITU-T Rec. X.667, ISO/IEC 9834-8:2005 and IETF RFC 4122. [3,4,5]

anyURI

xsd:anyURI is used in accordance with W3C XML schema [11].

globalReservationType

The global reservation type is of type anyURI.

connectionIdType

The connectionIdType is of type uuidType.

NsaIdType

The *NsaId* object is of type NsaIdType with the format urn:ogf:network:nsa:<NSAid>. The *NsaId* is used in the NSI protocol to identify the agent that manages a network [9, 10]. <NSAid> may be for example …..

NSnetworkType

The *NSnetwork* object is of type NSnetworkType with the format urn:ogf:network:nsnetwork:<networkId>. The *NSnetwork* identifies a network in the data-plane layer, where <networkId> could be for example layer2.netherlight.net [9, 10].

Notes on usage of the *NSnetwork* and *NsaId* objects

Note 1: The *requesterNSA* and *providerNSA* attributes in the NSI protocol use *NsaId* (i.e. not *NSnetwork*).

Note 2: The *requesterNSA* and *providerNSA* attributes are not to be used for authentication - only the transport layer (e.g. SOAP) endpoints are authenticated, i.e. the *NsaId* is only to be used for NSI layer identification.

Note 3: In v1.1 of the NSI there is a 1:1 relationship between *NSAid* and *NSnetwork*, however for flexibility in future versions of NSI more flexible assignments, e.g. one 1:many relationships may be supported.

### Common transport envelope and attributes

Figure 5 shows the message transport envelope. Each of the primitives: reserve, provision, release, terminate, query and forcedEnd are carried in a message transport envelope.



Figure 5: Message Transport Envelope

NSI and Connection Service versions

The protocol recognizes that new versions of the NSI framework and the Connection Service will be released in future. To support this, version numbering can use *NSIversion* and a *connectionServiceVersion* attributes; alternatively, web service implementations may use namespaces for this purpose.

*correlationId*

The *correlationId* attribute is of type UUID. The *correlationId* may be used to associate a response with the instance of the request that triggered the response.

*replyTo*

*The replyTo* attribute is of type *NsaIdType*, and is used as a destination for response messages. Note that this has been included since the reply-to endpoint may have a separate end-points for parent and child instances.

### Common message attributes

Some message attributes are common to all NSI messages regardless of the service type.

*requesterNSA*

The *requesterNSA* attribute identifies the source NSA of a request. The *requesterNSA* attribute also identifies the destination of a confirm or fail response. The *requesterNSA* is of type NsaIdType*.*

*providerNSA*

The *providerNSA* attribute identifies the destination NSA of request. The *providerNSA* attribute also identifies the source of a confirm or fail response. The *providerNSA* is of type NsaIdType*.*

*sessionSecurityAttr*

The *sessionSecurityAttr* is the security attribute associated with the NSI connection services session. This attribute is an opaque element that contains information that may be used to authenticate the requester NSA and authorize its request. For more details see the section of this document on security.

## Message types

This section describes the message types supported by the ver 1.1 of the NSI Connection Service.

### ReserveRequest

A Connection may be Reserved with the *reserveRequest* message. The attributes of this primitive provide information necessary to create a Reservation, this includes the start and end points of the Connection, the start and end time of the Reservation, and service parameters which describe the client framing and end-to-end performance of the Connection.

Reserve messages are of type ReserveType, the details of ReserveType are shown in Figure 6



Figure 6: ReserveType

The attributes of the *reserveRequest* are:

*reservation*

The *reservation* attribute contains sufficient information to allow a Connection Reservation to be created. This includes both the *serviceParameters* and the *path*.

*globalReservationId*

The *globalReservationId* is an optional reservation identifier that is common for the end-to-end connection. It is globally unique and can be used to correlate individual related service Reservations through the network. The *globalReservationId* is of type UUID.

*description*

The *description* attribute is optional description of the Connection instance. This attribute is type string.

*connectionId*

The *requesterNSA* assigns a *connectionId* for each Connection. The *connectionId* is a globally unique identifier, but the format is unspecified i.e. any URN can be used. Example usage: A UUID (or other globally unique identifier) can be assigned by the requesting NSA.



Figure 7: PathType

PathType is a grouping of the following attributes:

*directionality*

The *directionality* of the service describes whether the connection is unidirectional or bidirectional. The type is the string: ‘bidirectional’ or ‘unidirectional’.

*sourceSTP/ destSTP*

The source and destination STPs identify points in a Network where the service terminates and includes characteristics of the service. These identifiers consist of *stpId* and *techSpecAttrs*. \*\*\*tbc\*\*\*\*

*stpId*

The *stpId* is a globally unique label that identifies a service termination point. The format for this attribute is a tuple consisting of *NSnetwork:*<localid>.

STP: urn:ogf:network:stp:<NSnetworkid>:<localid>, where   
<NSnetworkid> = eg layer2.netherlight.net

\*\*\*tbc\*\*\*\*

*stpSpecAttrs*

The *stpSpecAttrs* are of type technolgySpecificAttributesType. The technolgySpecificAttributesType includes both *guaranteed* and *preferred* attributes. The *guaranteed* attributes MUST be met by the service. The *preferred a*ttributes MAY be met by the service. Both *guaranteed* and *preferred* are of type SAML atributeStatementType, but contents of these are not defined in the NSI CS protocol, they can only be understood in reference to the Service Definition (SD) – see Appendix B: Service Definitions for Connection Services.

*stpList*

The *stpList* is a simple ordered list if list of Service Termination Points (STPs). List order is determined by the order attribute in the OrderedServiceTerminationPointType. This is a tuple of *order* and *stpId*. *Order* is an int indicating the position in the ordered list of an STP



Figure 8: serviceParametersType

*serviceParameters* is a grouping of the following attributes:

*schedule*

The *schedule* attribute includes time parameters specifying the life of the service. The *startTime* is the Reservation start time, the *endTime* is the Reservation end time; both are of type DateTimeType [36]. The DateTimeType must use UTC time or be populated with a qualified time zone offset.

*bandwidth*

The *bandwidth* attribute describes the bandwidth of the service. The *desired* attribute refers to the target bandwidth (in Mb/s) being requested, type int.

*SeviceAttributes*

The *SeviceAttributes* are of type technolgySpecificAttributesType. The technolgySpecificAttributesType includes both *guaranteed* and *preferred* attributes. The *guaranteed* attributes MUST be met by the service. The *preferred a*ttributes MAY be met by the service. Both *guaranteed* and *preferred* are of type SAML atributeStatementType, but contents of these are not defined in the NSI CS protocol, they can only be understood in reference to the Service Definition (SD) – see Appendix B: Service Definitions for Connection Services.

### ReserveConfirmed

The attributes of the *reserveConfimedType* are shown in Figure 9.



Figure 9: reserveConfirmedType

In the case of a successful completion of the *reserveRequest* the *reserveConfirmed* response is returned with the following attributes: *requesterNSA, providerNSA and reservation.* The *reservation* attributes have the same format as in the *reseveRequest*. However, the contents of these attributes may be modified by the PA to reflect the actual Reservation.

\*\*\*\* note: which attributes are allowed to be changed by the PA?\*\*\*

### ReserveFailed

The *reserveFailed* message is of type GenericFailedType as shown in Figure 10.



Figure 10: reserveFailedType

In the case of a failed completion of the Reservation request the *reserveFailed* response is returned with the following attributes: *requesterNSA, providerNSA, globalReservationId, connectionId, connectionState and serviceException.*

*connectionState*

The *connectionState* attribute contains the current state of the Connection (as determined by the NSI CS state-machine in the providerNSA) and is of type string.

*serviceException*

The *serviceException* attribute contains a description of the reason for the *reserveFailed* response. This attribute contains the *errorId* of type string, a descriptive *text* field of type string, and *variables* attribute of type SAML attributeStatementType.

\*\*\*How is *variables* used?

## Terminating, Provisioning and Releasing messages and attributes

The *forcedEnd, provisionRequest, releaseRequest, terminateRequest* primitives all use the *genericRequestType*

The *provisionConfirmed*, *releaseConfirmed* and *terminateConfirmed* primitives all use the *genericConfirmedType*

The *provisionFailed, releaseFailed and terminateFailed* primitives all use the *genericFailedType*

**

Figure 11: genericRequest/Confirmed/Failed types

*genericRequestType*

The *genericRequestType* attributes include the *requesterNSA*, *providerNSA*, *sessionSecurityAttr* and *connectionId* as described earlier in this document.

*genericConfirmedType*

The *genericConfirmedType* attributes include the *requesterNSA*, *providerNSA*, *globalReservatinId* and *connectionId* as defined earlier in this document.

*genericFailedType*

The *genericFailedType* attributes include the requesterNSA, providerNSA, *globalReservatinId*, *connectionState* and *serviceException* as defined earlier in this document.

## Query messages and attributes

The query primitives are of type QueryType, QueryFailedType and QueryConfirmedType in accordance with Figure 12.



Figure 12: queryRequest/failed types and summary result type

### QueryRequest

The query primitive allows the RA to query the PA. Supports querying based on *connectionId* or *globalReservationId*. Filter items specified are OR'ed to build the match criteria. If no criteria are specified then all Reservations associated with the requesting NSA are returned.

A query can be requested sending the *queryRequest* message primitive with the following attributes: *requesterNSA, providerNSA, sessionSecurityAttr, operation* and *queryFilter.* The *queryFilter* contains 2 attributes: *connectionId* and *globalReservationId.* With the exception of *operation*, all of these attributes conform to the usage described in earlier parts of this document.

*operation*

The *operation* attribute is a string describing the query operation type, it can be set to either ‘summary’ or ‘detailed’. In summary mode the response populates only the *reservationSummary*, in the detailed mode the *queryConfirmed* response populates *reservationDetails* which includes details of both the local NSA and all of its children NSAs. i.e. walks the hierarchy of NSAs associated with the Connection identified in the query.

### QueryConfirm

In the case of a successful completion of the query request the *queryConfirm* message is returned. The attributes for a *queryComfirmed* response for a ‘summary’ type request is shown in Figure 12.

*queryConfirmed*

The *queryConfirmed* attributes include the *requesterNSA*, *providerNSA*, *reservationSummary* and *reservationDetails*. The usage of these attributes is the same as described earlier in this document.

*reservationSummary*

The *reservationSummary* attributes include the *globalReservationId, description, connectionId, serviceParameters, connectionState and path.* The usage of these attributes is the same as described earlier in this document.

*path*

The *path* attributes are the same as described in the *reserveRequest*.

The attributes for a *queryComfirmed* response in the case of a ‘detailed’ type request is shown in Figure 13.

**

Figure 13: queryConfirmed detailed type

*queryConfirmed*

The *queryConfirmed* attributes include the *requesterNSA*, *providerNSA*, *reservationSummary* and *reservationDetails*.

*reservationDetails*

The *reservationDetails* attributes include the *globalReservationId, description, connectionId, serviceParameters and.* With the exception of the he details of *detailedPath* the usage of these attributes are the same as described earlier in this document.

*detailedPath*

The *detailedPath* attributes include *providerNSA*, *connectionId*, *connectionState* and *pathList*. With the exception of *pathList*, these are as described in earlier parts of this document.

*pathList*

The *pathList* is a list of *ComponentPath*

*componentPath*

*ComponentPath* attributes include*, order, directionality, sourceSTP, destSTP, stpList, children*. With the exception of *children*, these are as described in earlier parts of this document.

*children*

The *children* attribute includes a list of n *child* attributes*.* This *child* attribute is of type *detailedPath* which is recursive and is defined as above.

### QueryFailed

In the case of a failed completion of the query request the *queryFailed* response is returned with the following attributes: *requesterNSA, providerNSA* and *serviceException*, these are all as defined earlier in this document.

# Connections: Transport and Service planes

The NSI Connection Service defines an abstracted representation of a Connection which is used in the service plane. This NSI Connection describes a conduit through which information is delivered from an ingress point to the egress point. While the model supports a uni-directional model of a connection, only point to point bidirectional symmetric Connections are supported in NSI version 1.0.

As illustrated in Figure 14, the Connection consists of three basic components: an ingress point where user data enters the connection, a transport section that carries the data across the network, and an egress point where user data exits the connection. These ingress and egress points are called Service Termination Points (STPs). This technology agnostic model of a Connection allows the NSI protocol to function with many different types of transport technologies.

The NSI CS protocol is an inter-network protocol, that is, it allows Connections to be built across multiple providers. The NSI v1.1 topology consists of STPs and Networks. Networks are groups of STPs configurable from a single NSA. Connections can be constructed between pairs of STPs and transit Networks - see Figure 14. In NSI v1.1 it is assumed that any two STP within a particular Network are able to be connected. This implies that all Networks are non-blocking.

Connections within a Network are intra-domain functions, and the technology details of how two STPs are actually connected is up to the local NRM and not a concern of the NSI protocols.

Two NSI Networks interconnect at a shared point known as an SDP. An SDP is a grouping of two adjacent STPs belonging to different Networks. A complete Connection can be built up by concatenating individual Connections at SDPs.

figure2

Figure : Inter-Network representation of a Connection

Once instantiated, an STP may have properties such as a framing type or a VLAN id. Some of these properties may reflect the options that can specified in the Service Definition. Labeling (cf. fiber id, wavelength, VLAN id) and aggregation (cf. combining multiple switch ports) can be modeled as a property of an STP.

The job of the physical instance of the Connection is to transport the user data (the “payload data”) across each section of the network inside a framing protocol. The framing protocol provides the necessary timing, control, and data integrity functions required to move the payload from node to node through the network. It is important to distinguish between a) the access framing protocols and b) the transport framing protocols.

It is the user payload data stream that is preserved from ingress to egress in a Connection. The transport framing can be any framing protocol as long as the end-to-end preservation requirement is honored. In fact, the only constraints on the transport framing is that the transport section be able to adapt the ingress payload data to each of the successive transport protocols that may be used along the path and ultimately be able to adapt the user payload to the egress framing at the egress point. While specification of the connection end points, access framing, and other parameters associated with a connection are defined by the connection requester (or implicitly by the service definition), the choice of the transport protocol and associated transport path parameters are explicitly delegated to the network service provider in order to allow the provider the greatest latitude in finding a valid, available, and optimal path for the connection request.

# Inter-domain vs intra-domain pathfinding

There are two levels of pathfinding related to the NSI architecture: the inter-domain pathfinding which is concerned with choosing a path across the global topology of Networks, and the intra-domain pathfinding concerned with the transport resources within the Network. NSI is concerned only with inter-Network pathfinding.

Inter-Network Connections extend across multiple networks; they are constructed by concatenating connections built across the individual networks. This is done by choosing appropriate STPs such that the egress STP of one connection corresponds directly with the ingress STP of the successive connection.

Both the Tree and Chain model reduce pathfinding to a constraint-based search over a topology to build a k-preferred path tree. The method, tree or chain, used to process a request is made exclusively in the requester NSA. No path finding algorithms (e.g. PCE or OSPF-TE) are mandated by NSI.

# STPs and Paths

## STP syntax

The STP topology uses STPs to identify network resources, where the *stpId* is a tuple of *NSnetwork:*<localid>.

*Where, NSnetwork* = urn:ogf:network:nsnetwork:<networkId>.

And<localid> = A locally unique identifier for the STP within the target network.

## Under-specified STPs

\*\*\* needs updating

## Path

The *Path* describes a route through the topology. When present in a Connection Request, the Path specifies a set of Service Termination Points (STPs) that the Connection transits, and in the order the connection must transit them. Within a *reserveRequest*, the *Path*, at a minimum, must specify the ingress and egress STPs for the Connection. Additional intermediate transit points may be included, and when present, they are considered a required constraint on the Connection’s route and must be honored.

The *Path* is made up of the following attributes: *directionality, sourceSTP/ destSTP, stpId, stpSpecAttrs and stpList* as described earlier in this document. The *stpList* is a simple ordered list if list of Service Termination Points (STPs).

## Path example in an NSA chain

An example of a *Path* in a NSA chain is presented to show how a *Path* can be described.

path_chain

Figure 15: Example of Connection managed by a NSA chain

In this example there is an Inter-Network Topology consisting of 3 networks, one per NSA. Each Network is described as a set of edge points on a network.

For this example the topology would look like this:

Network X: X:a X:b

Network Y: Y:c, Y:d, Y:e

Network Z: Z:f, Z:g

In this example, the NSAs are connected as a chain:

NSA-X(Requester) to NSA-Y(Provider), NSA-Y(Requester), to NSA-Z(Provider)

Assuming a Request comes from the Application-NSA to NSA-X to reserve a connection X:a to Z:g, then NSA-X will look in the topology and determine that to make this Connection, NSA-X will reserve a local connection from X:a to Xb, and then NSA-X must forward a request for the remainder of the connection to NSA-Y: Y:C to Z:g

NSA-Y gets this request and reserves a connection between Y:c and Y:e and requests a Reservation from NSA-Z for a connection Z:f to Z:g.

## Path example in an NSA tree

An example of a *Path* in a NSA tree is presented to show how a *Path* can be described.

path_tree

Figure 16: Example of a Connection managed by a NSA tree

The topology remains the same as for the previous example:

Network X: X:a X:b

Network Y: Y:c, Y:d, Y:e

Network Z: Z:f, Z:g

In this example, the NSAs are connected as a tree:

NSA-X(Requester) to NSA-Y(Provider) and

NSA-X(Requester) to NSA-Z(Provider)

Assuming a request comes from the Application-NSA to NSA-X to reserve a connection X:a to Z:g, then NSA-X will look in the topology and determine that to make this connection, a connection request is required locally between X:a and X:b. Next NSA-X must forward two requests:

1. To NSA-Y: Y:c to Y:e
2. To NSA-Z: Z:f to Z:g

# Authentication and Authorization

## Security Requirements

The basic security requirements of any trusted, distributed service are: 1. The requester and provider can be mutually authenticated: 2. The messages between them cannot be secretly tampered with (message integrity), 3. The provider should be able to get enough trusted information about the requester to satisfy its authorization requirements.

## Message Security

Message integrity and authentication should be provided by the transport protocol over which the NSI messages are sent. Some examples of underlying secure protocols are: a VPN, Transport Layer Security (TLS), or SOAP with digitally signed messages. The choice of this protocol is not included in this specification, but should be addressed in a Security Profile (see the NSI CSv1 Security document for an example).

## Authorization

Authorization of resource use may be based on bilateral trust agreements between an RA and a PA. In addition, the NSI connection protocol provides a means of carrying identifying information on which authorization can be based. All NSI messages contain *securityAttr* objects associated with the requesterNSA and the providerNSA. The *securityAttr* object is left opaque in the schema to facilitate various types of authorization schemes. One model that can be supported is to provide a set of mutually agreed upon attributes for the requester, The NSI CS v1.1 document specifies a schema for an *securityAttr* that contains set of defined attributes that can apply to the RA or to the user who originated the request.

# Failures and exceptions

## Service plane failures

Service plane failures are detected by the NSI transport and as such is not part of this NSI CS protocol. See Appendix A: Best Practices for NSA implementation for a discussion on handling time-outs.

## Transport plane failures

Failures in the transport plane can occur at any time, however within the framework of the NSI architecture, there are two time windows in which a transport plane failure is significant:

1. The time between the service Reservation completed and Provisioning start
2. The time between the service Provisioning completed and teardown started

The errors only need to be handled by the NSA if the Data Plane errors affect the user service.

Figure : Local/Remote Failures

Transport failure during the service Reservation and Provisioning: An element in the Data Plane becomes unavailable due to a soft or hard failure causing a Provisioning failure of a confirmed Reservation; the NRM can handle this by either reserving an alternate path as long as it meets the requested service characteristics or applying a forcedEnd to the Reservation. The local policy of a Network provider and availability of resources will determine what recovery action is taken.

Transport failure during Provisioning phase and teardown phase: In case a failure in the Data Plane affects an active Connection, the first recovery mechanisms will be triggered by the protection mechanisms Provisioned with the service. If the Connection Service is unprotected, then the failure notification will be sent to the Domain’s NSA. At that point, NSA will take appropriate action based on service and user policies by either re-routing the Connection within the Network or tearing down the service.

# Appendix A: Best Practices for NSA implementation

This appendix lists a set of best practices to ensure interoperability between NSA implementations.

## Message transport error handling

Additional error condition handling: The following set of checks is required to pass for messages to be considered vaild, otherwise a message transport layer fault will be returned:

* HTTP authentication – if the message does not have valid credentials it will be rejected with an HTTP 40x message.
* *correlationId* - needed for any acknowledgment, confirmation, or failed message. Must be unique within the context of the providerNSA otherwise the request cannot be accepted.
* *replyTo* - we will send the confirmation, or failed message back to this location. We do not validate the contents of the endpoint, just that it exists.
* *Reservation* – if the reservation parameters are not present then we reject.
* *requesterNSA* and *providerNSA* – must be present and resolve to an *NSnetwork* in topology. Also, the *providerNSA* must be the *NSnetwork* that the NSA is managing or the message will be rejected.
* *connectionId* – this is used as the primary reference attribute for Reservation state machines and must be present.
* If any of these fields are missing or invalid the NSA will return a message transport fault containing the *NSIServiceException* set to an appropriate error message. Typically this will be MISSING\_PARAMETER - "SVC0001", "Invalid or missing parameter" for this generic case and specify attributes identifying the parameter in question.

The following list of parameters should be validated when receiving a reservation message:

|  |  |  |  |
| --- | --- | --- | --- |
| *errorId* | *errorDescription* | *text* | *variables* |
| SVC0001 | MISSING\_PARAMETER | Invalid or missing parameter |  |
| SVC0002 | UNSUPPORTED\_OPTION | Parameter provided contains an unsupported value which MUST be processed |  |
| SVC0003 | ALREADY\_EXISTS | Schedule already exists for connectionId |  |
| SVC0004 | DOES\_NOT\_EXIST | Schedule does not exist for connectionId |  |
| SVC0005 | MISSING\_SECURITY | Invalid or missing user credentials |  |
| SVC0006 | TOPOLOGY\_RESOLUTION\_STP | Could not resolve STP in Topology database |  |
| SVC0007 | TOPOLOGY\_RESOLUTION\_STP\_NSA | Could not resolve STP to managing NSA |  |
| SVC0008 | PATH\_COMPUTATION\_NO\_PATH | Path computation failed to resolve route for reservation |  |
| SVC0009 | INVALID\_STATE | Connection state machine is in invalid state for received message |  |
| SVC0010 | INTERNAL\_ERROR | An internal error has caused a message processing failure |  |
| SVC0011 | INTERNAL\_NRM\_ERROR | An internal NRM error has caused a message processing failure |  |
| SVC0012 | STP\_ALREADY\_IN\_USE | Specified STP already in use |  |
| SVC0012 | BANDWIDTH\_NOT\_AVAILABLE | Insufficient bandwidth available for reservation |  |

Table : error messages

\*\*\*We will also need to agree on the format of the message/errorId.

## ACK handling

Delays on the transport layer can result in ACK arriving after the confirm/fail message. The following guidelines are recommended for handling web-service ACKs:

1. For protocol robustness, the NSA should accept any confirm/fail messages even if these are received out-of-order w.r.t. the ACK, i.e. before the associate ACK has been received.
2. The receipt of a confirm/fail message cancels out the need to receive an ACK. So the NSA should not only continue to process the confirm/fail message, but not gate on or wait for the ACK, i.e consequent-messages may be sent without waiting on the receipt of the ACK.
3. The NSA should send the ACK before sending the associated confirm/fail message.
4. The message transport layer takes care of ACK retransmission in case of a packet loss.
5. If the message transport layer is broken, the ACKs will eventually timeout and generate a message transport error that the NSA will need to handle.

## Guidelines on timeouts:

1. Timeouts should be configurable on a per operation basis and set to 2 minutes as a default.
2. Requester side timeouts: It is up to the individual provider to choose appropriate NSA timeouts for their network. As a guide the timeout should be set to 1 minute for reservations to a provider only NSA, and longer for hierarchical requests to aggregator NSAs depending on the number of levels of recursion. Provisioning requests are likely to take longer than Reservation requests. The timeout will need to be tailored to meet the response times of the participating networks.

# The requester NSA may choose to send queries to check the status of a request rather than terminating at timeout.

## Parallel processing of messages:

# The provider NSA should respond to queries even if still working on a response to a request.

## NTP servers

The server running the NSA should use NTP version 4 [8]. This will reduce the risk of clock skew between the NSAs.

# Appendix B: Service Definitions for Connection Services

A Service Definition describes, in very formal and declarative terms, the service being offered by a service provider to the consumers of that service.

The Service Definition (SD) is a machine readable textual document that identifies each attribute of the service and the range of values that are allowed for each attribute within that service. The Service Definition consists of an XML file containing a *serviceName* and a set of *serviceAttributes* that must be defined in order to fully specify a service instance.

In this respect, the SD serves as a template for the *reserveRequest* primitive. The Service Definition (specified by the Provider Agent) and the Service Request (specified by the Requesting Agent) relate to each other in the following manner:

* Each service request must specify which service offering - i.e. the Service Definition, it desires.
* The Service Request must contain the desired values for each service attribute defined in the Service Definition.
* If the Service Request does not explicitly specify a value for a service attribute found in the Service Definition, then that attribute will take a default value as specified in the Service Definition.
* A service attribute that has no default value in the Service Definition, it should be specified explicitly in the Service Request.

The following is a sample XML file defining a simple Ethernet Transport Service:

Santitize URL

<?xml version="1.0" encoding="UTF-8"?>  
<nsi:schema xmlns:nsi="file://local/Users/jerry/work/NORDUnet/OGF NSI-WG/nsi-SD-v1.0.xsd ">   
 <nsi:ServiceDefinition>  
 <nsi:ServiceName>Ethernet Transport Service</nsi:ServiceName>  
 <nsi:ServiceDesc>

Ethernet frames are transported from the ingress "Orig" NSI endpoint to  
 the egress "Dest" NSI endpoint.

</nsi:ServiceDesc>

<nsi:ServiceAttributeList>  
 <nsi:Attribute>  
 <nsi:AttrName>Payload Bit Rate</nsi:AttrName>  
 <!-- Maximum transfered bits. Since the "payload" for the ETS  
 is the actual ethernet frame, the bit rate will include all frame   
 header information as well as the payload section of the frame.  
 -->  
 <nsi:AttrMin>0</nsi:AttrMin>  
 <nsi:AttrMax>10000000000</nsi:AttrMax>  
 <nsi:AttrStep>50000000</nsi:AttrStep>  
 <nsi:AttrDefault>50</nsi:AttrDefault>  
 </nsi:Attribute>  
 <nsi:Attribute>  
 <nsi:AttrName>Payload MTU</nsi:AttrName>  
 <!-- Maximum size of the user supplied frames in bytes.   
 Note: this is a policing function and does not imply that frames  
 exceeding this length will be segmented. This parameter is used to   
 select paths that can carry frames of the specified size - it does   
 not imply segmentation of frames exceeding this size.  
 -->  
 <nsi:AttrMin>1500</nsi:AttrMin>  
 <nsi:AttrMax>9280</nsi:AttrMax>  
 <nsi:AttrDefault>1500</nsi:AttrDefault>  
 </nsi:Attribute>  
 <nsi:Attribute>  
 <nsi:AttrName>Max Frame Error Rate</nsi:AttrName>  
 <!--  
 The Max Errored Frame Rate specifies the maximum number of

errored frames that are allowed within a properly functioning

service instance each second If this frame rate is exceeded,   
 an "error" is declared.  
   
 This attribute is specified as "n" where n:= 1x10^-(n).   
 Thus larger values indicate fewer errored frames (more reliable).   
 And so a user service request specifying a number less than the   
 Max Frame Error Rate is within allowable range.  
   
 Note: this service definition does not describe how the errored   
 frames are detected.  
 -->  
 <nsi:AttrType>Integer</nsi:AttrType>  
 <nsi:AttrMinVal>0</nsi:AttrMinVal>  
 <nsi:AttrMaxVal>10</nsi:AttrMaxVal>  
 <nsi:AttrDefault>8</nsi:AttrDefault>  
 </nsi:Attribute>  
 <nsi:Attribute>  
 <nsi:AttrName>StartTime</nsi:AttrName>  
 <!--   
 Default equal NOW.  
 -->  
 <attrType>DateTime</attrType>  
 <nsi:AttrDefault>Now</nsi:AttrDefault>  
 </nsi:Attribute>  
 <nsi:Attribute>  
 <nsi:AttrChoice>  
 <nsi:AttrName>EndTime</nsi:AttrName>  
 <nsi:AttrType>DateTime</nsi:AttrType>  
 </nsi:AttrChoice>   
 <nsi:AttrChoice>  
 <nsi:AttrName>Duration</nsi:AttrName>  
 <nsi:AttrType>Time</nsi:AttrType>  
 </nsi:AttrChoice>  
 </nsi:Attribute>  
   
 </nsi:ServiceAttributeList>  
 </nsi:ServiceDefinition>  
</nsi:schema>

The SD provides a template for an NSA to consult to insure that all service attributes are fully and properly specified in the reserveRequest. Formally, a reserveRequest must fully specify all attributes associated with a Connection before the PA can allocate the resources. The set of attributes that must be specified is the set of attributes defined in the SD. The SD defines both the attributes which should be explicitly specified by the RA (i.e. they have no default value specified in the SD), and the attributes which may be omitted by the RA (the SD specifies a default value for the attribute.)

In the NSI inter-domain context, several network operators define a single *common* Service Definition for a service that they all want to deploy. They can each then take that SD back home and engineer it into their respective Networks and be assured then that the services they deploy are consistent and inter-operable across their Networks – thus substantially increasing inter-operability and reach of the end-to-end services.

The common Service Definition is a sort of ideal service: ideally, all participating Networks will conform fully to the SD. But perhaps not all Networks can meet the full range of the service attributes defined in the SD. (for instance some aspects of a SD may require forklift upgrades to meet the full capability.) The Service Definition mechanism allows Networks to offer certain variations on the service without sacrificing interoperability. For instance, perhaps some Networks can only offer 1500 byte MTU, while other may offer 9000 byte MTU. This difference is easily specified in the local Service Definitions associated with each network. The pathfinders can take this into account and the services remain compatible for Service Requests requiring conventional MTUs. Those characteristics that a network cannot meet can be adjusted in their local SD to insure that requesters know which aspects are inter-operable and which are not.

Processing of Service Definitions requires the NSA to do two things: First, it must be able to find and parse the Service Definition and use it to qualify service requests, insuring that all request parameters are fully specified. Second, the NSA must then be able to relate those requested service parameters to the attributes of the resources that are available in the network to choose a candidate path.

The first step of qualifying the request to insure it lies within the service envelope is relatively easy.

The second step of relating the service characteristics to the available resources is more complex. For instance, if the Service Definition specifies a *bandwidth* attribute with a service range of 10 to 1000 Mbps and a conforming request is received asking for 500 Mbps, then the NSA must be able to intelligently search the topology database (the resources) for those path segments that have an available capacity of 500 Mbps, tentatively reserving resources and releasing them as potential path segments are pruned.

In NSI CS v1.1, the Service Definitions are only used to qualify service requests.

**NSI Service Definitions v1.1**

NSI v1.1 uses the Service Definition to act as a template to fully specify a service request. The basic form of a Service Definition is:

<serviceDefinition>

<serviceNameDecl>

<serviceDesc>

<serviceAttributeList>

</serviceDefinition>

The Service Attibute list are of the general form:

<Attribute>

<AttrName>

<AttrValue>

</AttrName>

For NSI v1.1, the Service Definition will be encoded as an XML document conformant to an XML Schema Definition.

|  |  |
| --- | --- |
| ServiceDefinition | Comprises the specifications that fully and completely define an NSI “Service” |
| ServiceName | A string that distinguishes the service described in this SD from other services that may also be offered in the same NSI network. The name can be any length and may contain any characters printing characters. The name string carries no encoded information within an NSI context. It is simply a string. special characters are explicitly disallowed. Multi-lingual service names may be supported |
| ServiceDesc | A textual description of the service, or alternatively a URN pointing to a file containing the textual description. |
| ServiceAttributeList | A list of attributes that fully specify the service being offered. |

Table : attributes of service definition

# Contributors

Chin Guok, ESNET

Jeroen van der Ham, University of Amsterdam

Radek Krzywania, PSNC

Tomohiro Kudoh, AIST

John MacAuley, Surfnet

Takahiro Miyamoto, KDDI R&D Laboratories

Inder Monga, ESnet

Guy Roberts, DANTE

Jerry Sobieski, NORDUNET

# Glossary

Connection

A Connection is a conduit that transparently moves user information between STPs across a Network. A Connection has a set of properties (for instance, capacity, or authorization, or start time).  These properties, and their allowed range of values, are defined by a Service Definition. A Connection instance on the Transport Plane is identified by a Connection Identifier exchanged on the Service Plane. Connections are unidirectional.

Connection Service

A Connection Service is a service that allows a Requester NSA to request and manage a Connection from a Provider NSA

Connection Service Protocol

A Connection Service Protocol is the protocol that describes the messages and associated attributes that are exchanged between RA and PA.

Control and Management Planes

The Control Plane and/or Management Plane are not defined in this document, but follow common usage.

Data Plane

The Data-Plane refers to the infrastructure that carries the physical instance of the Connection, e.g. the Ethernet switches that deliver the circuit.

Edge Point

A network resource that resides at the boundary of an intra-network topology, this may include for example a connector on a distribution frame, a port on an Ethernet switch, or a connector at the end of a fibre.

Inter-Network Topology

This is a topological description of a set of Networks and their transfer functions, and the connectivity between Networks.

Network  
A Network is an Inter-Network topology object that describes a set of STPs with a Transfer Function between STPs.

Network Resource Manager (NRM)

The Network Resource Manager owns a set of transport resources and has ultimate responsibility for authorizing and managing the use of these resources. Each NRM is always associated with a single NSA.

Network Services

Network Services are the services offered by an NSA. Each NSA will support one or more Network Services.

Network Service Agent (NSA)

The Network Service Agent is a concrete piece of software that sends and receives NSI Messages.  The NSA includes a set of capabilities that allow Network Services to be delivered.

Network Service Interface (NSI)

The NSI is the interface between Requester NSAs and Provider NSAs.  The NSI defines a set of interactions or transactions between these NSAs to realize a Network Service.

Network Services Framework (NSF)

The Network Services framework describes a NSI message based platform capable of supporting a range of Network Services.

NSI Message

A NSI Message is a structured unit of data sent between a Requester NSA and a Provider NSA.

Provision/Provisioning/Provisioned

Provisioning is the process of creating the physical instance of a Connection in the data plane. A Provisioned Connection is ready to carry user data. This term has a formal definition in the CS state-machine.

Requester/ Provider NSA (RA/PA)

An NSA acts in one of two possible roles relative to a particular instance of an NSI.  When an NSA requests a service, it is called a Requester NSA. When an NSA realizes a service, it is called a Provider NSA.  A particular NSA may act in different roles at different interfaces.

Release/Releasing/Released

Releasing is the process of de-provisioning resources on the data-plane. When a Connection is Released on the data-plane, the Reservation is retained. This term has a formal definition in the CS state-machine.

Reserve/Reserving/Reservation

A Provider Agent holds a Connection Reservation. This Reservation has start and end times, ingress and egress STPs and an associated set of additional local resources necessary to build the Connection locally. This term has a formal definition in the CS state-machine.

Service Definition

The Service Definition consists of a set of attributes that formally and explicitly define the complete scope of a service offering. Each provider defines its service with an SD, each request identifies requirements in terms of SD attributes, and each Connection has an associated Service Definition instance.

Service Demarcation Point (SDP)

Service Demarcation Points (STPs) identify a grouping of two Edge Points at the boundary between two Networks.

Service Termination Point (STP)

Service Termination Points (STPs) identify the Edge Points of a Network in the intra-network topology.

Service Plane

The Service Plane is a plane in which services are requested and managed; these services include the Network Service. The Service Plane contains a set of Network Service Agents communicating using Network Service Interfaces.

Terminate/Terminating/Terminated

Terminating is the process which will completely remove a Reservation and Release any associated Connections. This term has a formal definition in the CS state-machine.

Transport Plane

The Transport Plane contains is the set of transport equipment and associated resources that carry user data through the network.

# Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

# Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

# Full Copyright Notice

Copyright (C) Open Grid Forum (2008-2012). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

# References

1. OGF GFD.173: “Network Service Framework v1.0”, http://www.gridforum.org/documents/GFD.173.pdf
2. IETF RFC 5905, Network Time Protocol Version 4: Protocol and Algorithms Specification
3. IETF RFC 4122, A Universally Unique IDdentifier (UUID) URN Namespace
4. ITU-T Rec. X.667 Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components
5. ISO/IEC 9834-8:2005 Information technology -- Open Systems Interconnection -- Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components
6. IETF RFC 4655, "A Path Computation Element (PCE)-Based Architecture", http://www.rfc-editor.org/rfc/rfc4655.txt
7. ISO 8601:2000 “Data elements and interchange formats — Information interchange — Representation of dates and times” or xsd dateTime
8. IETF RFC 5905, “Network Time Protocol Version 4: Protocol and Algorithms Specification”, http://tools.ietf.org/html/rfc5905
9. IETF RFC 6453, “A URN Namespace for the Open Grid Forum (OGF)”, <http://tools.ietf.org/html/rfc6453>
10. OGF GFD-CP.191 "Procedure for Registration of Subnamespace Identifiers in the URN:OGF Hierarchy”, http://www.ogf.org/gf/docs/
11. W3C XML “Schema Definition Language (XSD) 1.1 Part 2: Datatypes”, http://www.w3.org/TR/xmlschema11-2/#anyURI