**GLIF 2011 Rio NSI Plugfest Guide and Interop Challenges**

**Version 1.1**

**18 August 2011**

The NSI Plugfest 1.0 will be conducted at the GLIF 2011 meeting in Rio de Janiero, Brazil.

The Plugfest is intended to demonstrate *substantive progress* towards deployable software conforming to the OGF NSI Connection Service protocol standard. The Plugfest will consist of a series of “Challenges” that will showcase both a) the ability of the multiple implementations to successfully interoperate with one another, and b) the capabilities of the NSI Connection Service protocol. It is anticipated that the Plugfest will also expose shortcomings in the NSI specification or interpretations that must be resolved.

The interop test will be conducted among the following NSI implementations developed by the indicated organizations:

1. AutoBahn (PSNC/Dante)

2. dynamicKL (KISTI)

3. G-LAMBDA/AIST (AIST)

4. G-Lambda/KDDILabs (KDDI Labs)

5. OpenDrac (SURFnet)

6. OpenNSA (NORDUnet)

7. Oscars (ESnet)

The Interop Process:

The Plugfest will consist of a pre-defined set of technical interoperability tests the participants will be asked to perform, and it will culminate with the live demonstrations and continued testing at the GLIF Conference itself. Therefore, the interoperability effort will in fact be an ongoing effort commencing in August and continueing through the GLIF Plugfest. We hope and anticipate most of the technical “Challenges” will have been successfully tested prior to arriving in Rio, and that much of the demonstrations in Rio will be public interaction and presentation with the developers.

The Plugfest will be designed around several technical “Challenges”. Each Challenge will test key features of the protocol implementation and show that the implementation(s) interoperate according to the specification. The test scenarios in each Challenge consist of a scripted sequence of protocol events to be initiated by one or more of the Network Service Agents involved in the test. The test run will dump or display the sequence of protocol messages and other events that occur as a result. The Challenges may have several permutations to be run in order to show each NSA in different functional roles of the test scenario (for example – a particular NSA might serve as a RA in one run, and as the PA in another run.)

The Plugfest Challenges will address the NSI Connection Service protocol version 1.0 only. The Challenges are very limited in scope. And at this early stage of development, they focus on the normal life cycle of a connection instance from initial reservation to the final termination. Evaluation of correct protocol functioning will be based upon the externally observed behavior of the protocol implementations as described in the protocol specification document. The Plugfest does not intend to evaluate implementation specifics as relates to the internal design or methodology of each NSA.

The Challenges are *not* intended to be a suite of conformance tests. Rather, they are intended to show tangible progress towards real deployable and interoperable implementations of the protocol, and to aid the developers in insuring that they are all interpreting the NSI CS specification in a consistent manner. At this early stage in the development efforts we are trying assist the NSI implementation efforts and to portray the protocol and the emerging implementations in their best light - not focus on the bugs or other non-NSI features or limitations of each implementation.

The Challenges for Plugfest Rio will essentially address four basic aspects:

1. Self consistent messaging and lifecycle management of each implementation,
2. Cross implementation consistency in messaging and lifecycle management,
3. Segmentation and correct service tree construction and processing,
4. Query processing.

These do not constitute a comprehensive set of NSI features or functionality. These are simply a set of key basic capabilities that are intrinsic to the NSI CS protocol. (A more complete and formalized conformance test suite might be a task for the future, but such is beyond the scope of this document or the Rio Plugfest.)

The Challenges will be available to the Plugfest participants well in advance of the live testing activities at the GLIF Rio conference. This will give the development teams time to set up and begin testing of their implementations prior to the live Interop testing/demonstrations in Rio.

Each Challenge depends upon the capabilities tested in prior Challenges, and so they should be performed in the order defined. This allows each Challenge to focus on the next key functional area of the protocol or framework. At the beginning of each test run, all participating NSAs will be allowed a “cold start”. A cold start does two things: 1) It allows all the NSAs to begin the test with a clean and freshly initialized software runtime environment. This is intended to help avoid any issues not directly related to the NSI operation of the software. 2) It establishes a known and consistent state among all participating agents with regards to reservation databases, topology, resource allocations, AA information, or any other configuration issues.

Each of the NSI implementations will be expected to present certain results to verify that the implementation is indeed performing as required by the protocol. Generally, in the Rio interoperability tests, the information to be dumped for inspection will be 1) the NSI protocol messages, 2) NSI protocol events (such as timeouts), 3) possibly state information of connection instances, and 4) possibly information relating to the southbound NRM interactions. The format of this output is left to the implementer but should be easily interpreted by other software engineers familiar with the protocol and NSI framework. It will be necessary for each implementation to have the ability to script the Challenge, i.e. provide a mechanism for pacing the sequence of actions that must take place in order to stage the test.

At the GLIF Plugfest, the implementation teams will be expected to show these Challenges live. Since the specifics of each test will generate detailed results not easily evaluated by the general audience and not suitable for a large presentations, the “live” demonstrations of these tests will be done in the interop venue for the NSI evaluation process, not as part of the general GLIF program presentations. Attendees are invited to look over the shoulder while these tests are run and to interact with the developers. Many of the advanced challenges will involve multiple implementations. The teams should be prepared to work together to run these Challenges. The teams should be prepared to present each Challenge in an organized and easily repeatable and easily interpreted manner. The organizers will work with the participants to set up a schedule of live demonstrations of the Challenges. This will hopefully streamline the Plugfest. The Plugfest will be continuously active from Tuesday thru Thursday from 9am to 5pm. GLIF attendees will be invited to stroll thru the Interop venue at any time, and watch the testing and/or interact with the implementation teams.

LCD monitors and/or a printer will be provided for the Plugfest participants to aid in presenting and/or analyzing the results in order to show success in each Challenge.

The results of the Challenges will be evaluated by active personnel in the OGF NSI Working Group. In case of discrepancy, the NSI WG will clarify the “correct” operation and that clarification will be announced to all the implementation teams and recorded for a subsequent errata circular. The implementation teams may appeal an “incorrect” evaluation to have the specification reviewed and possibly modified in cases of conflicting requirements.

Note: While this Plugfest Guide is very detailed in the specifics of the testing process, it should be re-iterated that the purpose of the Interop Plugfest is to aid the progress and success of these implementations. These are guidelines and an attempt to organize the interop testing. This is NOT a conformance test and no grand certifications will be forthcoming. We are trying to get *all* of these implementations to a point of common interoperability – not weed out substandard implementations. Therefore, the process of testing and evaluation should be treated in a flexible, helpful, and collaborative manner to be as productive and useful to the implementers as possible, but in keeping with the specification. While all of the Challenges may not all be successfully completed at Rio, the “press release” will describe the overall progress of NSI development as demonstrated in the Plugfest – which we hope will be substantial. GLIF may elect to host a follow-up Plugfest at future conferences to propel the NSI development and deployment effort forward, and/or may elect to support the creation of a conformance test suite.

**Topology treatment for the GLIF Plugfest:**

Network topology is a critical component of any dynamic circuit provisioning system. NSI CS protocol version 1.0 has only a very rudimentary notion of “NSI Topology”. This NSI Topology model is solely an *inter*-domain adjacency model and does not officially or formally define any physical layer or *intra*-domain characteristics. In essence, an NSI Network service asserts that connectivity is possible amongst any of the NSI Endpoints that are members of that network service. The protocol specification is very careful to adhere to this technology agnostic model in both the NSI Framework and the CS protocol primitives, and so the NSI interoperability Challenges will likewise adhere very carefully to this model.

Topology is a complex topic, and there is no current topology standard that either conforms to NSI, or that is a common consensus model. The GLIF Distributed Topology eXchange working group (DToX) is exploring these topology issues and has offered an interim working model that has been amended to support NSI and the Rio Plugfest. The Plugfest organizers and participants have agreed to use this DToX topology model, represented textually using the OWL data description language, to describe the inter-domain topology. For the Plugfest Rio, a single static “global” (inter-domain) topology will be defined for each Challenge. This one static topology will be available to every NSA involved in that Challenge so that all NSAs will be operating with the same common view of the world.

While NSI as a protocol standard and framework does not deal with intra-domain provisioning issues (leaving this to the local NRMs), the Network Service Agents do, in a practical sense, need to interact with local NRMs and the intra-domain resource topologies they manage. Thus, it becomes necessary for any test of real NSA code to have a realistic topology to work from, such that the local NSA-NRM interaction can function properly and thereby support the correct functioning of the inter-domain NSI layer. For the Rio Plugfest, this means that in addition to defining the inter-domain topology and ENNI peering points, the topology provided for the Challenges must insure that the ENNI peering points are defined identically in each of the peering networks. While the peering point characteristics are intra-domain information, and would normally be left to the peering networks to work out bi-laterally between themselves, for the Rio Plugfest these peering points will all be configured apriori to have a single simple set of specific physical characteristics. Essentially, for the Rio Plugfest,the intra-domain physical characteristics of all peering points and end systems will be defined to be:

1. Maximum allocatable capacity := 1000 megabits per second
2. All peering points are untagged – i.e. no labels.

This should be sufficient to meet the object of Rio: to show interoperability. Perhaps subsequent public tests will provide a more varied topology in order to exercise more sophisticated resource selection processes.

The Rio Plugfest topology (the amended DToX topology model) consists of three inter-domain topological objects: a “NSnetwork” object, a “STP” object, and an “NSA” object. The DToX topology also includes three intra-domain objects: a “Node” object, a “switchMatrix” object, and a “Port”. These six objects are defined in more detail below.

*Inter*-domain resource classes:

1. NSnetwork – This resource represents an NSI Network.
   1. Attribute: Name – The NSI Network name, e.g. “EtherTransportService.Netherlight.net” or “Bonaire”.
   2. Relation: hasSTP - a set of relations that reference “STP” resource objects that are members of this network.
   3. Relation: hasComponent – a set of relations that reference the Node objects that comprise the intra-domain network elements.
   4. Relation: managedBy – reference to a single NSA object.
2. STP – This resource represents NSI Endpoints.
   1. Attribute: Name – the NSI Endpoint name.
   2. Relation: connectsTo – reference to the remote STP object that this STP peers with. (Note: STPs only “connectTo” other STPs in other networks.)
   3. Relation: mapsTo – reference to the intra-domain Port resource that corresponds to this STP. The mapsTo relation is considered intra-domain information for purposes of NSI.
3. NSA – This resource represents an NSA, containing contact information associated with that NSA.
   1. Attribute: Name
   2. Attribute: csProviderEndpoint – a string, presumably a URL, that constitutes the address for sending messages to this NSA.
   3. Relation: manages – reference to an NSnetwork object that this NSA manages.

*Intra*-domain resource classes:

1. Node – This resource describes a switching element within a network. For Rio, this is considered an intra-domain object from an NSI perspective.
   1. Attribute: Name
   2. Attribute: location – latitude, longitude
   3. Relation: partOf – reference to the NSnetwork object this Node is a member of.
   4. Relation: hasSwitchMatrix – reference to an optional switchMatrix resource object.
   5. Relation: hasPort – a set of relations that reference Port objects.
2. switchMarix – This resource describes detailed technology specific switching capabilities. This object constitutes the “transfer function” of the device and currently supports only any-to-any port level switching.
   1. Attribute: Name
   2. Attribute: canSwap – indicates whether this matrix can swap labels from input port to out put port. Not used for Rio.
   3. Relation: part of – reference to the Node object this matrix is a component of.
   4. Relation: hasPort – set of references to Port objects that comprise the I/O interfaces for this switchMatrix.
3. Port – This resource describes the physical capabilities and state of an I/O interface that is part of a network element.
   1. Attribute: Name
   2. Attribute: labelSet – the set of available labels. An empty “labelSet” attribute implies an untagged Port. This field must be empty for all inter-domain Ports for Rio.
   3. Attribute: maxCapacity – The maximum allocatable capacity for this Port. This is specified as megaBytes/second.
   4. Relation: connectsTo – reference to another intra-domain Port object that is connected to this Port. For Rio, this relation must only reference other Port objects within the local NSnetwork.
   5. Relation: mapsTo – reference to an STP object that corresponds to this Port. The STP is the interdomain representation of this Port.

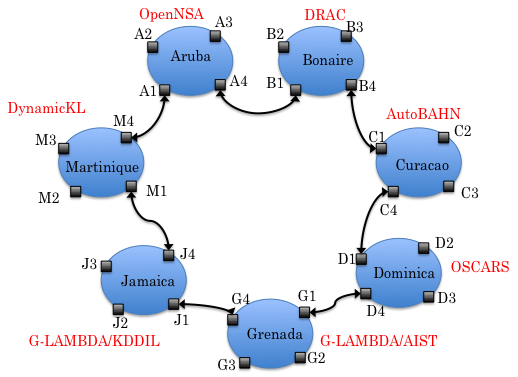
Intra-domain topology issues are, in general, out of scope for this NSI CS interoperability demonstration. However, two aspects of intra-domain resources must be consistent with the inter-domain topology: First, it is necessary to map the inter-domain NSI Endpoints (STPs) to the corresponding intra-domain topological constructs recognized by the NRM. For Rio, the intra-domain topological construct will always be a Port resource. Second, it is crucial that the internal Ports that correspond to the NSI Endpoints have identical physical characteristics– e.g. they must have the same maximum capacity, the same framing, same label sets, etc.

The inter-domain STP resource and the intra-domain Port resource both support a “connectsTo” relation that indicates another STP/Port respectively that they are connected to in the conventional (fiber jumper) sense. External STPs connect to other external STPs, and internal Ports connect to other internal Ports.

The inter-domain STP resource provides a “mapsTo” relation that identifies the corresponding intra-domain topological construct recognized by the NRM. Likewise, that intra-domain construct (typically a physical “Port” ) will also need a similar relational mechanism that maps back to the corresponding NSI Endpoint (STP). The mapsTo relations, the connectsTo relations, and the NSnetwork and Node objects define a comprehensive topology that enables end-to-end path computation and selection.

Note: not all internal ports have an STP associated with them. But all advertised NSI Endpoints (STPs) must map to an internal Port.

The specific topology that will be used at Rio will consist of seven networks interconnected at the data plane to form a simple ring. Each NSI implementation is randomly assigned to a particular Network in the configuration. The following is the ring topology:



In this topology, for instance, the AutoBAHN NSA implementation will manage the NSI Network “Curacao”, with STPs C1, C2, C3, and C4. Curacao has ENNI peering connections to network Bonaire and network Dominica.

This global topology will be provided in a .OWL file generated by the SNE network editor. Each NSA involved in a Challenge will need to import the topology associated with that Challenge into their respective internal topology database.

The Network assignments and NSA URL information is as follows:

|  |  |  |
| --- | --- | --- |
| Aruba | Open NSA (NORDUnet) | http://orval.grid.aau.dk:9080/NSI/services/ConnectionService |
| Bonaire | DRAC (SURFnet) | <https://dracproxy01.surfnet.nl:8443/nsi/ConnectionServiceProvider> |
| Curacao | AutoBAHN  (PSNC) | http://194.132.53.174:9080/nsi/ConnectionProviderPort |
| Dominica | OSCARS  (ESnet) |  |
| Grenada | G-LAMBDA AIST |  |
| Jamaica | G-LAMBDA KDDI Labs |  |
| Martinique | DynamicKL (KISTI) |  |

Note: Path selection is a component of advanced Challenges, but this is primarily intended to demonstrate the inter-domain NSI path segmentation and service tree construction rather than NRM specific resource allocation schemes.

Note: The specifics of NRM interaction is not specified in the NSI CS standard, but the NRM does constitute an important part of the NSI/NSA framework and will be of some interest to the Plugfest attendees and participants. Thereore, it may be useful for the NSA to dump the proxy commands and/or NRM interaction details to aid in debugging and/or to show how NSI service features are being communicated to and/or processed by the underlying NRM.

One final comment on topology: It is the intention of the Plugfest to test the NSI code implementations – not the organizations funding the development or the networks that may ultimately deploy one or another of the implementations. Therefore, the Challenge topology consists of a collection of homogeneous and artificial NSI Networks that are explicitly chosen to avaoid any resemblance to existing network organization or infrastructure. The mapping of NSA implementations to these test networks is completely arbitrary.

**Authentication and Authorization:**

For purposes of this initial interop test at GLIF Rio, only NSA-NSA *authentication* will be enforced. I.e., each NSA will be required to authenticate according to the NSI CS protocol requirements to the other NSA. *Authorization* of NSA sessions will use a single “shaman” authorization credential that will be accepted by all NSAs as omnipotent – i.e. such credentials will never fail authorization of an otherwise valid CS protocol primitive. Further, the shaman authorization shall be the only recognized authorization credentials. At this stage in the implementation of NSI CS, we want to evaluate proper functioning of the NSI CS protocol itself, not internal authorization or resource allocation schemes. By having a single Authorized usr for the Interop, we are able to show that Authorization is being performed without delving into complex authorization policies or protcols. The specifics of these credentials is TBD.

**Challenge 1.1 Self-consistent Protocol Agent**

Objective:

Show that each NSI CS 1.0 implementation can be deployed as multiple independent agents that are able to communicate with each other using the NSI CS protocol to establish and progress a connection through the entire lifecycle.

Description:

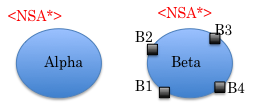
Challenge 1 defines two Test Scenarios: 1.1 and 1.2. Scenario 1.1 is essentially a manual sequencing of a reservation though its lifecycle. Scenario 1.2 addresses automated triggering of lifecycle transitions at start and end along with a release/provision sequence midway. So each NSA implementation will have to run two scripted Test Scenarios to complete this Challenge.

In this Challenge, each implementation will be required to establish two completely independent NSAs that are able to use the NSI CS protocol to sequence a connection though its full lifecycle. Each NSA must run on a separate physical server platform, or otherwise reasonably show that such operation is possible (e.g. via a recognized VM environment, etc.)

One NSA will act as a Requesting Agent (RA), and the other as a Provider Agent (PA). The RA will request a connection reservation across the PA’s network. Each NSA will have a minimal network over which it presides. For this Challenge, the RA request will not require the PA to perform any inter-domain segmentation- the reservation request for this Challenge will be between two Endpoints in the PA’s network.

Topology:

The topology below should be used for Challenge #1. In this Topology, network Alpha will act as the RA, and network Beta as the PA. The <NSA\*> indicates that each implementation should insert their NSA for Challenge #1. (It is acceptable for the development teams to replace network Beta with their assigned network in the Rio Ring topology, thus enabling their primary NSA server to use the same topology for all tests.



The RA network is not actually used in Chalenge #1 except to act as the RA NSAID in the service request(s); it is defined and present in order to comply with NSI specifications for the NSA RA.

The fully specified NSI Endpoint names will be a URN of the form: “urn:ogf:network:stp:Beta:B1” (this should reflect the final NSI CS specification.) A fully specified NSI Network name will be a URN of the form: “urn:ogf:network:NSnetwork:Beta”

(Note: The Network and Endpoint names in the Plugfest topologies were chosen so as to not resemble or endorse any existing network domains, hardware vendor, NRMs, or configuration.)

**Test Scenario 1.1: Basic Manual (RA driven) LifeCycle**

This exercise will begin with the cold start initialization of all NSAs. A “cold start” implies that an NSA will have no prior or existing reservations, all topology information will be initialized from the original Challenge topology, and the software will have a clean and fresh runtime environment. Once all NSAs have completed initialization, the exercise can begin.

This exercise will perform a manual sequencing of the connection through its Lifecycle:

1. T=00:00 Reserve. At relative time T=00:00, NSA Alpha will issue a reserveReq() primitive to NSA Beta and await the results. The request will specify the following parameters:
   * + Source STP: Beta:B1
     + Destination STP: Beta:B4
     + Start Time: T=01:00)
     + End Time: 0 (no end time; an indefinite duration)
     + Capacity: 1000 Mbps
     + Directionality: Bidirectional
     + Service Authorization: “Shaman”
     + Global Reservation ID: null

A reserveResp=Confirmed message should be sent from Beta PA to Alpha RA once all the arrangements have been made in the PA.

1. T=01:20 Provision. Upon reaching the Start Time plus 20 seconds, the Alpha RA shall send a provisionReq() primitive to the Beta PA (a “manual start”.) Since the provisionReq() message is sent after the scheduled Start Time, the PA should immediately commence provisioning the connection. Appropriate connection state changes and/or NRM interactions should be evident. Upon completing the provisioning process across Beta, Beta should send a provisionResp=Confirmed back to Alpha.
2. T=02:00 Release. At two minutes, the connection should be in service and in a quiescent state. The Alpha RA will issue a releaseReq() primitive to the Beta PA. The circuit should be immediately released and a releaseResp=confirmed message sent from Bonaire to Aruba.
3. T=03:00 Terminate. The Alpha RA shall issue a terminateReq() primitive to completely cancel the reservation.
4. The zombie ConnectionID should timeout causing an NSI Event as it is deleted.

Results:

The NSAs are to display the following information:

1. All NSI protocol messages that are sent or received. All information elements of those messages should be included in the log entry/printout.,
2. All other NSI events that occur. NSI events are those non-message events described in the specification that cause a transition in the State Machine. (E.g. a reservation Start Time or End Time arriving, or a response Timeout, etc.) Relevant information should be included such as the type of event, the ConnectionID associated with the event, etc.
3. All state changes of all ConnectionIDs. Note: this is the state of the Connection – not the protocol state machine. This Challenge requires all ConnectionID status transitions be logged/dumped. The ConnectionID, the triggering event, and both the old and new state should be logged.
4. Each log entry should include a timestamp with at least millisecond accuracy.
5. Each NSA should have the ability to also dump the WS Message Transport Layer if necessary or requested to resolve messaging issues at the NSI protocol layer.

**Test Scenario 1.2: Auto (PA driven) LifeCycle.**

In Scenario 1.2, we will use the same topology as 1.1 but exercise a slightly different Life Cycle sequence. In this exercise, we do a Auto Start, an iterative release/provision cycle, and let the reservation expire automatically.

This test scenario will begin with the cold start initialization of all NSAs. Once all NSAs have completed initialization processing, the exercise can begin.

* + 1. T=00:00 Reserve. NSA Alpha will issue a reserveReq() primitive to NSA Beta and await the results. The request will specify the following parameters:
* Source STP: Beta:B2
* Destination STP: Beta:B3
* Start Time: T=01:00
* End Time: T=04:00 (a 3 minute duration)
* Capacity: 1000 Mbps
* Directionality: Bidirectional
* Service Authorization: “Shaman”
* Global Reservation ID: null

A reserveResp=Confirmed should be received from the Beta PA.

* + 1. Provision. Immediately after receiving the reservation confirmation message, the Alpha RA shall send a provisionReq() message to the Beta PA. A state transition should occur in the Beta PA allowing the Start Time to trigger the provisioning function (auto-start). A provision Confirmed message should be sent from Bonaire to Aruba once the provisioning across Bonaire has completed.
    2. T=02:00 Release. The Alpha RA will issue a releaseReq() primitive to the Beta PA. The circuit should be immediately released and a releaseResp=confirmed message sent from Beta to Alpha. Corresponding connection state transitions should be observed as well.
    3. T=03:00 Re-provision. Alpha shall send a provisionReq() message to Beta to [re-]provision the connection back into service. Appropriate messages should be seen and associated connection state transitions should occur.
    4. T=04:00 Terminate. The RA and PA should both see the End Time event and this should trigger the complete tear down of the reservation.
    5. A Zombie Timeout associated with the ConnectionID should occur causing an NSI Event to be logged. This terminates the test.

Results:

The NSAs are to print/log following information:

1. All NSI protocol messages that are sent or received. All information elements of those messages should be included in the log entry/printout.,
2. All other NSI events that occur.
3. All state changes of all ConnectionIDs. Note: this is the state of the Connection – not the protocol state machine. The ConnectionID, the triggering event, and both the old and new state should be logged.
4. Each log entry should include a timestamp with at least millisecond accuracy.
5. Each NSA should have the ability to also dump the WS Message Transport Layer if necessary or requested to resolve messaging issues at the NSI protocol layer.

**Challenge 2. Multi-Implementation Interoperability**

Objective:

This Challenge is intended to show that the different NSI implementations can interoperate correctly with one another using the NSI CS protocol to establish and progress a connection through the entire lifecycle. This Challenge is similar to Challenge 1 but now the RA and PA will be NSAs from different development efforts.

Upon successfully completing the entire Chal\ enge #2 test suite, all of the NSI implementations will be confident that their message transport layer, their NSI protocol messaging, and their state machines (protocol behavior) are functioning in a consistent manner across all the participating implementations.

Description:

In this Challenge, each NSI implementation will be required to interact with another NSI implementation. Each implementation will be required to run Test Scenarios 2.1 and 2.2, serving in the role of the RA and in the role of PA in each. This constitutes four scripted tests that each implementation must successfully complete.

The RA will request a connection reservation across the PA’s network. Each NSA will have a minimal network over which it presides. The RA request will not require the PA perform any inter-domain segmentation- the reservation request for this Challenge will be between two Endpoints in the PA’s network.

The test matrix for this Challenge is defined below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| \ PA RA \ | A. OpenNSA | B.  OpenDRAC | C. AutoBAHN | D.  OSCARS | G.  G-Lambda  AIST | J.  G-Lambda  KDDI Labs | M.  DynamicKL |
| 1. OpenNSA | Chal 1. | Chal 2. | - | - |  |  | Chal 2 |
| 2. OpenDRAC | Chal 2. | Chal 1. | Chal 2. | - | - |  |  |
| 3. AutoBAHN | - | Chal 2. | Chal 1. | Chal 2. | - |  |  |
| 4.  OSCARS | - | - | Chal 2. | Chal 1. | Chal 2. |  |  |
| 5.  G-Lambda  AIST |  | - | - | Chal 2. | Chal 1. | Chal 2 |  |
| 6.  G-Lambda  KDDI Labs |  |  |  |  | Chal 2. | Chal 1. | Chal 2 |
| 7. DynamicKL | Chal 2 |  |  |  |  | Chal 2. | Chal 1. |

A full test regimen addressing each permutation would be prohibitively complex and time consuming and of questionable interop value. The reduced scope of Challenge 2 tests above will provide a transitive interoperability assertion, i.e. If OpenNSA can interact with OpenDRAC, and OpenDRAC interoperates with AutoBAHN, then we will assume that OpenNSA would interoperate just fine with AutoBAHN. This reduced matrix will still require two tests scenarios to be run for each matrix box, and each Implementation participates in four boxes, thus each implementation will need to run eight tests to complete the Challenge.

Note: Transitive Interoperability is sufficient for our purposes at the Rio Plugfest. A “reference implementation” is commonly used for this purpose – I.e. an implementation that acts as a known correct implementation. All other implementations can test against the reference to provide initial conformance verification. NSI does not have a “reference” implementation at this point.

Topology:

The Rio Ring topology will be used for Challenge 2. The role of each NSA implementation (RA or PA) is indicated in the test matrix. Both of the Challenge 2 test scenarios, 2.1 and 2.2, are to be run in each configuration. Using the OpenNSA and OpenDRAC as examples from the test matrix, the following scenarios will occur:

Matrix element 2A: OpenNSA/Aruba is the PA, OpenDRAC/Bonaire is the RA.

Run scenarios 2.1 and 2.2.

Matrix element 1B: OpenNSA/Aruba is the RA, OpenDRAC/Bonaire is the PA.

Run scenarios 2.1 and 2.2.

In each test matrix element, the associated networks are indicated by the NSA under test. The Test scenarios described below should amend the scripts to reflect the appropriate networks under test in each scenario.

Upon completing the assigned permutations of the Challenge #2 tests, a consensus confirmation of the protocol message interpretation and handling will be evident. I.e. Seven independently developed versions of the protocol will have been shown to interpret the basic protocol messages in a consistent and interoperable manner. Subsequent Challenges therefore will assume the protocol messaging and associated basic lifecycle sequencing are working correctly among all of the implementations and so will focus more on correct functioning of the NSAs with respect to the NSI framework and service tree.

**Test Scenario 2.1: Basic Manual (RA driven) LifeCycle**

This exercise will begin with the cold start initialization of all NSAs. Once all NSAs have completed initialization processing, the exercise can begin. This exercise will perform a manual sequencing of the connection through its Lifecycle:

* + 1. T=00:00 Reserve. NSA Aruba will issue a reserveReq() primitive to NSA Bonaire and await the results. The request will specify the following parameters:
* Source STP: Bonaire:B1
* Destination STP: Bonaire:B4
* Start Time: T=01:00
* End Time: 0 (no end time; an infinite duration)
* Capacity: 1000 Mbps
* Directionality: Bidirectional
* Service Authorization: “Shaman”
* Global Reservation ID: null

A reserveResp=Confirmed should be received from the Bonaire PA.

* + 1. T=01:20 Provision. Upon reaching the Start Time+00:20 the Aruba RA shall send a “provisionReq” primitive to the Bonaire PA. (Manual Start) The connection should already be in a state awaiting the provisionReq() msg to be sent from Aruba to Bonaire to trigger the actual provisioning process in Bonaire. A Connection status transition should occur to “provisioning” in Bonaire upon receipt of the provisionReq() message. Upon completing the provisioning across Bonaire, Bonaire should send a provisionResp=Confirmed to Aruba. A Connection status transition should be observed going from Provisioning to InService.
    2. T=02:00 Release. At the two minutes, the connection should be in service and in a quiescent state. The Aruba RA will issue a releaseReq() primitive to the Bonaire PA. The circuit should be immediately released and a releaseResp=confirmed message sent from Bonaire to Aruba.
    3. T=03:00Terminate. The Aruba RA shall issue a terminateReq() primitive to completely cancel the reservation. A connection status transition should occur from Released to Terminating, and then to Terminated.
    4. The zombie ConnectionID should timeout causing a Connection state transiton and an NSI Event as it is deleted.

Results:

The NSAs are to print/log following information:

1. All NSI protocol messages that are sent or received. All information elements of those messages should be included in the log entry/printout.,
2. All other NSI events that occur.
3. All state changes of all ConnectionIDs. Note: this is the state of the Connection – not the protocol state machine. The ConnectionID, the triggering event, and both the old and new state should be logged.
4. Each log entry should include a timestamp with at least millisecond accuracy.
5. Each NSA should have the ability to also dump the WS Message Transport Layer if necessary or requested to resolve messaging issues at the NSI protocol layer.

**Test Scenario 2.2: Auto (PA driven) LifeCycle.**

In Exercise 2.2, we will use the same topology but exercise a slightly different Life Cycle. In this exercise, we do a Auto Start, an iterative release/provision cycle, and let the reservation expire automatically.

This exercise will begin with the cold start initialization of all NSAs. Once all NSAs have completed initialization processing, the exercise can begin.

* + 1. T=00:00 Reserve. NSA Aruba will issue a reserveReq() primitive to NSA Bonaire and await the results. The request will specify the following parameters:
* Source STP: Bonaire:B1
* Destination STP: Bonaire:B4
* Start Time: T=01:00
* End Time: T=04:00 (3 minute duration)
* Capacity: 1000 Mbps
* Directionality: Bidirectional
* Service Authorization: “Shaman”
* Global Reservation ID: null

A reserveResp=Confirmed should be received from the Bonaire PA.

* + 1. Provision. Immediately after receiving the reserveResp=Confirmed message, the Aruba RA shall send a provisionReq() primitive to the Bonaire PA. A state transition should occur allowing the Start Time to trigger a provision (autostart). Another state transition should occur in Bonaire upon reaching the start time causing provisioning to proceed in Bonaire. A provisionResp=Confirmed should be sent from Bonaire to Aruba once the provisioning across Bonaire has completed.
    2. T=02:00 Release. The connection should be in service and in a quiescent state. The Aruba RA will issue a releaseReq() primitive to the Bonaire PA. The circuit should be immediately released and a releaseResp=confirmed message sent from Bonaire to Aruba. Corresponding state transitions should be observed as well.
    3. T=03:00 Re-provision. Aruba shall send a provisionReq() message to Bonaire to [re-]provision the connection back into service. Appropriate messages should be seen and associated connection state transitions should occur.
    4. T=03:00 Terminate. The Aruba RA shall issue a terminateReq() primitive to completely cancel the reservation.
    5. The Zombie Timeout should occur generating an NSI Event and a Connection transition. This terminates the test.

Results:

The NSAs are to print/log following information:

1. All NSI protocol messages that are sent or received. All information elements of those messages should be included in the log entry/printout.,
2. All other NSI events that occur.
3. All state changes of all ConnectionIDs. Note: this is the state of the Connection – not the protocol state machine. The ConnectionID, the triggering event, and both the old and new state should be logged.
4. A summary of the actions of the NRM should also be logged. This should include enough information to show that the NRM actions correspond to the state of the Connection.
5. Each log entry should include a timestamp with at least millisecond accuracy.
6. Each NSA should have the ability to also dump the WS Message Transport Layer if necessary or requested to resolve messaging issues at the NSI protocol layer.

**Challenge 3. Inter-domain Path Segmentation**

Objective:

The purpose of Challenge #3 is to show that the NSAs are able to segment a reservation request and construct a suitable service tree that meets the request criteria.

Description:

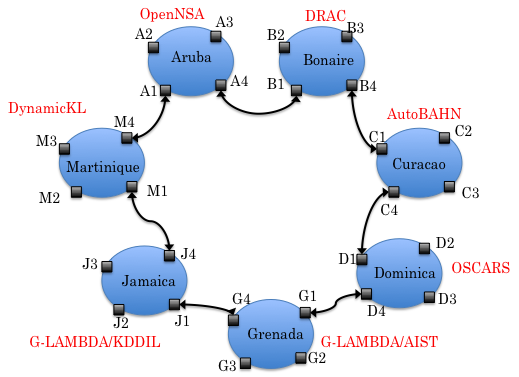
In this Challenge, the objective is to show that an NSA is able to receive a reservation request, construct a candidate inter-domain path, issue the children reservation requests, and upon confirming the children segments return a reservation confirmation to the RA.

In challenge #3, the NSA under test is the initial PA. For instance, NSA Aruba initiates a reservation request to NSA Bonaire for a connection from Bonaire:B2 to Dominica:D2. Bonaire is the NSA under test. By merit of the ring topology, Bonaire will need to segment the request to create a) a local segment, and b) and inter-domain segment(s). The inter domain segmentation may use either tree or chain. The resulting messaging between NSAs will indicate which segmentation approach was used.

A second reservation request will be issued from Aruba to Dominica. The request will go from Dominica:D3 to Bonaire:B3. This second request will be issued so as to overlap in the scheduled time with the initial reservation thus forcing an alternative path selection.

Topology:

The Rio Ring topology for Challenge #3 is diagramed below:



There will be two reservation requests issued in an overlapping timing sequence to demonstrate the PA’s capability for path selection and segmentation.

**Test Scenario 3.1: Simple segmentation**

This exercise will begin with the cold start initialization of all NSAs. Once all NSAs have completed initialization processing, the exercise can begin.

Proceedure:

* + 1. T=00:00. Reserve. Maui issues reserveReq() message to Aruba. The reservation will include the following parameters:
* Source STP: Aruba:Ashley
* Destination STP: Curacao:Calista
* Start Time: T=01:00
* End Time: T=04:00 (3 minute duration)
* Capacity: 1000 Mbps
* Directionality: Bidirectional
* Service Authorization: “Shaman”
* Global Reservation ID: null

The Aruba PA should be observed issuing reservationReq() messages for the children segments. and a signal to initiate reserve resources to its own NRM Subsequent reserveResp messages should be received by Aruba from the children. A reserveResp=Confirmed should be sent from Aruba to Maui.

Note: The decision to us tree or chain segmentation is left to the Aruba NSA as a local internal policy decision. Either is acceptable and will be evident in the children messages.

* + 1. T=01:00 Start Time. The NSI Start Time arrives, generating an NSI Start Event.
    2. T=01:15 Provision. Upon reaching the Start Time+00:15, the Maui RA shall send a provisionReq() primitive to the Aruba PA. Aruba should subsequently be observed to issue provisionReq() messages to its children segments and a signal to initiate provisioning to its own NRM. Aruba should then be observed to receive provisionResp messages from those children and a similar indication from its NRM. Upon receiving completion status from all children and the NRM, Arube should be observed to send a provisionResp() message to Maui.
    3. T=02:00 Release. At 2:00 minutes, the connection should be “InService” and in a quiescent state. The Maui RA will issue a releaseReq() primitive to the Aruba PA. Aruba should immediately issue similar messages to its children and its NRM. Corresponding responses should be observed arriving at Aruba from the children. A releaseResp() message should be sent from Aruba to Maui.
    4. T=03:00 Terminate. Maui issues a terminateReq() primitive to completely cancel the reservation. Aruba should be observed issuing similar messages to its children and NRM.
    5. The zombie ConnectionID should timeout causing an NSI event to be logged/dumped. This concludes the test.

Results:

For Test Scenario 3.1, only the Aruba NSA is required to dump its NSI messaging and events. The results to be displayed include:

1. All NSI protocol messages that are sent or received. All information elements of those messages should be included in the log entry/printout.,
2. All other NSI events that occur.
3. All state changes of all ConnectionIDs. Note: this is the state of the Connection – not the protocol state machine. The ConnectionID, the triggering event, and both the old and new state should be logged.
4. The interactions with the NRM should also be dumped. These should include enough NRM interaction information to show that the NRM actions are consistent with the primitive.
5. Each log entry should include a timestamp with at least millisecond accuracy.

**Challenge 4. Query request.**

Objective:

The purpose of Challenge #4 is to show the proper interpretation and handling of the Query() primitive.

Description:

In this Challenge, the objective is to show that an NSA is able to process two types of Query() requests: a “status” query, and a “path”query. A key aspect of the Query request processing is that the information returned is properly authorized. Since the Rio Plugfest is focused on basic interoperability rather than strict conformance testing, this Challenge will likewise ease the authorization requirements to use the same “shaman” omnipotent authorization credentials for all query requests. For Rio, authorization will consist of a simapl comprehensive policy of “Is this user the Shaman? If so, allow, else deny.”

The Status Query() returns information about the user’s reservation *request*. Status information includes all the confirmed service attributes (performance, schedule, etc.) that are part of the service definition and that were returned to the RA when the reservation was authorized and confirmed. The current State of the ConnecitonID is also returned.

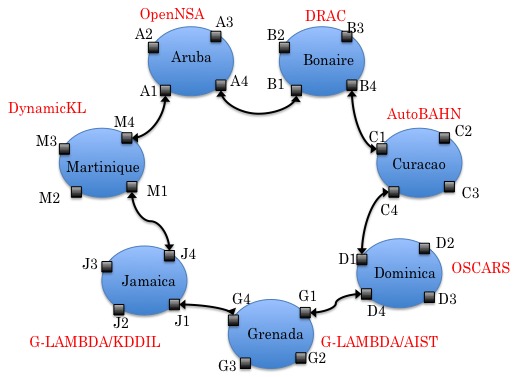
The Path Query() returns information that was chosen by the PA based upon PA internal policy. Because this information was based upon internal policies and requirements of the PA, the PA “owns” this information, and therefore access to this information is likewise subject to PA authorization policies – i.e. it is not required of the PA to authorize internal Path information under the same policy as the Status information. Where as the NSI CS standard requires that the Status Query() return certain information under the same credentials as requested the reservation, the Path information is authorized under PA policy which may (or may not) require additional privilege.

In Challenge 4 a non-trivial reservation request will be established, and various Query() requests will be submitted throughout the life cycle. Query “status” with the ConnectionID criteria will be preformed, a Query “path”, and a Query status using the Global Reservation ID field as criteria will be performed. The messaging information will be displayed to evaluate the handling of these NSI protocol messages.

For the Rio Plugfest, the authorization issues have been simplefied as these are not strictly NSI interoperability issues. So the same omnipotent “shaman” authorization credentials will be used for both requesting the reservation and querying the connection.

Topology:

The topology for Challenge #4 is identical to Challenge #3 and is diagramed below:



~~The ultimate RA will be NSA “Maui” and will initiate the reservation requests. Aruba will act as the initial PA. Curacao, being the far end terminus of the connection, will be queried using the Global ConnectionID.~~

**Test Scenario 4.1: Queries**

This exercise will begin with the cold start initialization of all NSAs. Once all NSAs have completed initialization processing, the exercise can begin.

Proceedure:

* + 1. T=00:00. Reserve. Maui issues reserveReq() message to Aruba. The reservation will include the following parameters:
* Source STP: Aruba:Ashley
* Destination STP: Curacao:Calista
* Start Time: T=02:00
* End Time: T=03:00 (1 minute duration)
* Capacity: 1000 Mbps
* Service Authorization: “Shaman”
* Global Connection ID: “GCID-Maui-Rio01”

Aruba should be observed to issue children reservation requests to other NSAs. Upon completing segment reservations, a reserveResp=Confirmed should be sent from the Aruba PA to the Maui RA.

* + 1. T=00:30 Query(Status). After the reservation has been confirmed, a Query(Status) request shall be sent from Maui to Aruba. Aruba should return the Connection Status and the “as-built” information related to the request (capacity, schedule, and any other relevant service parameters.)
    2. Query(Path). Immediately after completing the 00:30 Query(status) primitive, Maui will issue a Query(Path) request to Aruba. This should result in a recursive query by Aruba, and the associated messaging should be observed between Aruba and the children NSAs. Aruba should aggregate the children responses and return the Path information to Maui.
    3. Query(Status) Immediately after completing the prior Queries, Maui will issue a Query(Status) using the Global Connection ID to *NSA Curacao*. This query will use the “Global ConnectionID=GCID-Maui-Rio01” supplied in the reservation request to query the far end of the connection. Curacao should recognize the “shaman” authority and provide the Status info back to Maui.
    4. Query(Path) Immediately following the previous Query response, the Maui RA will issue a Query(Path) request to the Aruba. This request should specify non-existent Authorization credentials. The Aruba PA should reject the request.
    5. T=01:00 Provision. Maui RA shall send a provisionReq() primitive to the Aruba PA. Aruba should be observed to issue provisionReq() messages to its children segments.
    6. T= 01:15; 01:30;…03:00; 3:15; 3:30. Query(Status) Having issued the provisionReq() message putting the reservation into Auto-start mode, the Maui RA shall commence sending a Query(Status) request to Aruba approximately every 15 seconds for the next three minutes as the reservation auto-sequences through its lifecycle.
    7. T=02:00 Auto-Start At 2:00 minutes, the connection should auto-start. The Query(Status) requests should show the reservation status change from Provisioning to InService.
    8. T=03:00 Auto-Terminate. The reservation should reach its End Time and auto-terminate. The repeating Query(status) requests should show the reservation transitioning from In-Service to Terminated and ultimately disappear (the zombie timeout in Aruba should be set to 30 seconds or less for these tests.)
    9. The zombie ConnectionID should timeout causing an NSI event to be logged/dumped. This concludes the test.

Results:

For Test Scenario 4.1, both the Maui NSA and the Aruba NSA are required to dump its NSI messaging and events. The results to be displayed include:

1. All NSI protocol messages that are sent or received. All information elements of those messages should be included in the log entry/printout. The display or results should include recursive descent into the return Path results of the children.
2. All NSI Events should be logged and displayed. These include Start and End Time events, Response Timeouts, and Zombie Timeout.
3. Each message/event log entry should include a timestamp with at least millisecond accuracy.