NSI Error Handling

# Error Handling Architecture Overview

The NSI architecture is a distributed, multi-domain, multi-agent architecture. It is very important to deal with error cases to ensure predictable and deterministic state and user-experience of the network service. This section will describe the principles that will provide the basis of error handling and recovery for the design of the protocol.

Errors can happen in the network due to the following reasons:

1. Soft Failures

The NSA agents use the IP network to communicate with each other using the NSI protocol. Soft failures occur when the two NSA agents lose communication with each other or a NSA/NRM agent loses connection with the management plane of the transport network.

1. Hard Failures

These failures occur when the NSA software crashes due to a bug or the underlying server hardware fails. These kinds of failures may cause a loss of state in a message sequence.

Going hand-in-hand with errors is recovery. The NSI protocol and the NSA architecture need to be able to recover from errors to a consistent and predictable state in an NSA. The following architectural principles should guide the error handling and recovery in the NSI protocol:

1. Handling of failures should result in ***deterministic*** behaviors that are user centric and oriented towards the service model, e.g.
   1. Failure in the service plane (i.e NSA) should not affect connections that are provisioned and active in the transport plane.
   2. Failure in the service or transport plane should not result in a partially functioning service.
2. Recovery of transport and/or service plane **should not** be reliant on external entities or mechanisms, e.g.
   1. NSA should not be dependent on other NSAs to recover from (i.e. route around) a local transport plane failure (under the control of it’s NRM).
   2. NSA recovering from a hard failure error condition cannot depend on peer NSAs to reconstruct it’s state.

It is important to note that explicit user negotiations may change the above-mentioned behaviors. For example, a user may request that if any of the NSAs in a service tree fails, all the NSAs in the service tree should tear down the connection service in the transport plane or the user may negotiate a later starting time for a circuit due to the re-path computation and reservation delay. We will now discuss specific failure handling scenarios.

# Transport Plane Error Handling

Failures in the transport plane can occur at anytime, however within the framework of the NSI architecture, there are two time windows in which a transport plane failure is significant (see Figure 1.);

1. The time between the service reservation phase and provisioning phase (i.e. TReservationCompleted to TProvisionStart), and
2. The time between the service provisioning phase and teardown phase (i.e TProvisionCompleted to TTeardownStart).

Of course, the errors only need to be handled by the NSA if the transport resource errors affect the user service.

Figure 1. Transport Plane Failure Sensitive Sections

A few illustrative examples will help describe the kind of failure and recovery scenarios that have to be considered when building the state machine for the NSI protocol.

1. Transport failure during service reservation phase and provisioning phase : An element in the transport plane becomes unavailable due to a soft or hard failure causing a provisioning failure of a confirmed reservation, The reservation manager can handle this by either reserving an alternate path as long as it meets the requested service characteristics or canceling the reservation with notification. Domain policy and availability of resources will determine what recovery action is taken by that domain.
2. Transport failure during provisioning phase and teardown phase: In case a failure in the transport plane affects an active connection requested in the service plane, 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 domain or tearing down the service with notifications to other domains involved.

# Service Plane Error Handling

Failures in the service plane can result in inconsistent states across the various NSAs. Examples of service plane errors can be losing communication with a certain domain’s NSA, losing communication with the transport network, corruption/crash in the platform etc. These errors may result in service disruptions until these states can be synchronized, hence the NSI protocol and state machine design should account for such scenarios.

Unlike failures in the transport plane, service plane failures that interrupt an NSA workflow sequence (i.e. service action) can be problematic. This is especially true during the following service actions (see /figure 2);

1. Reservation,
2. Provisioning,
3. Teardown, and
4. Release .

Figure 2. Service Plane Failure Sensitive Sections

Within the context of an NSA which has access to both local and remote resources, the service plane errors can be in the remote resources, or within the local resources. Since a reliable transport is assumed for NSA interactions, failure in communication with a remote PA or RA can be assumed to be a remote NSA failure. The following diagram illustrates the two perspectives.

Regardless of where the error is, it is imperative that the complex NSA recover to a deterministic state from both the service state perspective that the user experiences and from a resource state perspective required for it to be in sync with other NSA’s.

Figure 3. RA/PA Local/Remote Failures

# State Machine and illustrative failure scenario

The distributed model of servicing user requests using tree/chain model projects that each domain, can assume the role of a “Requestor Agent” or “Provider Agent”. When service plane failures occur, it is quite possible that an entire domain will become disconnected from the service tree i.e. the failure will be in both the RA and PA of that domain. This scenario imposes a requirement on the NSA to have a linkage between its RA and PA state machines to understand the impact of the failure on the service tree and recover from it.

The following example error scenario can illustrate this complexity –

Imagine there are three domains in a chain topology,

A🡨🡪B🡨🡪C

with the connection service flow going from A 🡪 C. This implies the following roles will be interacting to complete the flow:

RA[A]🡨🡪PA[B]RA[B]🡨🡪PA[C]

If NSA for domain B loses IP connectivity, it loses service plane connectivity with A and C, both PA[B] and RA[B] will lose connectivity with their peers at the same time.

When RA[A] loses connectivity to PA[B] during one of the critical phases described above, lets pick Reservation phase, it has to assume that PA[B] was not able to reserve resources in its own domain and “may” have communicated the request to domain C. There are only two action options left for RA[A] – to re-do the path calculation to meet the service request that marks domain B as unreachable or to notify the user of a reservation failure. In either case, RA[A] will return to a deterministic state of “Resources Free” before proceeding with reservation of a new path, using the terminology of Figure 3.

If domain C had already received a setup request and reserved its local resources, it will try to communicate the confirmation to RA[B]. If domain B does not respond, it will free the reserved resources and go back to “Resources Free” state. In this case, if domain A found another way of reaching domain C, those resources will be available for the same connection service, maybe with different service end-points.

In a similar manner, both PA[B] and RA[B] will discover lost connectivity with their peers during this reservation setup process. They will assume that RA[A] and PA[C] are unavailable and in error condition, so it will release its own resources and come to a “Resource Free” state. Since the RA[B] got the service request from PA[B], RA[B] will notify PA[B] for this condition, so the service tree state and dependency can be cleared by PA[B].

This example illustration above demonstrates on how the state machine of RA and PA should be designed so the outcome of a distributed failure ends each state machine in a deterministic state. These are the architectural principles that will be utilized for the design of the NSI state machines and the protocol interactions during error conditions.