GENI Network Stitching Component Design - Example

Table of Contents

GENI Stitching Architecture - Component Design

1 Introduction

This document presents a proposal for the GENI Network Stitching Architecture Component design details. These architecture components defined in the GENI Network Stitching Architecture document are as follows:

- Stitching Resource Element
- Stitching Topology Service
- Stitching Path Computation Function
- Stitching Workflow Function
- AM API Network Stitching Extensions
- Common Stitching Topology Schema

The purpose of the document is to start conversations towards consensus and formal standardization of the GENI Network Stitching Architecture and associated component details. Toward this end, this document presents a design proposal for the following:

- Stitching Resource Element Definition
- Stitching Topology Service API
- Stitching Path Computation API
- Stitching Workflow API
- AM API Network Stitching Extensions
- Common Stitching Topology Schema

The designs proposed here focus on the information model and API definitions. After a consensus for this level of information is obtained, additional details on function logic, message processing, and other design details will be required.

2 Stitching Resource Element Information

The Stitching Resource Element would be provided by all aggregates as a mechanism to describe their connections to other aggregates and external networks. There is a need for this resource description to be provided in accordance with a "Common Stitching Topology Schema" so that multi-aggregate stitching operations can be planned and instantiated. This element will be in addition to and coexist with the various aggregate-specific RSPECs. For this proposal we are suggesting that the Stitching Resource Component may be optionally provided along with aggregate specific RSPECs via the various AM API messages. Since the Stitching Resource Element, formatted in accordance with the "Common Stitching Topology Schema" is really a standalone element, it can also be referred to a as "Stitching RSPEC".

The practical implication of this is that a single aggregate sliver may have associated with it both an aggregate specific RSPEC and a Stitching RSPEC. It is also possible for the user/host resources, described via an aggregate specific RSPEC, to be one sliver, and the stitching resources, described via the Stitching RSPEC, to be another sliver. Both of these slivers can then be associated with a single slice. These are largely individual aggregate implementation and policy decisions and the designs presented in this document are intended to support both.

In summary, we are saying there are two options for how the stitching resources are associated with a Sliver:
i) The user/host resources and stitching resources are all combined into a single Sliver.
ii) The user/host resources are one Sliver, the stitching resources are another Sliver, and both of these Slivers are associated with a single Slice

In all scenarios, the stitching resources will be described and defined by a Stitching RSPEC, and individual aggregates will be responsible for creating associations with the elements in their aggregate specific RSPECs.

The term "Stitching RSPEC" will be utilized in the remainder of this document to refer to the Stitching Resource Element.

The specific "Common Stitching Topology Schema" utilized for the Stitching RSPEC is less important then the information content. Therefore this section focuses only on the information content. A candidate topology schema is presented in Section 7.

Below is a high level description of some information elements proposed for the Stitching RSPEC:

- Aggregate Level Information
  - Aggregate Identifier
  - Aggregate Manager URL
  - Last Update Time
  - Aggregate Type (#planetlab | protogeni | openflow | others)
  - Stitching Mode (#chain | tree)
  - Aggregate Capabilities
    - Scheduled Services (#will accept future start times)
    - Negotiated Services (#will hold some resources temporarily)
- Node Level Information
  - Node Identifier
  - Port Identifier
    - Node Type
  - Link Identifier
    - Link Capabilities
      - Technology Type
      - Bandwidth Available (#for potential use)
      - VLAN Ranges available (#for potential use)
      - VLAN Translation Capabilities
      - Remote Connection Link Identifier (#points to remote peering link)

This is only a subset of the information that would actually be desired in a full Common Stitching Topology Schema specification. The information presented above is intended to describe the basic information that each aggregate would need to provide in order for a Topology Service to build a view of the interconnected GENI Aggregate space. The key information element for this, is the Link Identifier and the Remote Connection Link Identifier combination. The Remote Connection Link Identifier will point to a link in another aggregate (or external network) and will contain enough information to enable retrieval of that aggregate's (or external network's) Stitching RSPEC. This is the mechanism by which a global interconnected aggregate space can be constructed.

A more detailed description of the information need in the Stitching RSPEC and how it is utilized as part of the larger multi-aggregate stitching operations is provided in the GENI Network Stitching Architecture and GENI Network Stitching Architecture - Example documents. In addition, the candidate "Common Stitching Topology Schema" specification described in Section X also presents a more complete listing of the information elements.

3 Aggregate Manager API Extension for Stitching

In accordance with the GENI Network Stitching Architecture-Overview [1] and GENI Network Stitching Architectur-Example [2] documents we propose the following extension to the GENI-SE-CF-AMAPI-01.0 document [3].

Modifications to ListResources (indicated in red):

7.3 ListResources

Return information about available resources, or return resources allocated to a slice.

```
string ListResources(string credentials[], struct options)
```

This operation is similar to ProtoGENI's DiscoverResources operation and to the SFA's GetResources operation (sec. 6.2.4).

7.3.1 Arguments

`credentials[]`

An array of credentials. At least one credential must be valid for this operation (signed by a valid GENI certificate authority either directly or by chain, and not expired).

`options`

An XML-RPC struct containing members indicating the set of resources the caller is interested in or the format of the result. In addition to the members specified below, callers can pass additional members that specific aggregate manager implementations might honor. The prefix 'geni_' is reserved for members that are part of this API specification. Implementations should choose an appropriate prefix to avoid conflicts.

The following members are available for use in the options parameter. All aggregate managers are required to implement these options.

```
{
 boolean geni_available;
 boolean geni_compressed;
 string  geni_slice_urn;
 string geni_stitching;
```

```
            }
```

`geni_available`

An XML-RPC boolean value indicating whether the caller is interested in all resources or available resources. If this value is true, the result should contain only available resources. If this value is false both available and allocated resources should be returned. The Aggregate Manager is free to limit visibility of certain resources based on the credentials parameter.

`geni_compressed`

An XML-RPC boolean value indicating whether the caller would like the result to be compressed. If the value is true, the returned resource list will be compressed according to RFC 1950.

`geni_slice_urn`

An XML-RPC string indicating that the caller is interested in the set of resources allocated to the slice named by this URN. If no resources are allocated to the indicated slice by this aggregate, an empty RSPEC should be returned.

`geni_stitching`

An XML-RPC string value indicating whether the caller is interested in the Stitching RSPEC. If this value is "true", the Stitching RSPEC will be returned along with the other aggregate specific RSPEC. If this value is "false", the Stitching RSPEC will not be returned. If this value is "only", then only the Stitching RSPEC will be returned, and not the aggregate specific RSPEC. The Stitching RSPEC is formatted in accordance with the "Common Stitching Topology Schema" and will be appended to the end of any aggregate specific RSPECs that may be returned. This option may be used together with the geni_available, geni_compressed, and/or geni_slice_urn options. Note that geni_available value 'true' alone does not include the Stitching RSPEC in the returned string.

7.3.2 Return value

The return value is always a string. If *geni_compressed* is unspecified or set to false the return value will be an advertisement RSPEC in text format. If *geni_compressed* is specified and set to true the return value will be a string whose contents are base 64 encoded [RFC 1421] compressed advertisement RSPEC. Clients must first base 64 decode the string, then uncompress the decoded result.

From a network stitching perspective, the main caller of the ListResources is expected to be the GENI Stitching Topology Service. This will be utilized to so that the Topology Service can build a topology view via periodically polling for the Stitching RSPECs from all participating aggregates. Clients may also use this extension to expand their own topology view, in which case they may set both *geni_available* and *geni_stitching* to true to get both resource descriptions in one operation call.

It should be noted that when the geni_slice_urn string is provided, and the geni_stitching option is set to true, an aggregate specific RSPEC and a Stitching RSPEC may both be returned. In this scenario, the resources identified by the aggregate specific RSPEC and the Stitching RSPEC are considered to be part of a single Sliver, and associated with the indicated geni_slice_urn. It is also possible for a geni_slice_urn to have an associated RSPEC which contains only a Stitching RSPEC. This will be typical for an aggregate which is providing transit services for the stitching together of other aggregates, or for an aggregate who wishes to manage the user/host and stitching resources as two separate Slivers associated with the same Slice. In this context a Stitching RSPEC alone, may be referred to as a "Stitching RSPEC". It is also assumed that the aggregates will develop mechanisms to correlate elements in their aggregate specific RSPEC to those in the Stitching RSPEC.

In summary, we are saying there are two options for how the stitching resources are associated with a Sliver:
i) The user/host resources and stitching resources are all combined into a single Sliver.
ii) The user/host resources are one Sliver, the stitching resources are another Sliver, and both of these Slivers are associated with a single Slice

In all scenarios, the stitching resources will be described and defined by a Stitching RSPEC, and individual aggregates will be responsible for creating associations with the elements in their aggregate specific RSPECs.

Addition of CreateStitchingSliver:

7.9 CreateStitchingSliver

Allocate stitching resources as a slice sliver on the current aggregate. This operation can either start negotiation of the resource allocation or finalize the allocation. This operation is expected to start the allocated resources asynchronously after the finalizing operation has successfully completed. Callers can check status of the stitching resources using StitchingSliverStatus.

```
 string CreateStitchingSliver(string slice_urn, string
credentials[], string rspec, struct users[])
```

7.9.1 Arguments

`slice_urn`

The URN of the slice to which the resources specified in the *rspec* argument will be allocated.

`credentials[]`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*. Aggregates should ensure that the expiration time of the slice does not exceed the expiration time of the slice credential used to perform this operation.

`rspec`

A request RSPEC (plain text, uncompressed) containing the resources that the caller is requesting for allocation to the slice specified in *slice_urn*. The RSPEC is formatted with the “Common Stitching Topology Schema.”

`users`

An array of user structs, which contain information about the users that might login to the sliver that the AM needs to know about. In the case external network connection is required, the array may contain only one user who has the privilege to setup the connection via a proxy.

`options`

An XML-RPC struct containing members that specify how the stitching resource negotiation and allocation should be performed.

The following members are available for use in the options parameter. All aggregate managers are required to implement these options.

```
{
  boolean geni_stitching_negotiable;
  boolean geni_stitching_autostart;
  boolean geni_stitching_autorenew;
  boolean geni_stitching_autodelete;
}
```

`geni_stitching_negotiable`
An XML-RPC boolean value indicating whether the called aggregate manager is allowed to return a different RSPEC for negotiation.

`geni_stitching_autostart`
An XML-RPC boolean value indicating whether to start the allocated stitching resources automatically without a separate StartSliver call.

`geni_stitching_autorenew`
An XML-RPC boolean value indicating whether to renew the allocated stitching resources automatically without a separate RenewSliver call.

`geni_stitching_autodelete`
An XML-RPC boolean value indicating whether to delete the allocated stitching resources automatically without a separate DeleteSliver call.

7.9.2 Return value

The return value is a manifest RSPEC indicating the stitching resources that were allocated to the sliver. The result RSPEC may contain additional information about the allocated resources. The RSPEC is formatted with the "Common Stitching Topology Schema."

The CreateStitchingSliver call sends a Stitching RSPEC to the AM to request allocation of stitching resources. The Stitching RSPEC (formatted in accordance with the Common Stitching Topology Schema) will contain the details regarding the specific stitching request. These options may include items which require multi-aggregate stitching coordination. An example of this would be a Stitching RSPEC which indicates a request for stitching together two aggregates via an Ethernet VLAN. This may be reflected in the Stitching RSPEC by the identification of the two aggregates to be stitched, and a specification of which VLAN tag to use in the form an "any vlan tag" request. Use of the "any vlan tag" feature will be necessary to enable multi-aggregate stitching coordination functions to identify a specific VLAN tag which will work across all the aggregates involved in the specific stitching operation. This multi-aggregate coordination may be accomplished via direct aggregate to aggregate communications (chain mode) or via a stitching workflow function which communicates directly with multiple aggregates (tree mode).

The option *geni_stitching_negotiable* provides a feature useful for the multi-aggregate coordination. If the *geni_stitching_negotiable* is set to '*true*' the AM is expected to process the RSPEC in accordance with "Multi-Aggregate Coordination" processing. This processing is closely aligned with the information contained in the stitching RSPEC (and associated Common

Topology Schema). Section 7 includes a discussion of the "Multi-Aggregate Coordination" in the context of the example schemas presented.

The other three options in the CreateStitchingSliver call are *geni_stitching_autostart*, *geni_stitching_autorenew* and *geni_stitching_autodelete*. When the option *geni_stitching_autostart* is set to true, the AM will start the allocated stitching resources automatically or along with the call of StartSliver (not defined in AM API 1.0). Similarly, setting true values for *geni_stitching_autorenew* and *geni_stitching_autodelete* will automatically renew and delete the stitching resources when RenewSliver and DeleteSliver are called (already defined in AM API 1.0). At this point, we are not proposing separate StitchingSliverStart, StitchingSliverRenew and StitchingSliverDelete operations into the GENI AM API extension. All the three options, *geni_stitching_autostart*, *geni_stitching_autorenew* and *geni_stitching_autodelete,* should have default value *'true'*.

Addition of StitchingSliverStatus:

7.10 StitchingSliverStatus

Get the status of stitching resource sliver from *negotiating* to reserved to *configuring* to *ready* or *failed*.

```
string StitchingSliverStatus(string sliceorsliver_urn, string
    credentials[])
```

7.10.1 Arguments

`sliceorsliver_urn`

The URN of the slice for which the stitching resource sliver is requested.

`credentials[]`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*.

7.10.2 Return value

The return value is a manifest RSPEC indicating the stitching resources that were allocated to the slice. Status of the resources is indicated by the "status" attribute embedded in topology description in the RSPEC. The RSPEC is formatted with the "Common Stitching Topology Schema."

Unlike the SliverStatus operation that returns a status struct for the whole sliver, the StitchingSliverStatus operation returns status for the Stitching RSPEC only. Status of individual stitching resources will be available in the returned Stitching RSPEC. This will be needed for the stitching workflow, so that it can request stitching services and then check back later on status. It would be possible to include some stitching specific options in the current SliverStatus call as another method to provide this functionality.

4 Stitching Path Computation API

The Stitching Path Computation API may reside as a standalone service or embedded in another GENI component. In this proposal we are not defining the detailed path computation implementation. Here we define only a client facing API, which contains a

ComputeSliceStitching operation. This operation should support Point-to-Point (P2P), Point-to-MultiPoint (P2MP) and MultiPoint-to-MultiPoint (MP2MP) path computation. The GENI Network Stitching Architecture and GENI Network Stitching Architecture - Example documents do provide examples of how this API would be utilized in a larger stitching operation.

ComputeSliceStitching

Compute path of stitching resources that connect aggregates in a slice under network capacity and capability constraints. This operation is expected to return either an RSPEC with success or a failure status in a synchronous fashion.

```
    string ComputeSliceStitching(string slice_urn, string
        credentials[], string rspec, struct constraints[])
```

Arguments

`slice_urn`

The URN of the slice to which the resources specified in the *rspec* argument will be allocated.

`credentials[]`

An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*. The Stitching Path Computation Service should ensure that the expiration time of the slice does not exceed the expiration time of the slice credential used to perform this operation.

`rspec`

A request RSPEC (plain text, uncompressed) describing a P2P, P2MP or MP2MP path that will be used to stitch the slice specified in *slice_urn*. The RSPEC is formatted with the "Common Stitching Topology Schema.". The RSPEC will define fully the request from the client including desired constraints and options.

`options`

An XML-RPC struct containing members that defines options relating to the computation.

The following members are available for use in the constraints parameter.

```
    {
     date/time holdtime;
     boolean multiple_results;
    }
```

`holdtime`

An XML-RPC dateTime indicating a date/time until the path computation service will hold the computed result to avoid contention from other requests.

`multiple_results`

An XML-RPC boolean value which indicates that multiple computation results are desired if possible. The ability of a specific path computation function to return multiple results, and if so how many, will be implementation specific. The intent of this option is to allow the client to indicate a preference for multiple results, and for the computation service to decide how many additional results to return.

Return value

The return value is a manifest RSPEC indicating a more detailed path that was computed for stitching the slice. The result RSPEC may contain additional information about resource allocation / setup instructions for individual network segments. The RSPEC is formatted with the "Common Stitching Topology Schema." The returned RSPEC will also contain failure information for computation results which were not successful.

5 Stitching Workflow API

The Stitching Workflow API may reside as a standalone Stitching Workflow Function or embedded in another GENI component. In this proposal we are not defining the detailed workflow and internal protocols. The GENI Network Stitching Architecture and GENI Network Stitching Architecture- - Example documents do provide examples of how this API would be utilized in a larger stitching operation. Here we define the simple client facing API, which contains two operations: CreateSliceStitching and SliceStichingStatus.

CreateSliceStitching
Allocate stitching resources to a slice. This operation can either start negotiation of the resource allocation or finalize the allocation. This operation is expected to start the allocated resources asynchronously after the finalizing operation has successfully completed. Callers can check status of the stitching resources using SliceStitchingStatus.

```
string CreateSliceStitching(string slice_urn, string
    credentials[], string rspec, struct users[])
```

This operation is similar to the CreateStitchingSliver AM API call but applied to an entire slice.

Arguments

`slice_urn`
The URN of the slice to which the resources specified in the *rspec* argument will be allocated.

`credentials[]`
An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*. The Stitching Workflow Service should ensure that the expiration time of the slice does not exceed the expiration time of the slice credential used to perform this operation.

`rspec`
A request RSPEC (plain text, uncompressed) containing the resources that the caller is requesting for allocation to the slice specified in *slice_urn*. The RSPEC is formatted with the "Common Stitching Topology Schema."

`users`
An array of user structs, which contain information about the users of the requested slice.

`options`
An XML-RPC struct containing members how the stitching resource negotiation and allocation should be performed.

The following members are available for use in the options parameter.

```
{
    boolean geni_stitching_negotiable;
}
```

`geni_stitching_negotiable`
An XML-RPC boolean value indicating whether the called workflow controller is allowed to return a different RSPEC for negotiation.

Return value
The return value is a manifest RSPEC indicating the stitching resources that were allocated to the slice. The RSPEC is formatted with the "Common Stitching Topology Schema."

The use of *geni_stitching_negotiatble* option in CreateSliceStitching is similar to that of CreateStitchingSliver AM API call, but here it is applied to the entire slice. The communication here is between a client or an aggregate and the GENI Stitching Function. In most cases, the clients will give a straightforward request not allowing for negotiation (*geni_stitching_negotiatble = 'false'*).

The CreateStitchingSlice call sends a Stitching RSPEC to the GENI Stitching Function to request allocation of stitching resources. The Stitching RSPEC (formatted in accordance with the Common Stitching Topology Schema) will contain the details regarding the specific stitching request. These options may include "strict" or "loose" requirements. The Stitching Workflow Function will utilize the services of the Stitching Path Computation Function to turn client requests into specific actions for specific aggregates. The Stitching Workflow function will then communicate with the aggregates via the AM API to accomplish the network stitching task. This may include sequencing aggregate operations and multi-aggregate coordination in order to identify a specific set of resources (such as VLANs) that will work across all the aggregates and external networks needed. This workflow processing is closely aligned with the information contained in the stitching RSPEC (and associated Common Topology Schema). This processing is described in the GENI Network Stitching Architecture - Example document.

SliceStitchingStatus
Get the status of stitching resources for a slice from *negotiating* to reserved to *configuring* to *ready* or *failed*.

```
string SliceStitchingStatus(string slice_urn, string
        credentials[])
```

This operation is similar to the StitchingSliverStatus AM API call but applied to an entire slice.

Arguments
`slice_urn`
The URN of the slice for which the stitching resources are requested.

`credentials[]`
An array of credentials. At least one credential must be a valid slice credential for the slice specified in *slice_urn*.

Return value

The return value is a manifest RSPEC indicating the stitching resources that were allocated to the slice. Status of the resources is indicated by the "status" attribute embedded in topology description in the RSPEC. The RSPEC is formatted with the "Common Stitching Topology Schema."

Similar to the StitchingSliverStatus operation, the SliceStitchingStatus operation returns a complete slice RSPEC formatted by the common topology schema. Status of individual stitching resources can be obtained in the RSPEC.

6 Stitching Topology Service API

The Stitching Topology Service API may reside as a standalone Stitching Workflow Function or embedded in another GENI component. The API consists of two operations, GetStitchingTopology and RegisterStitchingTopology. Callers of the former operation include client and Stitching Path Computation service. Callers of the latter are individual Aggregate Managers.

In this proposal we are not defining the detailed design or internal processing. The GENI Network Stitching Architecture and GENI Network Stitching Architecture- - Example documents do provide examples of how this API would be utilized in a larger stitching operation. Here we define the simple client facing API.

GetStitchingTopology

Return information about available stitching resources for entire network for a list of aggregates.

```
string GetStitchingTopology(string credentials[],
     struct options)
```

Arguments

`credentials[]`

An array of credentials. At least one credential must be valid for this operation (signed by a valid GENI certificate authority either directly or by chain, and not expired).

`options`

An XML-RPC struct containing members indicating the set of resources the caller is interested in.

The following members are available for use in the options parameter. All aggregate managers are required to implement these options.

```
{
boolean geni_sts_all;
string geni_sts_level;
string geni_sts_aggregates[];
}
```

`geni_sts_all`

An XML-RPC boolean value indicating whether the caller is interested in topology of all the aggregates and external networks in the entire GENI network.

`geni_sts_level`

An XML-RPC string value indicating the level of details exposed in the topology. The values can be “aggregate_list” and “aggregate_topology” and “full_topology”, where:

-“aggregate_list” returns only the list of URNs of available aggregates and external networks.

-“aggregate_topology” returns the aggregate-level topology includes aggregate and external networks as abstract nodes and their interconnecting links.

-“full_topology” returns all the topology details learned by the Stitching Topology Service.

`geni_sts_aggregates[]`

An XML-RPC string array indicating the specific list of aggregates and external networks the caller is interested in. This option becomes invalid when geni_sts_all is ‘true’.

Return value

The return value is an RSPEC formatted by the “Common Stitching Topology Schema.” The RSPEC contains the requested network topology description. The RSPEC has a <lastUpdate> attribute to indicate when the topology has been last updated.

RegisterStitchingTopology

Send the local stitching resource component to register with the Stitching Topology Service.

```
string RegisterStitchingTopology (string credentials[],
     string rspec)
```

Arguments

`credentials[]`

An array of credentials. At least one credential must be valid for this operation (signed by a valid GENI certificate authority either directly or by chain, and not expired).

`rspec`

An XML-RPC string containing a Stitching RSPEC for the caller aggregate.

Return value

The return value is always an XML-RPC integer indicating *success* (1) or *failure* (0) status. In addition other failure codes may be defined to expand information returned.

The RegisterStitchingTopology operation is optional for the caller Aggregate Managers. The Stitching Topology Service can always poll the AMs to get their Stitching RSPEC. However, a “push” style registering operation may be desired by some AMs to accelerate topology updates so that local change can be more quickly reflected in the global stitching topology.

7 Common Stitching Topology Schema Definition

A common topology schema is needed so that each aggregate can describe their connections to other aggregates and external networks. There is a need for this resource description to be provided in accordance with a "Common Stitching Topology Schema" so that multi-aggregate stitching operations can be planned and instantiated. This element will be in addition to and coexist with the various aggregate-specific RSPECs. The term "Stitching RSPEC" is utilized interchangeably with the term Stitching Resource Element in this document.

The Stitching Topology Service will collect the common schema based resource descriptions to form an aggregate-level global topology view. The same schema can also be used by various stitching API functions to request, negotiate and set up stitching resources for a specific slice. Such a schema can be designed from scratch or extended from existing works such as NML and NDL. For GENI stitching, we need the schema to be able to describe the following:

1) connected topology graph
2) peer reference, i.e., interface of one aggregate can describe peering relationship with interface of another aggregate or an external network.
3) resource capacity, such as link bandwidth and VLAN tag range
4) resource capability, such as interface switching type and VLAN translation
5) multiple capabilities on same interface. For example, an interface can support both layer-2 and layer-3 connectivity.
6) negotiation parameters
7) resource allocation status

There are multiple possible inputs and starting points which can be identified for this common topology schema. Here are some:

- ProtoGENI RSpec version 2

- OGF NML (Open Grid Forum Network Markup Language) Working Group
  An official schema has not been published by this group. Additional information would be needed from the participants of that group to determine status and develop a version suitable for GENI Stitching work. Additional OGF NML information is available as identified in reference [4].

- NDL (Network Description Language)
  This is a schema as described in [5]. It is based on RDF (Resource Description Framework)

- InterDomain Controller Protocol (IDCP) based.
  This uses a schema as described in [6]. It is based on an OGF NM (Network Measurement) Working Group XML schema with extensions for control plane operations.

- Information model based on the IETF GMPLS [7] standards. Information model here refers to the information that the GMPLS community developed to describe network element, technologies, and capabilities as it relates to dynamic provisioning.

There are multiple schemas that could be used and/or adapted to satisfy the GENI Stitching Requirements. It is not critical which one is selected. The critical item is that only one schema is selected for the GENI Common Stitching Topology Schema, and that all GENI Aggregates provide a description of their external connection points in accordance with the chosen schema.

The Common Topology Schema could also be included into aggregate native schemas as an externally referenced namespace. As an example the ProtoGENI RSPECv2 AnyExtension element (http://www.protogeni.net/resources/rspec/2/any-extension-schema.xsd). It would also be possible to take the information elements needed for stitching and integrate them more natively into an aggregate specific schema. This would only be recommended if a common GENI RSPEC schema is defined which all aggregates will use for all slice and sliver operations. Until then, the following is recommended:

- Plan on have a common topology schema specifically focused on the stitching task
- Each aggregate will need to create references from elements in their native schemas to elements in the common topology schema. This will not require duplicating all the information elements in the common topology schema into their native schema. Only those elements which specifically pertain to their advertised external connections will need to be associated and referenced.

An example schema provided in Section 7.1 below. Others are encouraged to provide candidate schemas based on other options as well.

7.1 GENI Common Stitching Topology Schema - Example

This example schema is based on the following:

- The information model for describing network nodes, ports, and links is based on work from the IETF CCAMP Working Group [7]. That information model has been adapted to the purpose and mission of GENI stitching. The key items here is that the information model has been used for inspiration, but not taken verbatim. In addition, only the information model is adapted, not the messaging formats, operational practices.

- The motivation for looking to the GMPLS information model, is because that model represents the work of a larger number of experts in the area, and is proven by large scale deployments and implementation. While the overall GENI objectives are different the then the general Internet objectives, the network stitching task can benefit from their work.

- The above approach is similar to what was used for the IDCP [6] development.

- In addition, many of the architecture approaches and functions presented in the architecture documents [1,2] are similar to the IETF Path Computation Element (PCE) [8] architecture concepts.

- The format of this schema is based on XML

The schema is organized into an Aggregate/Node/Port/Link hierarchy. The information contained in the schema is:

<StitchingRSpec>
<Aggregate>

```
Last Update Time
Aggregate Manager URL
<Aggregate Level Information>
        Aggregate Identifier
        Aggregate Type  (#planetlab | protogeni | openflow | others)
        Stitching Mode  (#chain | tree |  chainAndTree)
        Aggregate Capabilities
                Scheduled Services  (#will accept future start times)
                Negotiated Services (#will hold some resources temporarily)
</Aggregate Level Information>

<Node Level Information>
        Node Identifier
        node address
        <Port Level Information>
        Port Identifier
                capacity
                maximum reservable capacity
                minimum reservable capacity
                granularity
                <Link Level Information>
                Link Identifier
                        remote link id (#this points to another link which allows the construction
                                        of a connected network graph)
                        traffic engineering metric
                        capacity
                        maximum reservable capacity
                        minimum reservable capacity
                        granularity
                        switching capability descriptors
                                switching capability type
                                encoding type
                                switching capability specific info
                                        interface MTU
                                        vlanRangeAvailability
                                        suggestedVlanRange
                                        vlanTranslation
                </Link Level Information>
        </Port Level Information>
</Node Level Information>
</Aggregate>

<Path Level Information>
<hop>
hop type (loose | strict)
hop definition (domain:node:port:link)
```

next hop
</hop>
</Path Level Information>

</StitchingRSpec>

Below is an example of what a StitchingRSpec may look like as used for the purpose of entering data into the Stitching Topology Service. The format for resource URNs incorporates the aggregate/node/port/link hierarchy into the GENI URN format as follows:

- urn::publicid:IDN+aggregate+node+port+link

```
---------------Stitching RSPEC Example---------------
<StitchingRSpec>
<aggregate id=" geni.maxgogapop.net" url="https://geni.maxgigapop.net:8443"
lastUpdateTime="20110220:09:30:21"  type="planetlab" stitchingmode="tree">
<aggregateCapabilities>
   <scheduledServices>true</scheduledServices>
   <negotiatedServices>true</negotiatedServices>
</aggregateCapabilities>

<node id="urn:publicID:IDN+aggregate=geni.maxgigapop.net+node=CLPK">
        <address>140.173.2.232</address>
        <port   id="urn:publicID:IDN+aggregate=geni.maxgigapop.net+node=CLPK+port=1-2-2">
                <capacity>1000000000</genistitch:capacity>
                <maximumReservableCapacity>1000000000</maximumReservableCapacity>
                <minimumReservableCapacity>1000000</minimumReservableCapacity>
                <granularity>1000000</granularity>
                <link id="urn:publicID:IDN+aggregate=geni.maxgigapop.net+node=CLPK+port=1-2-
                2+link=*">
                        <remoteLinkId>
                        urn:publicID:IDN+aggregate=protogeni.net+node=bbg+port=0-3-1+link=*
                        </remoteLinkId>
                        <trafficEngineeringMetric>10</trafficEngineeringMetric>
                        <capacity>10000000000</capacity>
                        <maximumReservableCapacity>10000000000</maximumReservableCapacity>
                        <minimumReservableCapacity>1000000</minimumReservableCapacity>
                        <granularity>1000000</granularity>
                        <SwitchingCapabilityDescriptors>
                                <switchingcapType>l2sc</switchingcapType>
                                <encodingType>ethernet</encodingType>
                                <switchingCapabilitySpecificInfo>
                                        <interfaceMTU>9000</interfaceMTU>
                                        <vlanRangeAvailability>
                                                0,5-149,151-614,616-899,905-1001,1025-1999,2004-
                                                3965
                                        </vlanRangeAvailability>
                                        <vlanTranslation>true</vlanTranslation>
                                </switchingCapabilitySpecificInfo>
                        </SwitchingCapabilityDescriptors>
                </link>
        </port>
</node>
```

```
</aggregate>
</StitchingRSpec>
--------------Stitching RSPEC Example--------------
```

In this example, one external connection link is advertised by the MAX GENI aggregate stitching topology. This external connection is to the ProtoGENI network. The ProtoGENI Aggregate would be expected to advertise an external connection to the MAX Aggregate, with the linkId and remoteLinkId fields the reverse of what is shown above.

If there were more external connections, then the above would include additional node, port, and/or link elements as needed. The use of link technology and layer switching capability information as the mechanism to define the network capabilities is modeled after the IETF GMPLS standards [6].

An actual schema based on this approach is shown in Appendix A.

GENI Network Stitching across more then one aggregate will require multi-aggregate coordination processing. This can be accomplished via a tree, chain, or hybrid mode. The information elements in the common topology schema are utilized along with the GENI AM API Stitching extensions to enable this workflow processing. A detailed description for the messaging and state machine processing will be the subject of future work. However, a high level summary of how this processing may occur is presented below:

Tree Mode

- Stitching Workflow receives a Stitching Request
- Stitching Workflow either uses the slice (path) stitching information provided or utilizes the Path Computation Function to get another path option
- Stitching Workflow inspects the path information and determines the proper sequence for AM interactions. Sequencing decision will be determined via inspection of the Stitching RSPEC advertisements and identifying which aggregates can support negotiated services and which ones can support VLAN translation. Using that information, a workflow function would typically contact aggregates based on capabilities in the following order: i) aggregates which do not support negotiation or translation first, ii) aggregates which support negotiation but do not support VLAN translation second, iii) aggregates which support do not support negotiation but do support VLAN translation third, iv) aggregates which support both negotiation and translation fourth. This maximizes the chance of finding a set of VLANs that will allow stitching across all the aggregates.
- Stitching Workflow submits a CreateStitchingSliver request to AM A with a request for "any vlan" on its external connection link with geni_stitching_negotiable set to true
- AM A responds with a range of VLANs available and one suggested VLAN it will hold available for 10 minutes. This time can be variable and possibly renewable depending on design and implementation details.
- Workflow collects similar information from each the other AMs
- Stitching Workflow makes a decision about which VLAN to use on which AM, and submits a CreateStitchingSliver request to AM A with geni_stitching_negotiable set to false
- AM A works on instantiating the stitching resources

- Stitching Workflow continues similar interactions with other AMs.
- Stitching Workflow checks status on all AMs and coordinates actions for any failure conditions and prepares status in anticipation for client status check.
- This workflow processing is envisioned to be handled in an asynchronous manner, where the workflow manager submits requests to the AMs, and then polls later to check status.

Chain Mode

- AM A receives a CreateSliverStitching Request
- AM A either uses the slice (path) stitching information provided or utilizes the Path Computation Function to get another path option
- AM A inspects the path information and determines that this requires a multi-aggregate stitching chain, and that it is the first AM in the chain
- Chain mode processing requires that all the aggregates in the chain support the "chain workflow processing" which requires forwarding requests down the chain of aggregates. While not strictly required, chain mode aggregates should also support negotiation feature. The following steps assume negotiation is supported.
- AM A submits a CreateStitchingSliver request to AM B with a VLAN Range and a Suggested VLAN value with the geni_stitching_negotiable set to true
- AM B inspects the VLAN Range and a Suggested VLAN value and adjusts as necessary. The VLAN Range may be narrowed down by AM B, or increased if AM B can do VLAN Translation.
- AM B identifies the next AM in the chain, AM C, and sends a CreateStitchingSliver Request with the modified VLAN Range and Suggested VLAN with the geni_stitching_negotiable set to true.
- This forwarding down the chain, continues until the last AM in the chain is reached, AM C.
- AM C now has VLAN Range and Suggested VLAN which has been modified by all the AMs in the chain. AM C uses the Suggested VLAN if it can, if not it uses a random function to select another VLAN from the VLAN range
- AM C sends a CreateStitchingSliver request to AM B with geni_stitching_negotiable set to false
- AM B sends a CreateStitchingSliver request to AM A with geni_stitching_negotiable set to false
- It should also be noted that those AMs which can do VLAN translation may be adjusting the VLAN values for upstream peers. That is, the downstream AM always selects the final VLAN for the upstream peer. Downstream refers to the direction of the chain message flow. The first AM in the chain sends the messages in the "downstream" direction.

These chain mode processing may be done synchronously, where the AMs wait for the responses to propagate up and down the chain, or asynchronously, were the AMs poll each other for status.

Hybrid Mode Tree Mode with Chain Regions

- The mode would allow a tree mode processing to recognize there are regions of chain capable aggregates. The Stitching Workflow function could then initiate a chain mode provision as part of its overall workflow.

8 Stitching Example Using Component Design

This section presents a description of how the component design described in this document would be applied to Use Case One from the GENI Network Stitching Architecture - Example [2] document. Figure 1 below is the same figure as presented in [2], but has been modified to show usage of the specific API extensions and messaging describe in this document. This figure shows the steps associated with stitching together BBN and ProtoGENI Aggregates. The other stitching operations required for Use Case One would be handled in a similar manner. An extended description of the component interactions is provided below which reference the numbered steps in Figure 1.

A. ListResources (geni_stitching=only)

This item is numbered with a letter A to indicate that retrieval of the Stitching Resource Elements by the Topology Service is an ongoing activity. That is, the Topology Service continuously works to maintain its topology database so that it is prepared to respond to topology requests from the path computation function. The Topology Service will be populated by Stitching RSPECs from each aggregate. These Stitching RSPECs will be similar to what is shown above in Section 7.2 for the MAX Stitching RSPEC. The Topology Service will use the ListResources command with the geni_stitching=only flag set to retrieve the Stitching RSPEC from each aggregate. The lastUpdateTime element from the topology schema can be inspected to quickly determine if udpated information is present.

1. ComputeSliceStitching Request

This is the Client Application requesting a Stitching Path Computation. At this point in the process the Client Application has decided that it would like a slice which consists of slivers on the BBN and ProtoGENI aggregate. In addition, it has formulated the RSPECs that it will submit to both of these aggregates. The Client Application also knows that it would like the slivers on BBN and ProtoGENI to be stitched together. As a result the Client Application has enough information to send a request to the Path Computation Function. This request will be submitted via the ComputeSliceStitching call of the Path Computation API. This will include an rspec argument containing a Stitching RSPEC formatted in accordance with the "path" element of the topology schema. The exact schema format is defined by the xsd document shown in Appendix A. A pseudo version of what is included in this path computation request is shown below:

```
<path>
hop1
hoptype=strict
link=urn:publicID:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1
        capacity=1Gbps
        interfaceMTU=9000
        vlanRangeAvailable=any
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=1+link=1
        capacity=1Gbps
```

```
          interfaceMTU=9000
          vlanRangeAvailable=any
nexthop=null
</path>
```

At this point the Client Application has asked the Stitching Workflow to compute a path which will allow BBN and ProtoGENI Aggregates to be stitched together between specific edge ports in each aggregate.

2. GetStitchingTopology (geni_sts_all)
This is the Stitching Workflow Function retrieving the global topology view from the Topology Service. The Stitching Workflow Function uses the GetStitchingTopology with the geni_sts_all flag set. This results in the Stitching Workflow Function receiving the Stitching RSPECs from all the GENI Aggregates and External Networks.

3. Path Computation Processing
Based on the information received in step 2, the Stitching Workflow Function can compute a path to satisfy the requests received in step 1. This will involve taking the information received in step 2 and constructing a connected network graph, using the computation algorithms of choice, and formatting the result in a manner compatible with the common topology schema. For this example, the result has found a path which is BBN:Internet2 ION:ProtoGENI. An alternate path of BBN:Internet2 ION:MAX:ProtoGENI could also have been selected, but for this example we will assume only one option will be returned. The format of the response is described in the next step.

4. ComputeSliceStiching Response
In this step the computation result is passed back to the Client Application via the response to the ComputeSliceStitching Request. This response will include an rspec argument containing a Stitching RSPEC formatted in accordance with the "path" element of the topology schema. The exact schema format is defined by the xsd document shown in Appendix A. A pseudo version of what is included in this path computation request is shown below:

```
<path>
hop1
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=any
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=2+link=2
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=any
```

```
nexthop=3
hop3
hopetype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node=NEWY+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=any
          vlanTranslation=true
nexthop=4
hop4
hopetype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node:SALT1+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=any
          vlanTranslation=true
nexthop=5
hop5
hopetype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=3+link=3
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=any
nexthop=6
hop6
hopetype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=any
nexthop=null
</path>
```

Here we can see the path computation has expanded the initial request from the client which has just two hops, to have 6 hops which specifically defines the ingress and egress for every aggregate and external network in the path. This forms the basis to go to the dynamic realtime resource identification, to see if bandwidth is available, and which vlans to use.

<u>5. CreateSliver</u>
This step represents the Client Application sending the RSPEC to the individual aggregates to request the host level resources be instantiated for each sliver.

<u>6. CreateSliceStitching Request</u>
This step is the Client Application sending a request to the Stitching Workflow Function asking it to instantiated the stitching between the BBN and ProtoGENI Aggregates. This will be

submitted via the CreateSliceStitching message of the Stitching Workflow API which will include an rspec argument which contains the path element from step 4.

The Stitching Workflow Function now has enough information to begin stitching workflow processing.

7. Stitching Workflow Processing
In this step the Stitching Workflow Function examines the path element received in the request. The context of the path element is a request from the client to instantiate this stitching. Here are some typical steps that will be recognized and done at this time:

- This particular path element identifies all the hops in the path as strict, so there is no need to request additional path computation services.
- The path element defines the ingress and egress points for three aggregates/external networks: BBN, Internet2 ION, ProtoGENI
- The Stitching Workflow Function will request information from the Topology Service using the geni_sts_aggregate[bbn.com, ion.internet2.edu,protogeni.net] option.
- From the above information, the workflow function will identify the following about each aggregate/external network:
  - BBN: stitchingmode=tree; negotiatedservices=true
  - Internet2 ION: stitchingmode=tree; negotiatedservices=false
  - ProtoGENI: stitchingmode=chainAndTree; negotiatedservices=false
- The workflow process will also observe from the path computation results that vlanTranslation=true for the Internet2 ION network.

Based on the above information, the stitching workflow will decide on the following sequence for contacting the Aggregate Managers and External Networks:
- Contact ProtoGENI (with negotiation off) and see what VLAN is utilized
- Contact BBN (with negotiation on) and present it with the suggested VLAN from ProtoGENI results. Receive back suggested vlan and vlan range.
- Submit request to Internet2 ION (with negotiation off) with suggested vlans and vlan ranges tailored based on responses from BBN and ProtoGENI.

A specific example of the Stitching Workflow to Aggregate Manager/External Network interactions is presented in the following steps.

8. CreateStitchingSliver Request/Reponse (negotiate off) to ProtoGENI

The request has a path element with the following:

```
<path>
hop1
hopetype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=3+link=3
        capacity=1Gbps
        interfaceMTU=9000
        vlanRangeAvailable=any
```

```
nexthop=2
hop2
hopetype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=any
nexthop=null
</path>
```

The response has a path element with the following:

```
<path>
hop1
hopetype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=3+link=3
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=1001
nexthop=2
hop2
hopetype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=1001
nexthop=null
</path>
```

Here we see ProtoGENI has selected vlan 1001.

9. CreateStitchingSliver Request (negotiate on) to BBN

The request has a path element with the following:

```
<path>
hop1
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          suggestedVlanRange=1001
          vlanRangeAvailable= any

nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=2+link=2
```

```
          capacity=1Gbps
          interfaceMTU=9000
          suggestedVlanRange=1001
          vlanRangeAvailable= any
nexthop=null
</path>
```

The response has a path element with the following:

```
<path>
hop1
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          suggestedVlanRange=3728
          vlanRangeAvailable= 3727-3735
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=2+link=2
          capacity=1Gbps
          interfaceMTU=9000
          suggestedVlanRange=3728
          vlanRangeAvailable= 3727-3735
nexthop=null
</path>
```

Here we see BBN has suggested vlan 3728.

10. CreateStitchingSliver Request (negotiate off) to Intenet2 ION

The request has a path element with the following:

```
<path>
hop1
hopetype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node=NEWY+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          suggestedVlanRange=3728
          vlanRangeAvailable=3727-3735
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node:SALT+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
```

```
        suggestedVlanRange=1001
        vlanRangeAvailable=1001
nexthop=null
</path>
```

The response has a path element with the following:

```
<path>
hop1
hoptype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node=NEWY+port=1+link=1
        capacity=1Gbps
        interfaceMTU=9000
        vlanRangeAvailable=3728
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node:SALT+port=1+link=1
        capacity=1Gbps
        interfaceMTU=9000
        vlanRangeAvailable=1001
nexthop=null
</path>
```

Here we see Internet2 ION has configured vlan 3728 facing BBN and translated to vlan 1001 facing ProtoGENI.

<u>11. CreateStitchingSliver Request (negotiate off) to BBN</u>

The request has a path element with the following:

```
<path>
hop1
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1
        capacity=1Gbps
        interfaceMTU=9000
        suggestedVlanRange=3728
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=2+link=2
        capacity=1Gbps
        interfaceMTU=9000
        suggestedVlanRange=3728
nexthop=null
</path>
```

The response has a path element with the following:

```
<path>
hop1
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1
        capacity=1Gbps
        interfaceMTU=9000
        vlanRangeAvailable= 3728
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=2+link=2
        capacity=1Gbps
        interfaceMTU=9000
        vlanRangeAvailable= 3728
nexthop=null
</path>
```

Here we see BBN has configured vlan 3728.

12. <u>CreateSliceStitching Response</u>

If we combine all the final responses the Stitching Workflow function received in steps 8-11, we see that the a stitched path is in place between BBN and ProtoGENI using Internet2 ION as a transit external network. This path is defined by the ingress and egress links of each aggregate/external network as follows:

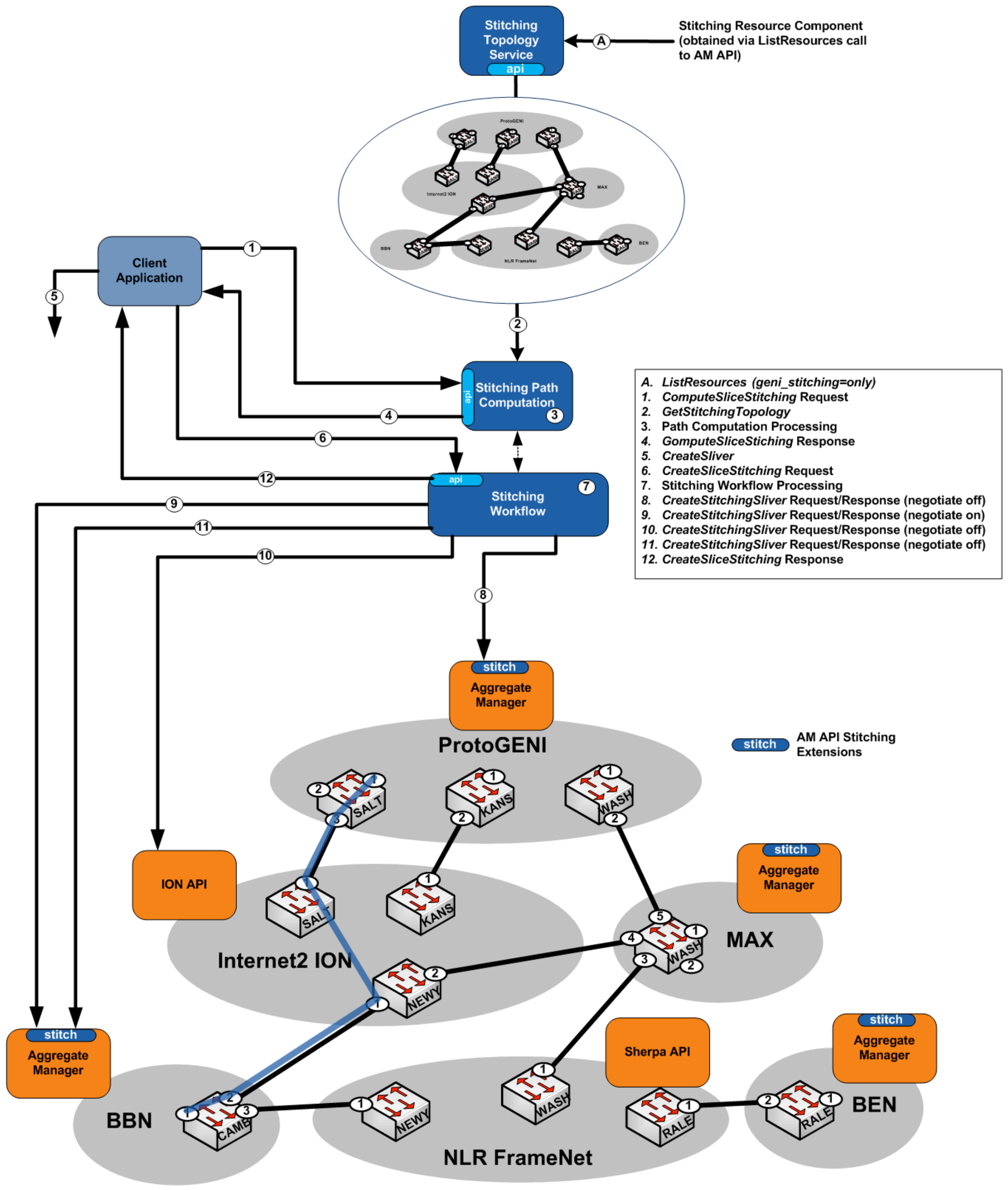
```
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1, VLAN=1001
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=2+link=2, VLAN=1001
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node=NEWY+port=1+link=1, VLAN=1001
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node=SALT+port=1+link=1, VLAN=3728
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=3+link=3, VLAN=3728
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=1+link=1, VLAN=3728
```

Combining the path elements received in steps 8-11, we can create the following path element which describes the instantiated path:

```
<path>
hop1
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=1+link=1
        capacity=1Gbps
        interfaceMTU=9000
        vlanRangeAvailable= 3728
nexthop=2
hop2
hoptype=strict
link=urn:publicid:IDN+aggregate=bbn.com+node=CAMB+port=2+link=2
```

```
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable= 3728
nexthop=3
hop3
hoptype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node=NEWY+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=3728
nexthop=4
hop4
hoptype=strict
link=urn:publicid:IDN+aggregate=ion.internet2.edu+node:SALT+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=1001
nexthop=5
hop5
hoptype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=3+link=3
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=1001
nexthop=6
hop6
hopetype=strict
link=urn:publicid:IDN+aggregate=protogeni.net+node=SALT+port=1+link=1
          capacity=1Gbps
          interfaceMTU=9000
          vlanRangeAvailable=1001
nexthop=null
</path>
```

This path element is sent back to the client as part of the CreateSliceStitching Response.



A. *ListResources (geni_stitching=only)*
1. *ComputeSliceStitching* Request
2. *GetStitchingTopology*
3. Path Computation Processing
4. *GomputeSliceStiching* Response
5. *CreateSliver*
6. *CreateSliceStitching* Request
7. Stitching Workflow Processing
8. *CreateStitchingSliver* Request/Response (negotiate off)
9. *CreateStitchingSliver* Request/Response (negotiate on)
10. *CreateStitchingSliver* Request/Response (negotiate off)
11. *CreateStitchingSliver* Request/Response (negotiate off)
12. *CreateSliceStitching* Response

Figure 2 Use Case One Example with Component Design Details

References

[1] GENI Network Stitching Architecture-Overview,
https://geni.maxgigapop.net/twiki/bin/view/GENI/NetworkStitching

[2] GENI Network Stitching Architecture-Example,
https://geni.maxgigapop.net/twiki/bin/view/GENI/NetworkStitching

[3] GENI Aggregate Manager API 1.0, Available at
http://groups.geni.net/geni/wiki/GAPI_AM_API

[4] OGF NML (Network Markup Language) Working Group,
http://www.ogf.org/gf/group_info/view.php?group=nml-wg

[5] NDL (Network Description Language), http://www.science.uva.nl/research/sne/ndl

[6] DICE IDCP Control Plane Topology Schema, *Available at http://www.controlplane.net/idcp-v1.1/nmtopo-ctrlp.xsd*

[7] IETF CCAMP (Common Control and Measurement Plane)
http://datatracker.ietf.org/wg/ccamp/charter/

[8] IETF PCE (Path Computation Element) http://datatracker.ietf.org/wg/pce/charter/

Appendix A - Common Stitching Topology Schema Example

(also available here in xsd format:
https://geni.maxgigapop.net/twiki/bin/view/GENI/NetworkStitching)

```
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://geni.net/schema/stitching/topology/geniStitch/20110220/"
    xmlns:GeniStitch="http://geni.net/schema/stitching/topology/geniStitch/20110220/">

    <xs:annotation>
      <xs:documentation>
        This is an example schema for the GENI Common Topology Schema.
      </xs:documentation>
    </xs:annotation>

  <!-- Aggregate Topology -->
  <xs:element name="topology" type="GeniStitch:GeniStitchTopologyContent"/>

  <xs:complexType name="GeniStitchTopologyContent">
    <xs:sequence>
      <xs:element name="lastupddatetime" type="xs:string"/>
      <xs:element name="AggregateUrl" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="GeniStitch:path"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="GeniStitch:aggregate"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="aggregateSignature"
        type="GeniStitch:GeniStitchAggregateSignatureContent"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:string"/>
  </xs:complexType>
  <!-- /Aggregate Topology -->

  <!-- Aggregate -->
  <xs:element name="aggregate" type="GeniStitch:GeniStitchAggregatgeContent"/>

  <xs:complexType name="GeniStitchAggregateContent">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="aggregatetype" type="xs:string"/>
        <xs:element minOccurs="0" maxOccurs="1" name="stitchingmode" type="xs:string"/>
        <xs:element minOccurs="0" maxOccurs="1" name="scheduledservices" type="xs:string"/>
        <xs:element minOccurs="0" maxOccurs="1" name="negotiatedservices" type="xs:string"/>
         <xs:element minOccurs="0" maxOccurs="1" name="lifetime" type="GeniStitch:Lifetime"/>
         <xs:element minOccurs="0" maxOccurs="unbounded" ref="GeniStitch:node"/>
         <xs:element minOccurs="0" maxOccurs="unbounded" ref="GeniStitch:port"/>
         <xs:element minOccurs="0" maxOccurs="unbounded" ref="GeniStitch:link"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:string"/>
  </xs:complexType>
  <!-- /Aggregate -->

  <!-- Node -->
  <xs:element name="node" type="GeniStitch:GeniStitchNodeContent"/>

  <xs:complexType name="GeniStitchNodeContent">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="lifetime" type="GeniStitch:Lifetime"/>
        <xs:element minOccurs="0" name="address" type="GeniStitch:GeniStitchAddressContent" />
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="GeniStitch:port"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
  </xs:complexType>
  <!-- /Node -->

  <!-- Port -->
  <xs:element name="port" type="GeniStitch:GeniStitchPortContent" />
  <xs:complexType name="GeniStitchPortContent">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="lifetime" type="GeniStitch:Lifetime"/>
        <xs:element minOccurs="0" name="capacity" type="xs:string"/>
        <xs:element minOccurs="0" name="maximumReservableCapacity" type="xs:string"/>
        <xs:element minOccurs="0" name="minimumReservableCapacity" type="xs:string"/>
        <xs:element minOccurs="0" name="granularity" type="xs:string"/>
        <xs:element minOccurs="0" name="unreservedCapacity" type="xs:string"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="GeniStitch:link"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
  </xs:complexType>
  <!-- /Port -->


  <!-- Link -->
  <xs:element name="link" type="GeniStitch:GeniStitchLinkContent" />

  <xs:complexType name="GeniStitchLinkContent">
      <xs:sequence>
        <xs:element minOccurs="0" name="remoteLinkId" type="xs:string" />
        <xs:element name="trafficEngineeringMetric" type="xs:string"/>
        <xs:element minOccurs="0" name="capacity" type="xs:string"/>
        <xs:element minOccurs="0" name="maximumReservableCapacity" type="xs:string"/>
        <xs:element minOccurs="0" name="minimumReservableCapacity" type="xs:string"/>
        <xs:element minOccurs="0" name="granularity" type="xs:string"/>
        <xs:element minOccurs="0" name="unreservedCapacity" type="xs:string"/>
        <xs:element minOccurs="0" maxOccurs="unbounded"
            name="linkProtectionTypes" type="xs:string"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="administrativeGroups"
            type="GeniStitch:GeniStitchAdministrativeGroup"/>
        <xs:element name="SwitchingCapabilityDescriptors"
type="GeniStitch:GeniStitchSwcapContent"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
  </xs:complexType>
   <!-- /Link -->

  <!-- Path -->
  <!--Path element is used to describe a simple point-to-point path-->
  <xs:element name="path" type="GeniStitch:GeniStitchPathContent" />

  <xs:complexType name="GeniStitchPathContent">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="lifetime" type="GeniStitch:Lifetime" />
        <xs:element minOccurs="0" maxOccurs="unbounded" name="hop"
type="GeniStitch:GeniStitchHopContent" />
        <xs:element minOccurs="0" maxOccurs="1" name="status" type="xs:string" />
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
      <xs:attribute name="direction" use="optional" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="GeniStitchHopContent">
      <xs:sequence>
        <xs:element minOccurs="0" name="hopeType" type="xs:string" />
        <xs:element minOccurs="0" name="aggregateIdRef" type="xs:string" />
        <xs:element minOccurs="0" name="nodeIdRef" type="xs:string" />
        <xs:element minOccurs="0" name="portIdRef" type="xs:string" />
        <xs:element minOccurs="0" name="linkIdRef" type="xs:string" />
        <xs:element minOccurs="0" ref="GeniStitch:aggregate"/>
        <xs:element minOccurs="0" ref="GeniStitch:node"/>
        <xs:element minOccurs="0" ref="GeniStitch:port"/>
        <xs:element minOccurs="0" ref="GeniStitch:link"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="nextHop"
                    type="GeniStitch:GeniStitchNextHopContent" />
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="GeniStitchNextHopContent">
    <xs:simpleContent>
        <xs:extension base="xs:string">
```

```
          <xs:attribute use="optional" name="weight" type="xs:int"/>
          <xs:attribute use="optional" name="optional" type="xs:boolean"/>
        </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="Lifetime">
      <xs:sequence>
        <xs:element name="start" type="GeniStitch:TimeContent" minOccurs="0"/>
        <xs:element name="end" type="GeniStitch:TimeContent" minOccurs="0"/>
        <xs:element name="duration" type="GeniStitch:Duration" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:string"/>
      <xs:attribute name="direction" use="optional" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="TimeContent">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute use="required" name="type" type="xs:string"/>
        </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="Duration">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute use="required" name="type" type="xs:string"/>
        </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <!-- /Path -->

<!-- Service Toplogy -->
  <!--Service Topology elements will be used to describe more complex topologies like
   a point-to-multipoint, multipoint-to-multipoint, and paths-with-protection.
   This will described in a later revision
   -->
<!-- /Service Toplogy -->

  <!-- Misc Types -->
  <xs:complexType name="GeniStitchAggregateSignatureContent">
    <xs:attribute name="aggregateId" use="required" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="GeniStitchAddressContent">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute use="optional" name="value" type="xs:string"/>
            <xs:attribute use="optional" name="type" type="xs:string"/>
        </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="GeniStitchSwcapContent">
    <xs:sequence>
      <xs:element name="switchingcapType" type="xs:string" />
      <xs:element name="encodingType" type="xs:string" />
      <xs:element name="switchingCapabilitySpecificInfo"
        type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="GeniStitchAdministrativeGroup">
    <xs:sequence>
      <xs:element name="group" type="xs:int"/>
      <xs:element minOccurs="0" name="groupID" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="GeniStitchSwitchingCapabilitySpecificInfo">
    <xs:sequence>
```

```
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_Psc1"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_Psc1"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_Psc2"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_Psc2"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_Psc3"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_Psc3"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_Psc4"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_Psc4"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_L2sc"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_L2sc"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_Tdm"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_Tdm"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_Lsc"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_Lsc"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="switchingCapabilitySpecificInfo_Fsc"
type="GeniStitch:GeniStitchSwitchingCapabilitySpecificInfo_Fsc"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="GeniStitchSwitchingCapabilitySpecificInfo_L2sc">
    <xs:sequence>
      <xs:element minOccurs="0" name="capability" type="xs:string"/>
      <xs:element  minOccurs="0" name="interfaceMTU" type="xs:int"/>
      <xs:element minOccurs="0" name="vlanRangeAvailability" type="xs:string"/>
      <xs:element minOccurs="0" name="suggestedVLANRange" type="xs:string"/>
      <xs:element minOccurs="0" name="vlanTranslation" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
   <!-- /Misc Types -->

</xs:schema>
```