Slice-based Network Transit Service:
Inter-Domain L2 Networking on ExoGENI

Yuanjun Yao, Qiang Cao, Jeff Chase  
Department of Computer Science  
Duke University  
yjjiao, qiangcao, chase]@cs.duke.edu

Abstract—The GENI network testbed was designed to enable experimentation with network protocols by offering the capability to construct virtual networks at the link layer (L2). GENI users build virtual networks in their GENI slices that span resources on multiple GENI aggregates, linked by L2 network circuits from network fabric providers (e.g., Internet2). This paper summarizes new features for cross-slice stitching in the ExoGENI deployment. They enable slice owners to create L2 links crossing slice boundaries within ExoGENI racks (OpenStack cloud sites), including links to slices owned by other tenants. Like other operations on virtual networks in ExoGENI, cross-slice stitching is dynamic and programmatically controlled. Users can adapt the network topologies of their slices and their interconnections with other slices over time.

Cross-slice stitching expands the potential of GENI as a platform for inter-domain network experimentation, including tenant slices that provide high-speed network-based services to other slices. This paper reports on a simple demonstration experiment for ExoGENI cross-slice stitching: a network transit service that runs within a slice, manages a shared circuit topology on behalf of customer slices that link to it, and routes customer traffic over its shared network backbone topology. This experiment shows the potential of cross-slice stitching to address a key limitation of GENI: the cost and limited availability of high-speed network circuits for use within GENI virtual networks.

I. INTRODUCTION

The slice abstraction of GENI [1] and its predecessors offers a natural container for multi-cloud applications that span multiple cloud sites and/or providers in a distributed cloud. A slice is a named set of virtual resources—such as VMs and L2 network links—allocated from infrastructure providers in a networked multi-cloud. A virtual resource that is managed independently of other resources is called a sliver. Slices are the granularity of access control for GENI slivers: a user with permission to control a given slice can control all of its slivers. Therefore, we can view a slice as a virtual network domain controlled by its owners.

In the GENI multi-cloud, slice owners (tenants) can create virtual network topologies at the data-link layer (L2) within their slices. Compute nodes within a slice communicate over the slice dataplane, which is a linked private network that may span multiple sites within the multi-cloud. The cross-site links are typically bandwidth-provisioned L2 network circuits obtained from providers outside of GENI, e.g., the Internet2 AL2S and ESnet research network fabrics. These circuits enable high-speed slice dataplanes across sites, and experimentation at higher layers of the protocol stack including L3.

However, communication between GENI slices has been restricted to interaction at L3, over the public Internet, as in the earlier PlanetLab model. Our goal is to support direct cross-slice networking at L2: it is more flexible to experimenters and the links are bandwidth-provisioned, so they are less vulnerable to contention from competing traffic.

To enable slices to connect to one another at L2, we introduce new features for dynamic cross-slice stitching in ExoGENI [2], [3], [4], one of the two GENI “racks” deployments. A cross-slice stitch is effectively an L2 network peering point between two slice networks. If the slices are owned by different tenants then the stitch requires mutual consent.

Cross-slice peering enables GENI to serve as an extensible platform for inter-domain networking across slices controlled by different tenants. Tenants may cooperate to build and maintain inter-domain networks above GENI’s foundational platform for networked infrastructure-as-a-service. For example, slices with high-speed network circuits may function as virtual autonomous systems as a basis for new approaches to inter-domain routing. More generally, cross-slice stitching generalizes the PlanetLab idea of unbundled management [5]—in which user slices extend the testbed platform by providing useful network-based services to other slices—to encompass flexible, elastic L2 inter-networking.

This paper reports on an exercise in inter-domain networking based on ExoGENI slice peering. We first outline ExoGENI’s new features for cross-slice stitching. We then report on an exemplary use case: a network transit service that runs within a slice, manages a shared circuit topology on behalf of its customer slices, and routes customer traffic over its internal dataplane (see Figure 1). This sharing of links in this example can mitigate the limited availability of high-speed network circuits, an important “pain point” in GENI today.

We intend that this capability will promote flexible experimentation with inter-domain networking in GENI in the future, e.g., for Software-Defined Exchanges (SDX), elastic NFV services, layered network storage infrastructures, and dynamic routing over diverse transit services.

1 This work is supported by the following NSF grants: ACI-1440715, ACI-1642140, ACI-1642142, CNS-1330659, and NSF GENI sub-awards.
II. BACKGROUND AND OVERVIEW

GENI and ExoGENI. This paper presents a new capability for slice-to-slice stitching in ExoGENI, and its use to build multi-domain virtual networks. ExoGENI is a networked IaaS testbed within the GENI federation. ExoGENI itself is a federation of xCAT/OpenStack cloud clusters spanning 20 locations across 4 countries on 3 continents, linked by the Internet2 AL2S and ESnets circuit fabrics. These infrastructure providers are called aggregates in GENI. ExoGENI control software supports end-to-end automated stitching of each L2 slice dataplane across multiple ExoGENI aggregates, bridging among circuit providers at ExoGENI exchange points (e.g., at Starlight) as needed.

Managing slice-based network topologies. Like other operations on virtual networks in ExoGENI, cross-slice stitching is dynamic and programmatically controlled. Our approach builds on earlier API extensions and internal mechanisms to allow tenants to modify existing slices and slivers dynamically [6]. In contrast, in other parts of GENI (e.g., the InstaGENI racks) a user creates all resources of a slice at each aggregate in one shot, and cannot modify a slice’s footprint at any aggregate once the slice is instantiated there.

In this way, ExoGENI offers flexible support for tenants to build slices with multiple points of presence on multiple ExoGENI sites (e.g., for edge services), and to link them with a bandwidth-provisioned network backbone (dataplane) that is built to order for the needs of the slice. The modify operations enable the tenant to adapt its dataplane topology as its needs change. A slice has OpenFlow SDN control over its own L2 dataplane and inter-site traffic flow over its circuit links, so network policies governing packet flow are also under the control of the slice; the default policy is a single L2 domain spanning the slice dataplane.

Semantic resource models. ExoGENI is able to offer this flexibility in part through its use of a logic language (NDL-OWL [7], [8]) to describe slice requests, instantiated slivers, and sliver relationships. The control software uses the language to make statements about slivers. A semantic resource model is a set of statements about resources in the logic language. The ExoGENI control software (based on ORCA [3]) includes plugin extensions to orchestrate and provision slices by operating on these models. Slivers have globally unique names, so it is easy for a model to represent relationships among slivers in different sites or domains. The statements in the model are distinct and separable, and relevant relationships are encoded in the statements themselves rather than in the structure of the description document (e.g., as it is in GENI’s XML-based RSpec). As a result, it is easy to change a model by adding or removing statements, or to combine models (e.g., across sites or domains) by taking their union. These properties enable the ExoGENI control software to modify slivers and slices dynamically and to reason about inter-domain infrastructure, e.g., to orchestrate end-to-end stitching automatically.

Controllers and the Ahab API. While semantic resource models have proven to be powerful, it is difficult for users (and their programs) to produce and understand them. We introduce a library—Ahab—to hide logic concerns behind a new provisioning API to construct and modify slices. Tenants may implement long-running Ahab controller programs to manage their slices. A controller uses the API to construct, query, and modify the slice. For example, a controller can specify a sequence of changes to a slice and then commit those changes to launch the provisioning engine and inject requests into ExoGENI via a selected Slice Manager (SM) server. An ExoGENI SM determines how to map requests onto available aggregates, and orchestrates the execution of those requests in concert with their Aggregate Manager (AM) servers. The slice controllers for our transit slice experiment use the new Ahab API (§III-B) to build and interconnect the transit and customer slices (§IV). The Ahab controllers also send commands to nodes within the slice over SSH.

The role of Ahab is similar to geni-lib [9], which provides a programmatic API for GENI requests. While geni-lib is interoperable with ExoGENI, it is limited to functions that are available using GENI-standard protocols and RSpec resource descriptions. Ahab goes beyond geni-lib in that Ahab enables sequenced changes to existing resources, including dynamic stitching to existing stitching points. The functions described here for dynamic slices, topology adaptation, and cross-slice stitching are not yet available through standard GENI protocols. Ahab accesses these functions using auxiliary protocols in ExoGENI in concert with NDL-OWL semantic resource descriptions and ORCA protocols. Note, however, that GENI slices can stitch to one another within ExoGENI even if they incorporate other resources outside of ExoGENI.

Stitching and stitchports. The stitch operation is a foundational building block for building dynamic L2 virtual networks. A controller program builds a slice by requesting various slivers and stitching them together. Each stitching point connects a pair of compatible slivers, e.g., a virtual node and a virtual link that are topologically adjacent at some aggregate. In a cross-slice stitch, one slice exposes a virtual node or link as a type of named stitchport, so that another slice with suitable authority can stitch to it, creating an L2 peering point between the two slices.

A stitchport is a named meeting point where two independent network domains can be stitched together, establishing a L2 connection. We have used “static” stitchports for a
number of years to connect ExoGENI virtual tenant networks (slices) to science assets outside of GENI [10]. A static stitchport is registered with an ExoGENI metadata service as a long-term network-facing endpoint attached to a fixed network segment (VLAN) behind the stitchport. This paper extends the stitchport concept with support for dynamic virtual stitchports exposed (or withdrawn) programmatically by the slices themselves. Controllers may use Ahab APIs to create and expose a slice stitchport, or stitch to another slice’s stitchport (§III-A).

Example: Network Transit Slice. As a running example, we consider a simple transit slice \( S \) that routes network traffic over its dataplane on behalf of customer slices that attach to it (e.g., \( A \) and \( B \)). In this example, the intent is to carry a customer’s traffic from one of the customer’s points-of-presence to another, and not necessarily to allow communication between the customers. This example has a compelling motivation: it frees \( A \) and \( B \) from the need to request cross-site network circuits for their own dataplanes, conserving the scarce supply of network circuits available to GENI. The transit service is suitable for GENI slices that want high-speed dataplane connectivity but are willing to multiplex their traffic over circuits shared with other transit customers.

To use the transit slice, a customer must conform to the transit provider’s L3 network model (if any) to allow traffic routing. We experiment with two simple alternatives for the transit provider: (1) a virtual L2 transit service that bridges disjoint L2 networks in a customer slice, and (2) an L3 domain that uses OSPF internally to route traffic among non-conflicting IPv4 prefixes advertised to it by its customers.

In the future we plan to extend this example to serve as a template for slice-based Software-Defined Exchanges (SDX). An SDX is an intermediary service that orchestrates L2/L3 interconnections according to the policies of customers attached to it. An SDX is often thought of as a physical facility where customer domains have presence and connectivity. However, cross-slice stitching and circuit providers with broad reach enable slice-based virtual SDX services that are decoupled from physical SDX peering points. Our intent with the virtual SDX concept is to provide an initial template for GENI-based experimentation with novel virtual SDXs outside of a dedicated physical SDX testbed.

These applications of inter-domain stitching demand richer control over trust and authorization. This paper focuses on an initial approach using simple authorization based on a sparse capability (a “hard-to-guess” random number used as a bearer token) to protect access to each dynamic stitching point. In the future we plan to use a logical trust management to address issues including stitchport access control, off-by-default interconnection of SDX customers by mutual consent, and ownership of IP prefixes (e.g., grounded in RPKI certifications) advertised through peering points.

III. BUILDING INTER-DOMAIN L2 NETWORKS

This section presents more detail on cross-slice stitching (§IIIA) and the Ahab controller API (§IIIB), and how our network transit experiment uses them.

A. Cross-slice Stitching

Stitching between ExoGENI slices is prototyped as new set of API calls with simple parameters that drive the underlying stitching functionality. In this implementation, any node-like or link-like sliver—a virtual machine, a bare-metal node, a point-to-point or a multi-point link between nodes—can become a stitching point. A stitch is only possible between a node-like and a link-like sliver, and the slivers must be hosted on the same aggregate and provider substrate/site. Currently at most one stitch is allowable between any pair of slivers.

In principle, it is possible to implement stitch requests for slivers on different aggregates or slivers of like types. For example, the cross-aggregate case might instantiate new link slivers to extend the specified link to the site that hosts the specified node. Any link-to-link stitching involves establishing a translation of VLAN tags (or other labels) for the stitched links, in order to join them together. These extensions are future work: the current API supports only local stitches between one link and one node.

To instantiate a local cross-slice stitch, the host aggregate’s AM merely creates a new network interface on the specified node and attaches it to the specified link. The basic add interface/remove interface sliverModify operations were implemented earlier as part of ExoGENI’s generic sliceModify() machinery [6]. Once the L2 stitch is established, an existing ExoGENI-specific extension (“neuca”) [2] within the node’s OS recognizes the new interface and configures it up (e.g., with ifconfig). Our experiments use node-to-link stitching, which allows the requester (the node owner) to pass an IP address for the new interface as a parameter.

Rather than explicit advertising of long-lived link stitchports, our prototype for dynamic stitching relies on bearer-token based authorization of stitching operations between slices, with one tenant (tenant \( S \)) allowing stitching operations to a particular sliver (e.g., a link serving as a dynamic stitchport) to any peer slice who knows a specified secret token. \( S \) then communicates the token out of band to tenant \( A \). \( A \) then requests a stitch, passing the token as proof of access.
The slice stitching prototype adds five user-facing API calls to the ExoGENI Slice Manager (SM) servers. This API may be invoked by command-line scripts or by ExoGENI’s GUI tools and client libraries (e.g., Ahab):

- **permitSliceStitch**(sliceID>, <sliverID>, <secret>). Establish a bearer-token secret for a stitchable sliver in a slice. One active secret per sliver is currently permitted. Internally a secret is stored as a salted hash of the secret and stored as a property of the sliver.
- **revokeSliceStitch**(sliceID>, <sliverID>). Revoke any secret for a stitchable sliver, blocking future requests to stitch to it. Existing stitches are unaffected.
- **performSliceStitch**(from sliceID>, <from sliverID>, <to sliceID>, <to sliverID>, <secret>, <stitch properties>). Request a cross-slice stitch between two compatible slivers. After checking that the slivers are colocated at the same AM, the SM invokes the AM to install the stitch. The stitch properties may include IP address, bandwidth attributes, etc.
- **undoSliceStitch**(from sliceID>, <from sliverID>, <to sliceID>, <to sliverID>). Break an existing L2 stitch. Either side may break the stitch unilaterally. This serves as an “emergency break”, e.g. in the event of a cross-slice network attack.
- **getSliceStitchInfo**(sliceID>, <sliverID>). Request information about stitching state and events on a sliver owned by the caller. Reports whether this sliver allows stitches (a secret is set) and its stitching history: start and end times, identities of slices, slivers and their owners. Currently active stitches are determined by absence of end-time for the reported stitch. Each stitch operation is given a unique GUID.

Our Ahab controllers for the transit slice experiments use these API calls, as shown in Figure 2.

**B. Instantiating and Connecting Slices Programmatically**

Many GENI users are familiar with using ExoGENI through the GENI tools based on standard RSpec (e.g. Jacks [11] or geni-lib). The GENI API and its tools are crucial for experiments that require slices spanning multiple testbeds in or geni-lib). The GENI API and its tools are crucial for experiments that require slices spanning multiple testbeds in the GENI federation. However, the advanced functions of ExoGENI—including automated dynamic stitching and cross-slice stitching—are available only through its native API.

Client programs may call ExoGENI’s native API directly, but these API calls produce and consume logical/semantic resource descriptions in NDL-OWL (§II). Initially, the only way for a user to invoke the native API without confronting NDL-OWL was to use ExoGENI’s Flukes GUI to draw and visualize slice topologies. This has long been a hurdle to using ExoGENI for complex experiments that demand more automation or that cannot be drawn easily in Flukes.

This paper introduces the Ahab Java library, which allows users to create and control ExoGENI slices programmatically while accessing all of the functionality provided by the native API. The Ahab API enables developers to create, modify, and destroy slices and resources on ExoGENI using simple Java objects. It is easy to incorporate it into a Java program: it is available from a Nexus Maven repository as a Maven dependency. With Ahab, it is now possible to create large complex slices easily and to create controller applications that can monitor, reason about, and modify long-running slices based on arbitrary policies.

The remainder of this section discusses the Ahab slice/sliver abstractions and the tools and workflow used to create and manage slices with Ahab. It offers simple Java classes and objects to abstract ExoGENI slices and resources. The primary object types used by an Ahab controller application are:

- **Slice**. Represents a slice.
- **Resource Objects**. A set of types representing each type of sliver available: ComputeNode, Network, StorageNode, Stitchport. These objects hold the configuration information for slivers, e.g. a ComputeNode’s boot script or a Network’s bandwidth. They also serve as an access point for accessing manifest information, e.g. a ComputeNode’s management IP address or the IDs of the VLANs that compose a Network.
- **Interface**. The interface between two stitched resources. An interface contains information specific to that type of stitch. For example, an Interface between a Network and a ComputeNode might contain an IP address.
- **SliceProxy**. Manages the user’s credentials required to submit requests to an ExoGENI Slice Manager (SM) server. Creating a SliceProxy requires a user’s GENI certificate and private key, as well as the url of a preferred ExoGENI SM to use.
- **SliceContext**. Manages the user names and public SSH keys that are to be installed in nodes within a slice.

The following is an example of the workflow that is used to instantiate a simple slice (some of the code is simplified for brevity):

```java
1 proxy = getSliceProxy(cert, key, url);
2 context.addToken(userName, pubKey);
3 s = Slice.create(proxy, context, name);
4 ComputeNode n = s.addComputeNode("n0");
5 n.setImage(imageURL, imageHash, imageName);
6 n.setNodeType(nodeType);
7 Network net = s.addBroadcastLink("net0");
8 Interface if0 = net.stitch(n);
9 if0.setIpAddress("172.16.0.1");
10 if0.setNetMask("255.255.255.0");
11 s.commit();
12 s.wait();
13 String ip = n.getManagementIP();
```

This code builds a SliceProxy and SliceContext in lines 1 and 2. These objects manage the certificates and keys to interact with the new slice and the ExoGENI provisioning services. Line 3 shows how to initialize a Slice object using the SliceProxy and SliceContext. No resources are provisioned for the slice until the code requests a sequence of updates to the empty slice (lines 4-10) and then commits them (line 11). In this example, line 4 adds a ComputeNode called “n0”. Lines 5 and 6 set the node’s image and node type. Line 7 adds a network. Line 8 stitches the node to the network.
creating an Interface. Lines 9 and 10 set the IP and netmask of the Interface. The commit action at line 11 forms the NDL request and sends it to the ExoGENI Slice Manager (SM) specified in the proxy. The commit returns as soon as the SM accepts the request, but the steps to provision the slice and its slivers proceed asynchronously. The Ahab controller may wait() (as in line 12) until the pending updates to the slice are complete (or failed). When wait() returns, the Ahab controller might perform additional configuration and/or send requests into the slice to kick off an experiment. In this example, the ComputeNode object is used to access manifest information that is necessary to complete the configuration and run an experiment: line 13 shows the application obtaining the management IP address of the ComputeNode. It can then use this IP to copy executables and other files to the ComputeNode and invoke any necessary tasks, e.g., with ssh and scp, authenticated by the keys in the SliceContext.

### IV. Demonstration Experiments

We conducted simple demonstration experiments for ExoGENI cross-slice stitching: we deployed two variants of a slice-based network transit service that passes customer traffic over its slice dataplane, i.e., it acts as a carrier slice. An Ahab controller for the transit slice runs as a service (outside of the slice itself), and accepts requests from customers to stitch to the transit slice at particular PoPs—cloud sites that host nodes and/or links for both slices. For each request it returns a sliverID and token that the customer slice may stitch to (see Figure 2). Each customer slice is controlled by its own Ahab program, which requests stitches from the transit controller.

The transit service provides connectivity to customer slices through its peering points. For example, tenant slices $A$ and $B$ may interconnect their PoPs through the transit slice, as an alternative to allocating dedicated network circuits for their own dataplanes. Figure 3 depicts an example using a simple dumbbell topology for a transit slice occupying two PoPs—the configuration we use in all of our experiments. The PoPs for all slices are instantiated on ExoGENI sites at two universities in Florida: UNF and UNL.

Once a cross-slice stitch is in place, either peer may inject arbitrary traffic into the other slice across the slice boundary. As with any inter-domain networking, each slice controls how it directs traffic into the peering point and how it handles incoming traffic. We experimented with two approaches for the transit slice:

- **L3/OSPF.** The transit slice and all customer slices run a standard IPv4 stack on Linux VMs (Ubuntu 14.04) enabled for Quagga/OSPF and configured with non-conflicting IP prefixes (RFC 1918 private addresses). When a cross-slice link is instantiated, each peer automatically advertises its routes to the other.

- **L2/SDN.** The nodes in the transit slice run Open vSwitch (OVS) with a shared SDN controller—an extended Ryu SimpleSwitch controller called PriorityNetwork—that implements a single L2 domain. PriorityNetwork can segregate traffic from the different customers (e.g., by swallowing ARPs and blocking broadcasts) and schedule traffic from different customers according to assigned priority weights. It ignores packet header fields above L2, and so preserves GENI’s flexibility for network experimentation at L3 and above.

The Ahab controllers take minutes to instantiate a virtual network for a slice: the delay is limited primarily by the time for the ExoGENI rack sites to launch VMs on their underlying OpenStack/Linux/KVM clusters. We used I2/AL2S network circuits for the transit slice: requests for such links are faster to provision, but they are often denied due to resource constraints.

Once the nodes and links are active, a cross-slice stitch takes 10 seconds or less. In fact, the stitch is instantiated almost immediately—the Ahab wait() primitive reports that the stitch is active—but the neucia extension within a node polls at 10-second intervals to discover each new interface and configure it up. The customer’s Ahab controller uses ssh (via the Jsch Java library) to notify the slice when the stitch request commits and wait() returns.

To illustrate, the Ahab customer controller for the L3/OSPF example works as follows. It uses scp (Jsch) to copy a stitch configurator script onto each node in its slice as it comes up. When a newly requested stitch is complete (wait() returns), the controller invokes the configurator program via ssh on the peering node within the slice, passing the configured IP address. The configurator loops until ifconfig reports an interface with the expected IP address. It then writes the interface data into Quagga’s configuration file and kicks Quagga to reload its configuration, which triggers OSPF route advertisements across the new interface. We found that it takes about 50s for Quagga to propagate new routes across all nodes.

Our measurements suggest that traversing a stitch has effectively zero performance cost. Single TCP streams (iperf with default parameters) between VMs on the same aggregate reliably yield 9.2-9.3 Gb/s over the 10 Gb/s Ethernet interconnect used for ExoGENI dataplanes, regardless of whether the nodes reside in the same slice or the traffic traverses a cross-slice stitch. CPU utilization is below 70% on the sender and below 85% on the receiver (single-core VM, XO-medium).
This result substantiates our view that inter-domain networking at L2 is a useful vehicle for GENI experiments involving data-intensive and high-speed network services that run in slices. Moreover, slice-based services accessed by cross-slice stitching are reasonably secure: connectivity is “off by default” and is enabled only by mutual consent, and the peer is strongly authenticated at least by its sliceID.

Figure 4 reports results of an experiment in which multiple attached customers pass traffic between their PoPs through an L2/OSPF transit slice with single shared 5 Gb/s I2/AL2S network circuit connecting the PoPs. Customer flows in this example use TCP. As expected, the TCP congestion algorithm naturally shares the link fairly among the flows. As discussed in §II, this simple demonstration example is not practical without additional authorization (e.g., to validate IP prefixes) and perhaps traffic policing in the transit/carrier slice.

The L2/SDN alternative illustrates the flexibility of network control within the carrier slice. In this example, the nodes in the carrier slice control network traffic with an OpenFlow SDN controller that implements a “big learning switch” at L2. The SDN controller is extended to use OpenFlow QoS queues to provide differentiated service to different customers. We use different OpenFlow queues for traffic between different pairs, with different relative shares 1, 3 and 6. We measured the bandwidth each pair gets as link capacities of the carrier slice change. As shown in figure 4, the bandwidth each pair gets is in proportion to its configured share weight.

V. CONCLUSION

This paper presents an approach to dynamic inter-domain (inter-tenant) L2 networking in a distributed IaaS cloud—ExoGENI. It is based on a new cloud primitive for cross-slice L2 stitching that connects the dataplanes of peering slices directly by mutual consent, yielding a high-speed friction-free network path between the slices. Our approach extends GENI to inter-domain networking experiments, while preserving GENI’s flexibility for alternative approaches at L3 and above. We show that these features enhance GENI’s potential for layering of slice-based services above the GENI foundation. In particular, they are useful for tenants and their slices to provide secure services to one another—e.g., NFV, SDX, inter-domain routing, storage backbone—with fast L2 connectivity that is “off by default”.

REFERENCES