Wide-area Software Defined Networking Experiments using Chameleon

Mert Cevik and Paul Ruth
Renaissance Computing Institute
{mcevik,pruth}@renci.org

Kate Keahey
Argonne National Laboratory
keahey@anl.gov

Pierre Riteau
StackHPC
pierre@stackhpc.com

***Abstract*—Recent advancements have expanded Chameleon's support for networking experiments by enabling deeply programmable networks spanning wide-areas and controlled by the user. New capabilities include: 1) bring-your-own-controller (BYOC) software defined networking (SDN) and 2) Layer 2 *stitching* to external testbeds and facilities including stitching between the two Chameleon sites.**

This paper presents the new networking capabilities of Chameleon along with corresponding experiments that evaluate limitations and features of using SDN in a wide-area environment. The experiments serve both as an evaluation of SDN in a wide-area environment and as a guide for designing advanced networking experiments on Chameleon.

I. Introduction

The Chameleon testbed represents an experimental instrument for Computer Science operating on the principles that users are allowed deeply reconfigurable access to environments that are isolated in ways relevant to the experiments each platform supports. Chameleon allows users to scale Big Compute and Big Data experiments to large amounts of compute and storage in different configurations using heterogeneous hardware. Recent advancements have expanded Chameleon's support for networking experiments by enabling deeply programmable networks spanning wide-areas and controlled by the user.

The two main networking capabilities that have been added to Chameleon are: 1) bring-your-own-controller (BYOC) software defined networking (SDN) and 2) Layer 2 *stitching* to external testbeds and facilities including stitching between the two Chameleon sites. BYOC networking allows users to allocate isolated hardware OpenFlow network switches controlled by custom or off-the-shelf controllers deployed and managed by the user. The use of OpenFlow allows users to have deep access to the network and deploy complex experiments beyond what is possible with traditional switched networks. In addition, these switches can be *stitched* using isolated layer 2 circuits (up to 100 Gbps) transiting wide-area circuit providers, such as Internet2 AL2S [1] and ESnet OSCARS [2]. Stitched networks allow users to connect programmable switch hardware between Chameleon sites (University of Chicago and Texas Advanced Computing Center (TACC)), as well as connect Chameleon switch hardware to other testbeds (e.g. GENI [3]), facilities, or even a user's home institution.

Although these new capabilities provide deeply programmable networking, users must carefully implement their experiments to achieve their desired results. Specifically, users should be aware of host tuning options for high-latency connections, as well as the effect of OpenFlow controller placement with respect to switch location.

In a typical OpenFlow deployment the switch has a subset of the controller's flow rules in its tables at any given time. When a packet arrives that does not match any flows known to the switch, the packet is forwarded to the controller which processes the packet and pushes a new flow rule back to the switch. The switch installs the flow in one of its tables and uses that flow to process future matching packets.

As the latency between the controller and the switch increases, the time necessary to configure the switch also increases. Even relatively small latency in switch configuration can result in dropped or delayed packets, which will negatively affect the performance of the network. Ideally, BYOC experiments should be designed to have the OpenFlow controller co-located with the switch and deployed within a Chameleon host at the same site as the network. However, co-location is not possible for many experiments (e.g. a single controller used for multiple distributed switches). In these cases, it is important to understand the effect of controller latency on an experiment.

This paper presents new networking capabilities of Chameleon along with corresponding experiments that evaluate limitations and features of using SDN in a wide-area environment. The experiments serve both as an evaluation of SDN in a wide-area environment and as a guide for designing advanced networking experiments on Chameleon. Specifically, we evaluate the effect placement of controllers with respect to SDN switches has on OpenFlow flow insertion times, TCP bandwidth, and the performance of wide-area multi-path routing.

II. Background

Chameleon is largely built using OpenStack [4], a widely supported, open source, cloud computing software. This section presents the building blocks used to enable the BYOC and stitching experiments: OpenStack, Corsa DP2000 series switches, and dynamic wide-area Layer 2 circuits.

A. *OpenStack*

OpenStack is composed of several services which manage compute, storage, and networking resources, as well as provide

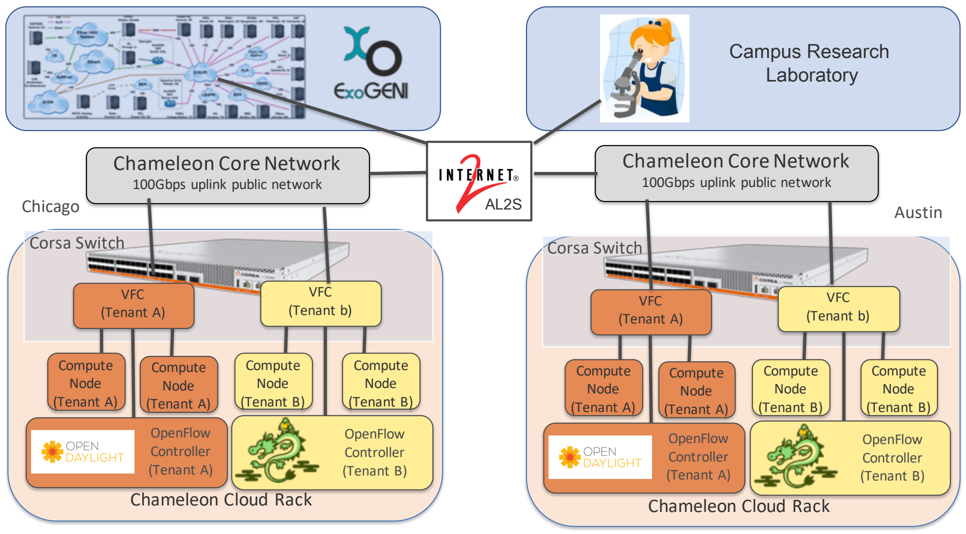

Figure 1. Each Chameleon site has been extended to include Corsa DP2000 series switches that are used to create mutually isolated line-rate OpenFlow forwarding contexts controlled by the user. The forwarding contexts can be stitched via Internet2 and ESnet to each other, ExoGENI, campus research laboratories, and other facilities.

identity management, monitoring, orchestration, etc.

Neutron is the Networking service within OpenStack. It implements APIs allowing users and administrators to define networking within an OpenStack cloud deployment [5]. It manages virtual networking infrastructures composed of networks, subnets, switches, and routers. It is also capable of configuring physical network devices, such as network switches. Neutron provides DHCP and NAT forwarding services for bare-metal nodes and virtual machines, and can provide advanced services such as firewalls and VPNs.

Neutron is composed of a main API server, plugins implementing specific networking configurations (such as VLAN or VXLAN), and agents providing various networking services that can be leveraged by compute instances in an OpenStack cloud (DHCP, L3 routing, access to metadata servers, etc.).

Users can define tenant networks, i.e. networks which are under their control and dedicated to their own use. Users configure these networks with a chosen IP subnet and connect them to external networks such as the Internet via virtual routers. These networks are isolated from the physical infrastructure and from other tenant networks. Giving full control of the network configuration to users allows them to define networks as required by their applications.

Ironic [6] is the Bare Metal service within OpenStack. It manages the configuration of bare metal nodes, deployment of disk images onto them, serial console access, etc. Originally, Ironic only supported placing all provisioned nodes and nodes under deployment into a single Layer 2 network shared by all tenants. This meant user-defined tenant networks could not be used for bare-metal, resulting in a lack of isolation between tenants and limited configuration capabilities for users. In the OpenStack Newton release, Ironic added support for networking multi-tenancy [7], allowing each tenant to define and use tenant networks isolated using VLAN tagging, with all the flexibility advantages described above.

Ironic interacts with Neutron to perform network configuration necessary to ensure nodes are on the right Layer 2 network. This can be done via Modular Layer 2 (ML2) drivers specialized for particular network devices or using networking-generic-switch (NGS), an extendable framework responsible for applying configuration information to networking hardware equipment [8]. NGS supports various families of switches from vendors such as Cisco, Huawei, Arista, and Dell Force10. The way NGS works is described as follows:

- When a new network is created (respectively deleted), Neutron allocates a VLAN ID to use as a *segmentation ID* for the network and invokes NGS to define (respectively undefine) the corresponding VLAN on each switch in the cloud infrastructure.
- When a bare metal instance is launched with a chosen network, Neutron invokes NGS to configure the port of the bare metal node hosting the instance as *access port* for the corresponding VLAN on the ToR switch.
- When a bare metal instance is terminated, Neutron invokes NGS to remove the port of the bare metal node from the corresponding VLAN on the ToR switch.
- NGS also supports configuring *trunk ports* for specific interfaces, such as uplinks or OpenStack controller nodes, which need to access a wide range of VLANs.

This paper presents an extension of the NGS plugin that enables provisioning of tenant controlled isolated OpenFlow switches using Corsa DP2000 series switches discussed in Section II-B.

B. Corsa DP2000 Series Switches

Corsa DP2000 series switches [9] are ideally suited for multi-tenant networking infrastructure. These switches are a deeply programmable switching platform that expose independent virtualized forwarding contexts (VFCs), seen in Figure 1, that operate at line-rate (up to 100 Gbps). Each VFC acts as an independent OpenFlow switch with its own controller and can forward traffic between a subset of the physical ports (or logical ports defined by a VLAN tag).

Multiple isolated tenants can be supported on the switch by creating independent VFCs for each tenant and attaching each tenant's nodes and VLANs only to their VFC. Deep programmability by the tenant can be achieved by attaching each VFC to a tenant-managed OpenFlow controller. Chameleon uses the Corsa's VFC abstraction to isolate tenant networks while providing tenant-controlled OpenFlow programmability.

C. Dynamic Wide-area Layer 2 Circuits

Recent advances made by wide-area network transit providers have enabled IT staff to connect campuses and other facilities using on-demand wide-area Layer 2 circuits. These next generation wide-area networking capabilities are increasingly necessary as we see more large-scale big data science collaborations. Example systems include Internet2's Advanced Layer 2 Services (AL2S) and DOE's ESnet.

Each of these providers capabilities can be accessed interactively or programmatically using well defined interfaces. ESnet uses the On-demand Secure Circuits and Advance Reservation System (OSCARS) while Internet2 AL2S uses the Open Exchange Software Suite (OESS). Before using such services, a campus or facility must be connected appropriately and a dynamic connection point must be named with a URN. After this initial configuration, campus or facility staff can create VLANs between their URN and the URN of any other authorized campus with a similar connection.

These dynamic Layer 2 circuits are commonly used to create temporary, high-bandwidth, dedicated, wide-area VLANs between facilities in order to transfer large amounts of data or to create complex wide-area network topologies for computer networking experiments (e.g. using GENI). Work presented in this paper describes how IT staff can connect tenant resources on Chameleon to their campuses and facilities using the dynamic advanced Layer 2 circuits.

III. Implementation

This section describes the general configuration and infrastructure required to enable BYOC and stitching to OpenStack clusters.

A. OpenStack Configuration: Layer2 Network Stitching

Previous sections discussed how dynamic Layer 2 circuit providers are used by GENI to stitch between cloud resources and to external campuses and facilities. Stitching Layer 2 circuits between OpenStack and external resources requires extending OpenStack Neutron networks outside the cluster to dynamic meeting points accessible by external domains. Figure 1 show how Chameleon follows ExoGENI [10] by referring to these meeting points as stitchports or sometimes *stitchable VLANs*.

OpenStack bare metal clusters use either a *flat* network shared by all tenants or VLANs to create isolated Layer 2 networks for each tenant. Each isolated network is configured with its own subnet and router providing layer 3 service for nodes connected to the network. Typically, an OpenStack cluster has a single pool of VLANs from which tenants request VLAN isolated networks.

Enabling external stitching in OpenStack relies on the same network isolation abstraction but extends a subset of the available VLANs to specific meeting points outside of the OpenStack cluster. If these VLANs extend to dynamic meeting points (e.g. Internet2 AL2S URNs), external facilities can be stitched to the OpenStack networks.

Each OpenStack cloud can access several stitchable meeting points, as well as have VLANs limited to the local site. The provider network abstraction is used to allow tenants to choose between VLANs and specific stitchable VLANs. Separate pools of VLANs are plumbed to each external stitching point. A separate provider network is created to manage each pool of VLANs. Each provider is assigned a separate virtual interface and VLAN range. Tenants can create isolated Neutron networks using standard OpenStack tools and specify the *provider network* to provision a VLAN that extends to a specific destination (e.g. ExoGENI).

There can be multiple stitching VLAN providers if there are multiple stitching paths (e.g. one for stitching to each testbed). As long as each of these virtual providers are assigned a disjoint set of VLANs, tenants that require stitchable VLANs will only need to request a VLAN from the provider network that manages access to the desired external stitchport.

Creating an OpenStack network mapped to a specific provider is normally reserved to administrators (as described in Section II-A). Using provider networks for stitching requires reconfiguring the policy to allow regular users to do so as well.

User Interface: In order to use Chameleon stitchable VLANs, a tenant must follow the existing CLI workflow (with slight modifications) for creating isolated tenant networks. The only change is that the provider network must be specified to be from the desired pool (e.g. *exogeni*). Upon creation of the network the CLI reports the *provider:segmentation_id* which is the VLAN that was assigned. Currently, there are 10 stitchable VLANs from each Chameleon site connecting to ExoGENI. Users must use the $segmentation\_id$ allocated by Chameleon to create an ExoGENI slice to complete the stitch. The remainder of the Chameleon workflow remains unchanged. An example CLI command is:

```
openstack network create                \
    --provider-network-type exogeni    \
    --provider-physical-network vlan \
    myTenantNetwork

provider:segmentation_id  | 3290
```

B. OpenStack Configuration: Corsa and BYOC

In a previous section we discussed OpenStack Neutron and the NGS plugin used by Chameleon to configure the network switches, including the Dell switches that are part of Chameleon's first-phase hardware. This work extends the standard NGS plugin to: 1) support Corsa DP2000 series switches, 2) support users to use their own OpenFlow controllers to control their networks, and 3) stitch networks to one or more external layer 2 circuits.

Recall that OpenFlow and NGS allow tenants to create isolated VLAN networks on many common types of networking hardware. This work extended the NGS plugin to support traditional OpenStack VLAN networking on Corsa switches. VLANs are implemented on the Corsas as a VFC that forwards traffic between a set of user nodes and a specified uplink VLAN. In this basic configuration, the VFC is configured as a learning switch by connecting to a Chameleon-managed OpenFlow controller. This functionality mimics that available through the NGS plugin on standard switches and allows Corsa switches to be used with a standard OpenStack VLAN network using the standard OpenStack API.

The addition of support for BYOC is similar to that of a standard isolated VLAN network except that the VFC is connected to a user-specified OpenFlow controller instead of the learning switch controller managed by Chameleon. BYOC uses the standard OpenStack networking API with the addition of an OpenFlow controller IP and port passed in the description field when creating the network. When the NGS plugin creates the isolated VFC it sets the OpenFlow controller to the IP and port in the description. Any controller IP and port can be used as long as the server is reachable using the Internet.

By default, every Chameleon network is assigned a VLAN that isolates traffic across the local Chameleon infrastructure. Recall that externally stitched networks use a VLAN from a pool that is pre-plumbed to externally facing stitchports. BYOC networks implement this by attaching a VLAN tagged logical port between the VFC and the physical uplink port on the switch. This results in a VFC that has one uplink port (possibly externally stitched) and one local port for each node connected to the network. Arbitrary OpenFlow rules can be applied to traffic in the VFC.

Users can add additional externally stitched circuits to any BYOC network. This is accomplished by assigning multiple OpenStack networks (and their associated VLANs) to a single VFC. When a network in created, the user can optionally name the network by adding another argument to the description field. The NGS plugin will associate all networks that specify the same name to a single VFC. Each Chameleon project has its own name space for network names so only VLANs within a project can be associated with the same VFC. When a VLAN is added to an existing VFC, a new logical uplink port associated with the new network's VLAN is added to the VFC, enabling the VFC to send traffic to the newly stitched circuit.

User Interface: In order to use Chameleon BYOC networks, a tenant must follow the existing CLI workflow (with slight modifications) for creating isolated tenant networks. The only change is that the controller must be added to the description field. The remainder of the Chameleon workflow remains unchanged. To specify a name for the VFC, *VSwitchName* should be appended to the *description* field. An example CLI command is:

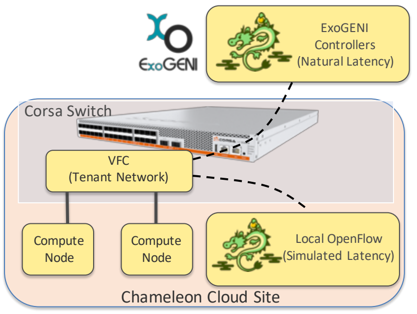

Figure 2. Experiments restricted to a single site contain a VFC connecting multiple nodes and controlled by either a local controller simulating latency with *tc* or a remote controller on ExoGENI.

```
openstack network create             \
  --provider-network-type exogeni    \
  --provider-physical-network vlan   \
  --description                      \
    OFController=<OF_IP>:<OF_Port>,\
    VSwitchName=<name>   \
    myTenantNetwork
```

IV. Evaluation

As discussed in previous sections, placement of an OpenFlow controller relative to the OpenFlow switch is important. Chameleon allows users to place their controllers at any IP accessible through the Internet. The rest of this section discusses a series of experiments evaluating the effect of controller placement.

Each experiment tests a different aspect of controller placement on BYOC performance. Several controller placements were used. The basic placement (Figure 2) deploys the controller in the same Chameleon rack as the switch and simulated increased latency using the *tc* tool. In cases where both Chameleon sites were used (Figure 3), a separate controller was placed in each site and latency was controlled with *tc*. We also experimented with naturally occurring latency by deploying the same Ryu controller on virtual machines at several sites across ExoGENI. Locations tested include: RENCI (Chapel Hill, NC), UH (Houston, TX), Starlight (Chicago, IL), Wayne State (Detroit, MI), UFL (Gainesville, FL), UVA (Amsterdam), UAF (Fairbanks, AK).

A. *Experiment 1: Flow Table Miss Reaction Time*

Recall that when a packet arrives at an OpenFlow switch, it is tested against all known flow rules. If it does not match any flows known to the switch, the packet is forwarded to the controller which processes the packet and pushes a new flow rule back to the switch. The time necessary to complete this process depends on the speed and complexity of the

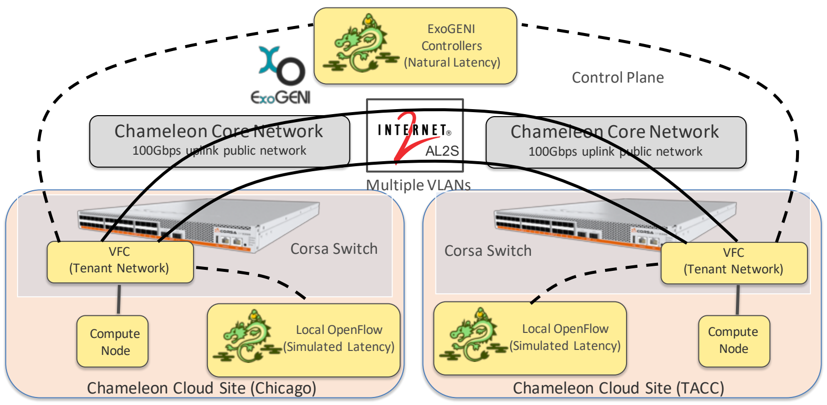
ExoGENI Controllers (Natural Latency)
Control Plane
ExoGENI
Chameleon Core Network 100Gbps uplink public network
INTERNET2 AL2S
Chameleon Core Network 100Gbps uplink public network
Multiple VLANs
Corsa Switch
VFC (Tenant Network)
Corsa Switch
VFC (Tenant Network)
Compute Node
Local OpenFlow (Simulated Latency)
Local OpenFlow (Simulated Latency)
Compute Node
Chameleon Cloud Site (Chicago)
Chameleon Cloud Site (TACC)

Figure 3. The multi-path routing experiment deploys VFCs on both Chameleon sites stitched together with two parallel AL2S circuits. Controllers are either co-located with the switches (simulated latency) or remotely deployed on ExoGENI (natural latency). The experiment uses a single path at any given time and finds the time necessary to switch between the paths.

controller hardware and software, as well as the network latency between the switch and the controller. This experiment measures the time required to install a flow caused by a table-miss with varying amounts of latency between the switch and the controller.

In the experiment, a BYOC network is deployed on the TACC Chameleon site. Two nodes are connected to the network with one node sending a ping (ICMP) packet to the other every 10ms. The OpenFlow controller programs the switch with permanent flows for all traffic, except for the flow rule forwarding traffic from the source of the ping to its destination. This flow has a hard timeout, which forces the switch to expire the rule periodically. The result is that when the flow expires, the next ICMP packet is sent to the controller, which responds

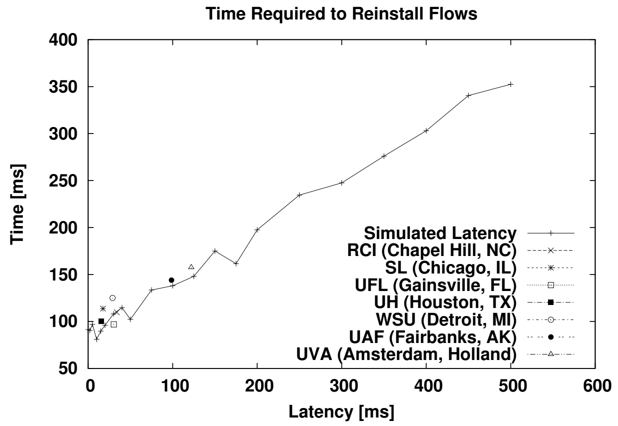

Figure 4. Time necessary to reinstall a flow that has timed out for various latencies between the switch and its controller. The line represents a co-located controller with simulated latency between the minimum possible (<1ms) and 512 ms. Individual dots show naturally occurring latency between the switch and controllers running on various ExoGENI nodes.

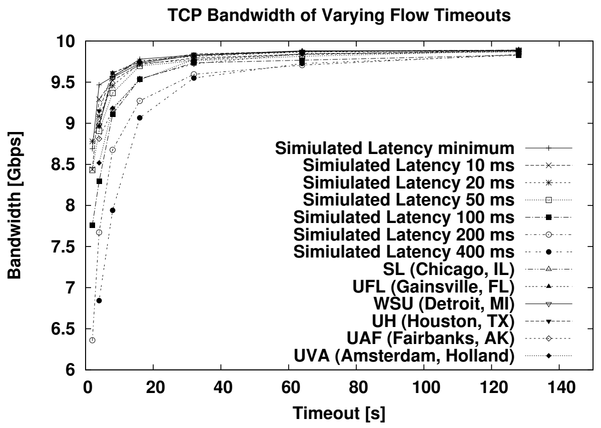

Figure 5. Single stream TCP bandwidth achieved by iperf3, across various hard timeouts, imposed on flows, with various simulated and natural latency between the switch and the controller.

with the same rule to be installed. While the new flow is being fetched, subsequent ICMP packets are dropped and never reach their destination. Once the flow is re-installed, ICMP traffic is successfully forwarded again. The experiment counts the number of ICMP packets that are dropped and calculates the time required to reinstall the flow based on the 10ms interval between packets.

The experiment is run using the Ryu controller configuration described above, with both simulated and natural latency. For each configuration latency, we recorded the time for 20 reinstalls. The resulting average flow reinstall times can be seen in Figure 4. The lowest achievable reinstall times were approximately 100ms with the highest naturally occurring reinstall time being from Amsterdam 157ms.

B. *Experiment 2: Effect of Flow Timeout on TCP Bandwidth*

Many OpenFlow configurations periodically induce dropped flows and subsequent flow re-installation. There are situations that require flows to be dropped and re-installed. Often this is a result of timeouts required for each rule that indicate when the switch should automatically remove the rule. After a rule times out, the next packet that would have matched that rule will instead be forwarded to the controller, which will re-install a new rule (possibly the same rule). The switch installs the flow in one of its tables and uses that flow to process future matching packets. As the latency between the controller and the switch increases so does the time necessary to configure the switch, negatively affecting the performance of the network. Increased reinstall time generally results in increased numbers of dropped packets.

TCP is particularly affected by dropped packets. In this experiment we use `iperf3` to send a 180-second single TCP stream bandwidth test between two nodes in the same Chameleon rack. The controller is configured with a hard timeout that triggers a flow rule re-install periodically. For each artificial and natural latency we vary the hard timeout and record the bandwidth using `iperf3`. For each case, hosts were tuned using the ESnet's suggested tuning [11]. Tests with high latency and/or low timeout achieve much lower bandwidth. Figure 5 can be used to design experiments that require bandwidth.

C. *Experiment 3: Wide-area Path Change*

The final experiment involves two BYOC switches, one on each Chameleon site, that are stitched together with two separate Internet2 AL2S circuits (i.e. two paths). Nodes at each sites are connected to the switches and can send traffic to each other. The controller is configured to send traffic between each site along one path at a time. When the controller sees specific traffic characteristics, it will switch the path used by all traffic. In the experiment we use ARP packets for specific IPs to trigger a path change. Other multi-path experiments might use congestion or traffic type to modify the path choice.

The experiment measures the time required to switch between two paths in reaction to an observed traffic characteristic. We vary the latency as in the other experiments and present the time required between when the characteristic traffic (ARP packets for a specific IP) is introduced and when traffic starts to use the other path.

Figure 6 shows the results. The minimum amount of time is seen with low latency configurations and requires 275 ms, while the longest amount of time required for a natural latency is the ExoGENI VM in Amsterdam, which requires 564 ms to switch the path.

V. Conclusions

This paper has presented work enabling BYOC and stitching of experiments across Chameleon, ExoGENI, and campus facilities. Further it evaluated controller placement with respect to switch location and provided useful information for users to design successful networking experiments on Chameleon.

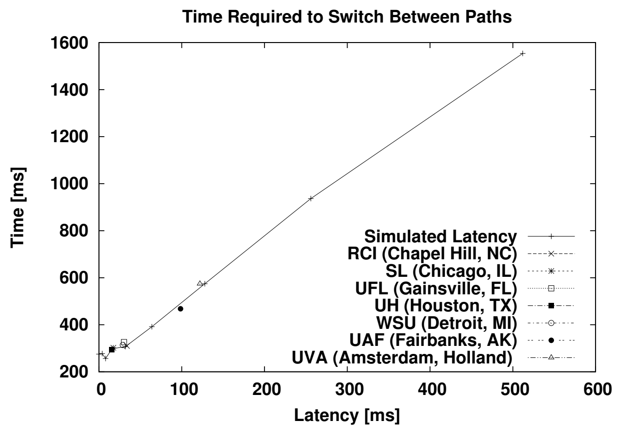

Figure 6. Time necessary to switch between two parallel stitched paths. The line represents a co-located controller with simulated latency between the minimum possible (<1ms) and 512 ms. Individual dots show naturally occurring latency between the switch and controllers running on various ExoGENI nodes.

Acknowledgment

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

[1] Internet2 contributors, "Advanced layer2 service (al2s)," http://noc.net.internet2.edu/i2network/advanced-layer-2-service.html.

[2] C. Guok, D. W. Robertson, M. R. Thompson, J. Lee, B. Tierney, and W. E. Johnston, "Intra and interdomain circuit provisioning using the oscars reservation system." in *BROADNETS*. IEEE, 2006. [Online]. Available: http://dblp.uni-trier.de/db/conf/broadnets/broadnets2006.html#GuokRTLTJ06

[3] R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds., *The GENI Book*. Cham: Springer International Publishing, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-33769-2

[4] OpenStack contributors, "OpenStack is open source software for creating private and public clouds," https://www.openstack.org, 2019.

[5] Neutron contributors, "OpenStack Docs: OpenStack Networking Guide," https://docs.openstack.org/neutron/latest/admin/, 2019.

[6] Ironic contributors, "Ironic documentation," https://docs.openstack.org/ironic/latest/, 2019.

[7] ——, "OpenStack Docs: Multi-tenancy in the Bare Metal service." https://docs.openstack.org/ironic/latest/admin/multitenancy.html, 2019.

[8] Networking-generic-switch contributors, "Networking-generic-switch Neutron ML2 driver," http://git.openstack.org/cgit/openstack/networking-generic-switch/tree/README.rst, 2019.

[9] Corsa, "Corsa DP2000 Product Family," https://www.corsa.com/products/, 2018.

[10] I. Baldin, J. S. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills, "Exogeni: A multi-domain infrastructure-as-a-service testbed." in *The GENI Book*, R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds. Springer, 2016, pp. 279–315. [Online]. Available: http://dblp.uni-trier.de/db/books/collections/MBER2016.html#BaldinCXMRCOHM16

[11] ESnet contributors, "Esnet fasterdata knowledge base," http://fasterdata.es.net/.