Virtual Network Migration on the GENI Wide-Area SDN-Enabled Infrastructure

Yimeng Zhao, Samantha Lo, Ellen Zegura, Mostafa Ammar
Georgia Institute of Technology
Email: {yimeng.zhao,samantha,ewz,ammar}@cc.gatech.edu

Niky Riga
Raytheon BBN Technologies
Email: nriga@bbn.com

Abstract—A virtual network (VN) contains a collection of virtual nodes and links assigned to underlying physical resources in a network substrate. VN migration is the process of remapping a VNs logical topology to a new set of physical resources to provide failure recovery, energy savings, or defense against attack. Providing VN migration that is transparent to running applications is a significant challenge. Efficient migration mechanisms are highly dependent on the technology deployed in the physical substrate. Prior work has considered migration in data centers and in the PlanetLab infrastructure. However, there has been little effort targeting an SDN-enabled wide-area networking environment — an important building block of future networking infrastructure. In this work, we are interested in the design, implementation and evaluation of VN migration in GENI as a working example of such a future network. We identify and propose techniques to address key challenges: the dynamic allocation of resources during migration, managing hosts connected to the VN, and flow table migration sequences to minimize packet loss. We find that GENI’s virtualization architecture makes transparent and efficient migration challenging. We suggest alternatives that might be adopted in GENI and are worthy of adoption by virtual network providers to facilitate migration.

I. INTRODUCTION

Virtualization is well-recognized as a technique to share physical resources, providing the appearance of dedicated resources and isolation from others sharing the same physical resources. Virtual networks run over a physical network substrate, with an allocation of physical network resources (e.g., routers, switches, links, paths, or portions thereof) to the virtual network. Network virtualization allows significant flexibility in network operation. Most important are the flexibility in the VN’s placement (the specific mapping of VNs elements to substrate resources [1]) and VN agility (the ability to remap the VN to a different set of substrate resources over time). Our interest in this paper is on enabling VN agility through VN migration mechanisms. This refers to the process of remapping some or all of a VN’s logical topology to a new set of physical resources.

VN migration research considers both policy, when and why a VN is migrated, and mechanism, how a VN is migrated. Research into VN migration policy is motivated by specific objectives. These have included: efficient utilization of dynamic resources [2], [3], recovery from failure [4], [5], defending against attacks [6], and reducing energy consumption [7].

Our focus in this paper is on VN migration mechanisms. The development of such mechanisms can be quite challenging because of the desire to make them transparent or seamless — informally, with minimal impact on running applications. A further challenge is that a VN migration mechanism is highly dependent on the technology deployed in the substrate. There is, therefore, no generic mechanism that can be used universally. Previous research has developed mechanisms for VN migration in different environments. In the data center context, Ghorbani et al. develop migration methods that focus on a transparency definition based on valid behaviors in a migration-free setting [8]. In the wide-area context, Lo et al. [9] develop a tool to migrate virtual routers in PlanetLab [10]. Their PL-VNM tool implements a migration schedule heuristic that minimizes packet loss under ideal conditions, but in practice cannot ensure zero loss, due in part to packet filtering rules and coarse timing control in PlanetLab.

In this paper we focus on developing VN migration mechanisms for software-defined network (SDN) on GENI, a recently developed infrastructure for sharing wide-area network resources [11]. Our focus on GENI is motivated by the fact that SDN-enabled wide-area networks are likely to become an important building block of future networking and GENI represents a fully-functional instantiation of this technology. As such, techniques developed for GENI will have wider applicability. Further, because SDN technology is at a stage where its future can be influenced, lessons we learn about the capability of such technology in supporting network agility can have significant value on future developments.
connected to the VN. Figure 1 illustrates the migration steps assuming an SDN-enabled infrastructure. A migration controller interacts with the SDN controller to initialize and schedule the migration process. Prior to migration, virtual switches on the VN1 are controlled by the client application running on the SDN controller, and VN1 is used to deliver traffic (Step 1). When the migration starts, VN2 is setup (Step 2) and flow tables on the virtual switches in VN1 are cloned to the virtual switches in VN2 based on the mapping (Step 3). The migration controller issues commands to reconnect hosts from VN1 to VN2 in Step 4 and to disconnect VN1.

This paper addresses several challenges in realizing the basic VN migration steps above. In addressing these challenges we make the following contributions: (1) We develop approaches that enable VN agility in the SDN-enabled GENI infrastructure; We evaluate options for dealing with the dynamic allocation of resources inherent in migration (Section II); (2) Propose an approach for managing the hosts that will remain in place when the VN migrates and develop techniques to mitigate the disruption caused by live VN migration. We carefully manage process steps and flow table migration sequences to minimize packet loss observed by the application in the data plane (Section III); (3) Design a migration controller that orchestrates the VN migration on GENI (Section IV); (4) Evaluate, using an implementation running on GENI, how the performance of live VN migrations as a function of design decisions and network parameters (Section V); and (5) Expose some limitations of the GENI infrastructure and propose approaches to their mitigation (Section VI).

II. ENABLING FULL VN AGILITY IN GENI

GENI is relatively mature shared infrastructure. As such GENI provides support for sharing, isolation, resource allocation, and multiple physical substrate “owners”. GENI, however, was not designed specifically to support our desired transparent and efficient VN agility. Our work, therefore, involves the development and evaluation of options to support VN agility within GENI. This section explores some critical aspects of providing agility support. Because GENI uses its own unique terminology, Table I summarizes how the general VN terminology maps to GENI terms.

| Component                                      | GENI Context    |
|------------------------------------------------|-----------------|
| Substrate networks                             | GENI testbed    |
| Virtual Network(s)                             | GENI slice      |
| Physical location                              | GENI aggregate  |
| Virtual links within a VN                      | LANS            |
| Virtual links between VNs                      | Shared VLAN     |
| Virtual links connecting different physical locations | Stitched links |
| Mapping between VN to physical substrate       | Rspec file      |

Table 1: GENI Context vs. Virtual Components

1Note that this model for hosts differs from the data center context where hosts are migrated with the VN. In shared wide-area infrastructure, we assume the hosts are customer-premise equipment and remain in place when the VN moves.

A. Allocating VNs to Slice(s)

GENI is a “sliced” platform that allows concurrent experiments on shared infrastructure. A GENI slice is a unit that contains all the resources for an experiment, including computing resources and network links. Slices exist in one or more aggregates; each aggregate is an independent collection of physical resources often operated by the same entity (e.g., a specific university). Figure 2 shows two slices on the GENI infrastructure. Slice 1 has three virtual nodes in Aggregate A, while Slice 2 has six virtual nodes across Aggregate A and Aggregate B connected with stitched link through the backbone network. Each slice is an isolated environment where virtual nodes and virtual links can be added.

Slices are meant to be deployed for the long term and are thus not agile. We consider two options for mapping VNs to slices with an eye to migration. The first option is to build all VNs (original and future) and hosts for migration within the same slice. This approach follows the common usage model for GENI to include all resources for an experiment within a single slice. However, this option is not dynamic. Most GENI resources cannot be modified after the reservation. In the case of migration, this restriction requires us to reserve all resources for hosts and VNs, including those that will be migrated to in the future, at the outset.

Alternatively, it is possible to allocate a single VN to a slice, starting with the original VN and later allocating a VN to migrate to. Deploying a VN on one slice is straightforward. The challenge for deploying and migrating a VN between two slices is caused by the difficulty to enable the inter-slice communication during migration. We cannot create a virtual link to connect virtual components in different slices directly. Instead, we can set up a VLAN, a broadcast domain at the data link layer, to connect virtual components in different slices. All virtual components in the same VLAN will receive the broadcasting packets even when the virtual components are in separate slices. Compared with deploying all VNs on one slice, this second design provides clear separation between VNs and gives more flexibility in resource reservation. We can reserve one VN first and create another VN when needed. However, it complicates the virtual topology during migration. We will talk further about shared VLANs and migration in Section III.

B. Mapping Virtual Switches to Physical Machines

While GENI aggregate’s automatic assignment of resources can meet the requirements of most experiments, it may be
necessary to have the flexibility of mapping virtual nodes to specific physical resources for VN migration research. Although Rspec generating tools do not directly support resource assignment, we are able to map a virtual node to a specific machine by manually modifying the request Rspec. We refer the readers to the accompanying technical report [12] for the detailed steps.

C. Assigning VNs to Substrates

In VN migration, it might be necessary to migrate between different physical substrates, or aggregates in GENI terminology. A GENI aggregate comprises a set of resources under common control including storage resources, computing nodes, and OpenFlow switches [11]. Experimenters can reserve multiple aggregates within the same slice and connect them with stitched links (See Figure 2). It is also possible to allocate each VN to a different aggregate and connect them with both shared VLAN and stitched links. We will show how to use shared VLAN and stitched links together in Section III.

III. DEALING WITH VN MIGRATION CHALLENGES

A. Inter-slice Connection

We described two VN-to-Slice allocation options in Section II-A. In the first option, all hosts, the old VN, and the new VN are located within the same slice. We will not discuss the first design in detail since it follows the common usage of the GENI testbed. We will focus on the second design, where the old VN, the new VN, and the hosts are assigned to three different slices. The challenge in the second design is to direct traffic from one slice to another given the current GENI constraints which do not support virtual links between slices. It should be noted that the dynamic tunnel implemented in LIME [8] does not apply for our case. The tunnel uses the control plane for data links and cannot guarantee performance such as bandwidth and latency. Moreover, the control plane is a shared channel on GENI and should not be used to send a large amount of data.

1) Broadcasting problem in a virtualization environment:

To enable inter-slice communication, it may seem natural to use a shared VLAN to connect a host slice to VN slices. The traffic is broadcast within the same VLAN, no matter in which slice a switch is located. The connection/disconnection of the VNs with the hosts are controlled by turning up/down the network interfaces on virtual switches. Figure 3 presents an example of this approach. The topology includes three hosts, the old VN (VN1), the new VN (VN2), and the controller slice. In our virtual topology, each host connects to both VN1 and VN2 with a shared VLAN. When host1 sends data to host2, the data will be broadcast to both OVS1 and OVS1’. When VN1 is in use, the network interfaces of OVS1 is up and the network interfaces of OVS1’ is down. After the migration, we redirect traffic from VN1 to VN2 by turning down the interfaces of OVS1 and turning up the interfaces of OVS1’.

Unfortunately, this approach can violate the correctness of the migration in a virtualized environment. GENI uses XEN [13] as a virtual machine monitor to allow multiple virtual machines to share the same hardware resources. Xen only allows a privileged virtual machine called domain 0 to access the physical Network Interface Card (NIC). Domain 0 communicates with other virtual machines through a set of back-end interfaces. All the packets destined to a virtual machine will be first transferred to domain 0 and then destined to the virtual machine. The packets stored in domain 0 are not dropped when the network interfaces in the virtual machine is turned down. When the virtual network interface goes up again, these buffered packets will be copied from domain 0 memory to the receiver virtual machine’s memory.

We illustrate why this small number of buffered packets can be a problem through a one-node VN example as shown in Figure 4. In actual substrate network, there is a rack switch residing in the shared VLAN to broadcast packets to all switches in the same VLAN. Before migration, the data from host1 to host2 is broadcast by Rack_SW1 to eth1 and eth2, and then broadcast by Rack_SW2 to eth2’ and host2. Although we turn down the virtual network interface eth1 and eth2’, a small number of packets are still stored in the XEN domain 0. During the migration, we switch from VN1 to VN2 by turning up eth1’ and eth2’ and turning down eth1 and eth2. Previously buffered packets in domain 0 are transferred through eth1’ and eth2’ to the virtual machine that hosts SW2. These packets have the same matching fields (e.g., same source and destination IP) but request different actions (e.g., send through different ports). In the worst case, when conflicting rules are installed on the openflow switch, the switch may stop forwarding packets, which requires manual configuration to recover.
2) Gateway design: To avoid the broadcasting problem, we propose a gateway design which establishes additional SDN switches as 'gateways' to switch traffic from the old VN to the new VN. The gateways are layer 2 devices that sit between hosts and VNs, hiding changes in VNs from end hosts. Figure 5 presents an example of the gateway design that enables migration within the same aggregate. Each host is connected with a gateway, and each gateway uses two different shared VLANs to connect to the two VNs. The gateway switch is responsible for forwarding packets from hosts to a certain VN. In the process of VN migration, the migration controller issues commands to the gateway switches, asking them to redirect traffic from VN1 to VN2. The controller sends SDN commands to the gateway switches to update the flow tables, redirecting traffic from VN1 to VN2. We therefore calculate the number of dropped data c_{1,2} as the traffic flow from host1 to host2, and c_{2,1} as the traffic flow from host2 to host1. We migrate the virtual network from VN1 to VN2. Before migration, GW1 directs f_{1,2} from in-port 1 to out-port 2, directs f_{2,1} from in-port 2 to in-port 1, and drops any traffic from in-port 3 to disconnect VN2. The same applies for GW2 to control traffic from/to host2. When the migration begins, our migration controller issues commands to GW1 and GW2 and updates their flow tables to redirect traffic from VN1 to VN2. We assume GW1 finishes update at time t_1,2 and GW2 finishes at time t_2,1. We define d_1 as the latency from GW1 to GW2 and d_2 as the latency from GW2 to GW1. The data rate of f_{1,2} is r_1 and the data rate of f_{2,1} is r_2. We therefore calculate the number of dropped data c_{1,2} for f_{1,2} and c_{2,1} for f_{2,1} as follows:

\[ c_{1,2} = \begin{cases} (t_{2,1} - t_{1,2}) - d_1 \times r_1, & \text{if } t_{2,1} - t_{1,2} \geq d_1 \\ (t_{1,2} - t_{2,1}) + d_1 \times r_1, & \text{otherwise} \end{cases} \]

\[ c_{2,1} = \begin{cases} (t_{1,2} - t_{2,1}) - d_2 \times r_2, & \text{if } t_{1,2} - t_{2,1} \geq d_2 \\ (t_{2,1} - t_{1,2}) + d_2 \times r_2, & \text{otherwise} \end{cases} \]

It is obvious that t_{2,1} - t_{1,2} = d_1 and t_{1,2} - t_{2,1} = d_2 cannot both be satisfied. At least one of c_{1,2} and c_{2,1} is larger than 0, which means additional packet loss is unavoidable in this setting.

1) Flow Migration Sequence: SDN shows promise to enable lossless migration with an optimized sequence of rule installation. We propose a scheduling sequence to remove the additional packet drop introduced by VN migration. Algorithm 1 shows pseudocode for the traffic redirection process. We install rules to let traffic coming from the new VN to go through the gateway switches. Then we update rules on gateway switches to direct traffic from hosts to the new VN. Finally, we insert drop rules to disconnect the old VN. By following this sequence, we avoid dropping packets buffered in the old VN.

**Algorithm 1 Traffic Redirection Algorithm**

1: for gateway \( \in \text{gatewayList} \) do
2: \( \text{Ports}_h \leftarrow \text{Ports on gateway that point to hosts} \)
3: for Port \( \in \text{Ports}_h \) do
4: \( \text{install new rule } r \text{ where } r.\text{inPort} = \text{PortToVN}_N2 \)
5: for gateway \( \in \text{gatewayList} \) do
6: for Port \( \in \text{Ports}_h \) do
7: update rule \( r \) set \( r.\text{outPort} = \text{PortToVN}_N2 \) where \( r.\text{outPort} = \text{PortToVN}_N1 \) and \( r.\text{inPort} = \text{Port} \)
8: for gateway \( \in \text{gatewayList} \) do
9: for Port \( \in \text{Ports}_h \) do
10: update rule \( r \) set \( r.\text{action} = \text{dropPkt} \) where \( r.\text{outPort} = \text{PortToVN}_N1 \) and \( r.\text{inPort} = \text{Port} \)

Algorithm 1 also applies to partial VN migration when only part of the old VN is remapped to different physical machines. In the partial VN migration, the traffic redirection occurs at the neighboring nodes of the partial network instead of the gateways. In this case, all neighboring nodes of the partial network should be treated as gateway switches and the same algorithm can be applied to minimize the packet loss.

IV. MIGRATION CONTROLLER ARCHITECTURE

The migration controller stands in the center of our migration architecture and is responsible for the migration process. It clones the flow tables from the old switches to the new switches, schedules the migration sequences and switches traffic between VNs. The controller architecture is shown in Figure 7.

**Mapping Module:** specifies how to map the switches in the old VN to the switches in the new VN. It also includes mapping of the virtual network interfaces in the old switches and in the new switches.
Flow Table Manager: When a request for VN migration is initiated, the Flow Table Manager polls switches that are affected by migration, translates flow tables from the old VN based on the mapping information stored in the Mapping Module, and installs the flows into the new switches.

Scheduler: calculates the sequence of rule installation based on our traffic redirection algorithm to minimize the packet loss.

VN Presenter: intercepts events from switches, translates them based on mapping information from the Mapping Module, and presents a consistent VN topology to client applications. This module hides all migration process from clients.

Migration API: The migration APIs allow client SDN applications to configure migration parameters such as migration destinations and triggering requirements. The applications should use migration API to retrieve virtual switch information, the connections to virtual switches, and events from virtual switches to get a consistent view of the VN.

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our migration mechanism in terms of the migration time and packet loss during migration. More evaluation results can be found in the accompanying technical report [12].

A. Packet Loss During Migration

1) Migrate a VN: We build a prototype of the migration controller using the POX controller [14] and evaluate its performance on GENI through experiments on the topology illustrated in Figure 5. We use iperf to generate UDP traffic for 10 seconds between all pairs of hosts and vary the data sending rate to see whether our migration controller works well in relatively high data rate. We perform three sets of experiments: (a) a baseline experiment where no migration occurs, (b) migration with symmetric routing, where traffic redirection commands are issued at the same time by controller, and (c) migration with asymmetric routing, where traffic redirection commands are issued in an optimized sequence. We repeat the experiments for 30 times for each data rate.

We only present results for forwarding and reverse flows between host1 and host3 due to space constraints. In Figure 8, the percentage of average packet loss on y-axis is based on the measurement of UDP traffic for 10s, and the x-axis shows baseline experiment, symmetric routing, and asymmetric routing for different data sending rates. For both the forwarding and the reserve flows, the packet loss rate in asymmetric routing is almost the same with that in a migration-free setting. It demonstrate that asymmetric routing prevents hosts from experiencing significant increase in packet loss during migration.

2) Impact of RTT: During the migration, packet loss occurs when packets buffered in the old VN are dropped by gateway switches because of the traffic redirection. To test the effect of the network latency on the packet loss, we use tc to simulate RTT ranging from 1ms to 300ms. As shown in Figure 9, the performance of the symmetric routing is much worse than that of the asymmetric routing. The average packet drop rate for symmetric routing increases linearly with the increase of the RTT while the packet drop rate for asymmetric routing is very close to zero for any RTT values.

B. Migration Time

Figure 10 shows how migration time changes as the flow table size grows with 95% confidence level upper and lower bands. The migration time is negligible when flow table size is small. It takes less than 1s to finish the migration when the flow table size is smaller than 1000. The migration duration increases roughly linearly with the number of rules per switch and can take 7s when there are 10,000 rules per switch (six switches in total). To test whether other parameters such as the topologies and the scales of the virtual networks can affect the migration time, we also experiment with a six-node mesh topology, a ten-node tree topology, and a twenty-node ring topology. Our experiments show that the migration time depends on the total number of rules but is independent of the topology.

VI. MITIGATING GENI LIMITATIONS

While it is possible to deploy virtual networks on GENI and use proper remote scheduling implementation to enable live migration, we observe that some GENI limitations complicate the design. These constraints are not only particular to our VN migration research, but may also apply to other types of experimentation. We summarize the features that are not well supported by GENI. This will aid in future GENI development and also in informing the designs of GENI-inspired SDN-enabled wide area infrastructure.
design their experiments. Each design is particular to a specific topology: whenever we need a new virtual topology, we need to reconsider the shared VLAN. The restriction in resource reservation makes it difficult to scale up an experiment.

VII. CONCLUSION

In this paper we consider the design, implementation and evaluation of virtual network migration on GENI as a working example of a future SDN-enabled wide-area infrastructure. VN migration adds network agility to the repertoire of network functions and provides a mechanism that enables the deployment of important policies for resource management, energy conservation and attack defense. We show how agility can be enabled on top of GENI’s slicing approach, enumerate and address challenges in the design of efficient VN migration mechanisms, and develop and deploy an implementation of a controller architecture that achieves our VN migration objectives. We perform a set of experiments that help us understand the implications of various design decisions and network parameters on VN migration performance. Our work also exposes some limitations on the current design of GENI.

VIII. ACKNOWLEDGEMENT

This work was funded in part by the National Science Foundation grant CNS-1319490.

REFERENCES

[1] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, “Virtual network embedding: A survey,” IEEE Communications Surveys & Tutorials, 2013.
[2] J. Fan and M. Ammar, “Dynamic topology configuration in service overlay networks: A study of reconfiguration policies,” in INFOCOM, 2006.
[3] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, “Vnr algorithm: A greedy approach for virtual networks reconfigurations,” in GLOBECOM, 2011.
[4] Y. Tang, E. A. Shaer, and R. Boutaba, “Efficient fault diagnosis using incremental alarm correlation and active investigation for internet and overlay networks,” Network and Service Management, IEEE, 2008.
[5] F. Gillani, E. Al-Shaer, M. Ammar, and M. Demirci, “Fine-grain diagnosis of overlay performance anomalies using end-point network experiences,” in CNSM, 2012.
[6] F. Gillani, E. Al-Shaer, S. Lo, Q. Duan, M. Ammar, and Z. Zegura, “Agile virtualized infrastructure to proactively defend against cyber attacks,” in INFOCOM, 2015.
[7] V. Eramo, E. Miucci, and M. Ammar, “Study of reconfiguration cost and energy aware vne policies in cycle-stationary traffic scenarios,” IEEE Journal on Selected Areas in Communications, 2016.
[8] S. Ghorbani, C. Schlesinger, M. Monaco, E. Keller, M. Caesar, V. Kuffner, and D. Walker, “Transparent, live migration of a software-defined network,” in the ACM Symposium on Cloud Computing, 2014.
[9] S. Lo, M. Ammar, E. Zegura, and M. Fayad, “Virtual network migration on real infrastructure: A planetlab case study,” in 2014 IFIP Networking.
[10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “Planetlab: an overlay testbed for broad-coverage services,” ACM SIGCOMM Computer Communication Review, 2003.
[11] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, “GENI: A federated testbed for innovative network experiments,” Computer Networks, 2014.
[12] Y. Zhao, S. Lo, E. Zegura, N. Riga, and M. Ammar, “Virtual network migration on the geni wide-area sdn-enabled infrastructure.” [Online]. Available: http://arxiv.org/abs/1701.01702v1
[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” ACM SIGOPS Operating Systems Review, 2003.
[14] “Pox wiki,” 2015. [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki

A. Interaction with the Substrate Network

a) Little knowledge about substrate networks: Under the current GENI context, we only have access to limited information about the substrate network such as the geographical information about GENI aggregates and VM load on each physical machine. Without sufficient real-time information about the physical substrate, it is difficult or impossible to implement an algorithm that has interaction with the substrate network. For example, some VN migration research may require real-time statistics about the substrate network to determine when to trigger the migration and where to migrate a VN. We expect GENI to expose more network statistics such as link utilization, throughput, and latency.

b) Difficulty in debugging: The virtualization techniques are deployed on GENI to support simultaneous experiments on limited physical infrastructure. However, virtualized architecture not only implies a trade-off between performance and isolation, but also makes debugging challenging. The virtualization architecture may bring unexpected problems, and the limited access to physical substrate further increases the difficulty in debugging. In our VN migration research, we had a hard time finding the cause of the duplicated packets when shared VLANs are used. We can only debug by observing the traffic in virtual topology and infer what is happening in the physical substrate. We expect GENI to develop efficient debugging tools to make the debugging process easier.

c) No control of substrate networks: We have flexibility to assign bandwidth to our virtual links in the reservation stage, but we cannot adjust parameters for the substrate network. Therefore, it is difficult to evaluate an algorithm with bandwidth constraints. This constraint makes it difficult to observe how dynamics in physical substrate such as changes in bandwidth or latency can affect the performance of a virtualized architecture.

B. Dynamic Resource Reservation

The GENI platform requires experimenters to reserve all resources on GENI slices before running their experiments. Most GENI resource does not provide flexibility to partially modify the resources. In our work, we take advantage of the shared VLAN feature to make resource reservation more dynamic. This resource reservation method requires the experimenters to consider which virtual links in the first slice should be converted to shared VLAN at the beginning when they