Enabling SDN Experimentation in Network Testbeds

Sivaramakrishnan S R
Jelena Mirkovic
USC
{satyaman, mirkovic}
@usc.edu

Pravein G Kannan
Chan Mun Choon
National University of Singapore
{pravein, chanmc}
@comp.nus.edu.sg

Keith Sklower
UC Berkeley
sklower@cs.berkeley.edu

ABSTRACT

Software-defined networking (SDN) has become a popular technology, being adopted in operational networks and being a hot research topic. Many network testbeds today are used to test new research solutions and would benefit from offering SDN experimentation capabilities to their users. Yet, exposing SDN to experimenters is challenging because experiments must be isolated from each other and limited switch resources must be shared fairly. We outline three different approaches for exposing SDN to experimenters while achieving isolation and fair sharing goals. These solutions use software implementation, shared hardware switches and smart network interface cards to implement SDN in testbeds. These approaches are under development on two operational SDN testbeds: the DeterLab at USC/ISI/Berkeley and the NCL testbed at the National University of Singapore.

Keywords

SDN; network testbeds; sharing

1. INTRODUCTION

Software Defined Networking is widely used in datacenter networks [1, 2] and at Internet exchange points [3]. A key component of SDN involves the separation of the data plane and the control plane. The control plane is shifted to a centralized controller, which is responsible for system configuration, management, and exchange of routing table information between switches.

SDN is a hot research topic today, drawing attention from top network, security and systems conferences and several newly-formed workshops. But today's SDN experimentation resources are very limited. Often, experiments with SDN use a single machine with virtualization of nodes and controllers [4]. Users are restricted by limited applications supported on virtualized nodes or by the number of possible nodes packed in a given system. Another common approach involves researchers purchasing one or a few SDN-enabled switches and configuring their own testbed in a lab. Yet, SDN switches are expensive and few researchers have necessary funds for a large testbed.

Today, there are many free, public research testbeds, used for experimentation in networks and distributed systems. These testbeds could be extended to support SDN-based experimentation, but this brings challenges, specifically related to sharing of SDN resources. In this paper we explore possible implementation techniques for SDN experimentation on testbeds. Mainly, we present three approaches – one software-based approach and two hardware-based – to implement SDN in testbeds. We consider various challenges faced while implementing some of these techniques on two operational testbeds – the DeterLab at USC/ISI/Berkeley and the NCL testbed at the National University of Singapore.

2. RESEARCH TESTBEDS

Research testbeds provide unique opportunities to researchers to access large-scale, high-end resources for free, thus promoting diversity and equal opportunity in networking and systems research. Many research testbeds exist today. Some of them are public – accessible by any researcher – like Emulab [5], DeterLab [6], Planetlab [7] and GENI [8]. Others are institutional or national testbeds, like the NCL testbed [9], accessible only within a given country or the hosting institution. While the primary goal of these testbeds is to support research, they are increasingly being used in education to provide practical experiences to students in areas such as networking, security and distributed systems.

We now provide more details about two operational testbeds that we are affiliated with: the DeterLab testbed and the NCL testbed. Both testbeds contain some number of user-accessible nodes that can run a custom OS. These nodes are connected to switches, which enables them to talk to each other. A user obtains exclusive access to multiple nodes, with superuser privileges, and can request the nodes to be connected into a custom topology. Multiple experiments and multiple users can share the testbed but each node is a part of only one experiment. Further, multiple experiments can use nodes connected to the same switch. Thus care must be taken to isolate experiments from each other, and to protect the switch from experimental traffic.

2.1 DeterLab Testbed

DeterLab [6] is a state-of-the-art scientific computing facility for cyber-security researchers engaged in research, de-

SDN-NFV Sec'17, March 22-24, 2017, Scottsdale, AZ, USA
© 2017 ACM. ISBN 978-1-4503-4908-6/17/03…$15.00
DOI: http://dx.doi.org/10.1145/3040992.3040996

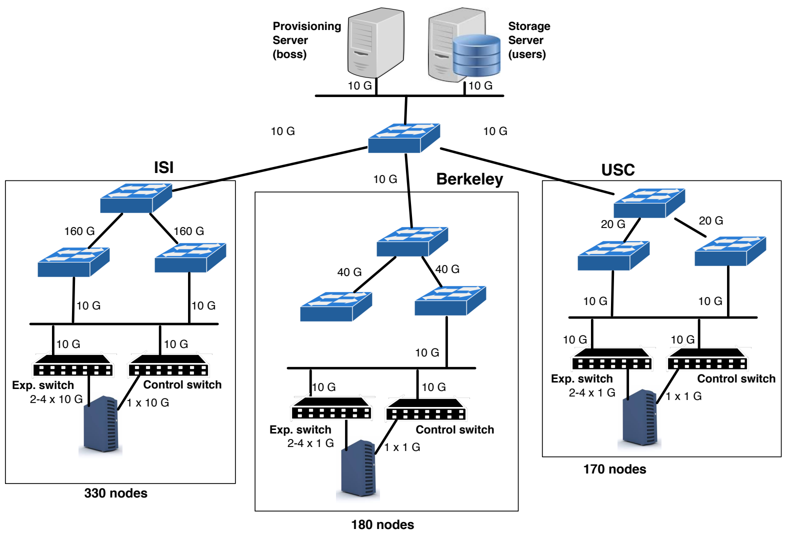

Figure 1: Architecture of the DeterLab Testbed

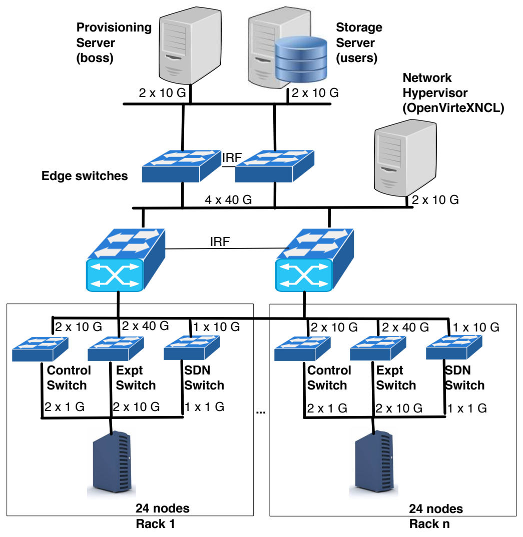

Figure 2: Architecture of the NCL Testbed

velopment, discovery, experimentation, and testing of innovative cyber-security technology. It is hosted at the The Information Sciences Institute, of the University of Southern California Viterbi School of Engineering (ISI) and University of California at Berkeley (Berkeley). In 12 years of its existence DeterLab has been used by 316 research projects, from 241 institutions (202 locations and 40 countries) and involving 862 researchers. Since 2009, DeterLab has been extensively used in education as well. It has been used by 157 classes, from 103 institutions and involving 9,893 users.

DeterLab consists of almost 700 nodes housed at three locations: USC, ISI and Berkeley. Each enclave connects to others via the public, 10 Gb Internet. A high-level architecture of the testbed is shown in Figure 1. There are two distinguished service nodes, called Boss and Users in the testbed, which offer provisioning and storage services, respectively. Each node has one control and several (2-4) network interfaces that connect to one or several of switches. Isolation between experiments at the experimental and the control network is achieved through VLAN-based isolation. Custom Controlnet-Isolation (CI) software runs on the boss to enforce this. The CI software ensures that each node can only send traffic to other nodes in the same experiment. However, traffic sent to the control network gets broadcast to the boss and users nodes and may overwhelm them or exhaust the share network's bandwidth.

2.2 NCL Testbed

The National Cybersecurity Research & Development Laboratory (NCL) [9] is a national shared infrastructure funded by the National Research Foundation (NRF) of Singapore that provides computing resources, repeatable and controllable experimentation environments, as well as application services.

In order to better support user experiments, NCL aims to provide tools/libraries for common vulnerabilities and exposure (CVE) environments, blockchain environment and algorithm validation and SDN provisioning. The NCL testbed provides an isolated environment for conducting security/ networking experiments. The experiment nodes are split into clusters of 24 nodes each. Each node is connected to three switches: (1) control switch, (2) experiment switch, and (3) SDN switch. The control switch connection enables loading of images and management of testbed experiments. The experiment and the SDN switches are used to build connections between nodes in an experimental topology. Isolation of the experiments is done using the VLAN mechanism for the control/ experiment network similar to DeterLab [6]. We explain the methodology of provisioning SDN experiments in Section 3.2.2.

3. SDN IN TESTBEDS

There are several methods of implementing SDN in testbeds. We look into some of these methods next.

3.1 Software SDN: Open vSwitch

A possible execution of SDN testbed is having dedicated nodes which run software switches such as Open vSwitch[10]. Here, the user's topology consists of physical nodes which are knit together with dedicated nodes running Open vSwitch.

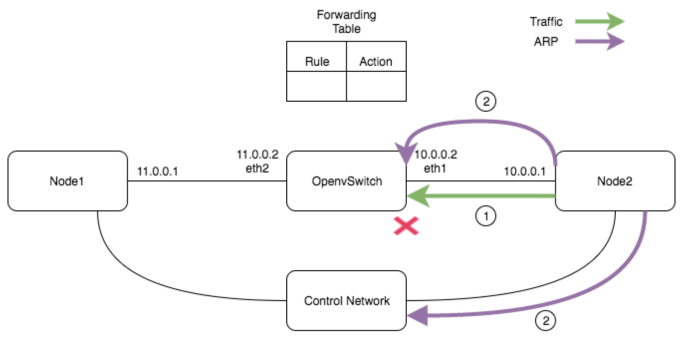

Figure 3: Node 2 sends traffic to Node 1 via an unconfigured Open vSwitch which drops all traffic causing ARP flooding to the control network of the testbed.

Since these are software switches running on the user's reserved nodes, the user has complete control over the switch and an exclusive access to it. The user can control the rule space on these switches using controller without affecting other experiments in the testbed.

Though this method isolates the experiment, incorrect implementation of Open vSwitches can affect the entire testbed. In Figure 3 two nodes are connected via an Open vSwitch. A naive user experimenting with this setup may not set the appropriate rules in Open vSwitch for connecting nodes over the experimental network. When such a user tries to reach *node2* from *node1*, Open vSwitch will send ARP packets on experimental interfaces. If such interfaces span two or more enclaves, the Internet link between them may get flooded. On the other hand, if an experimental node generates traffic for an IP that is not assigned to an experimental interface (e.g., a public IP), Open vSwitch will send ARP requests on the control network, flooding this shared resource.

Moreover, performance of software switches are limited. Software switches are designed to work on commodity machines which do not have the necessary hardware for fast packet processing and thus underperform at this operation, compared to a hardware switch.

3.2 Switches with SDN

Another approach to implement SDN in testbeds is to use hardware switches, which support SDN. These hardware switches are designed to process traffic at a high rate, and can offer high performance to users. Given that these hardware switches are capable of supporting multiple nodes at the same time, there may be multiple users using the switch at any given time. This leads to new challenges for resource allocation and isolation. We must ensure that users can only install SDN rules about nodes and links in their own experiment and may not observe nor control other experiments. We further must ensure that users cannot accidentally or maliciously affect the performance of the switch. Finally, we want to achieve fair sharing of limited switch resources – the rule space tables and the bandwidth. We now describe our initial work in this space, which we are exploring on the DeterLab and the NCL testbed.

3.2.1 DeterLab Approach

A primary component of approach taken at the DeterLab testbed is the SDN service, similar to the Controlnet-Isolator, which assists in setting up the SDN experiment and monitoring it afterwards. The SDN service runs on the boss node and proxies all OpenFlow requests. We now explain how creation, starting, and operation of an SDN-enabled experiment work in this paradigm.

Creating an Experiment: The user creates an experiment (SDN or regular) by specifying the required topology in an NS file(Network Simulator file). In the file every link between nodes is associated with a custom, user-chosen identifier. Each link is instantiated as one or several VLANs on one or several switches in the testbed. During experiment initialization, DeterLab will create a mapping between the link identifiers and the VLANs and store it in a database. SDN experiments can use these link identifiers in OpenFlow requests.

Starting the Experiment: After swapping in the experiment, the SDN service finds a suitable IP range available to assign to the experimental interfaces of the nodes and the switches involved in the experiment. Since switches are shared across different experiments, and since some OpenFlow rules may be IP-based, it is necessary that the nodes in one SDN experiments have unique IP addresses and subnets at the experimental network. The SDN service proceeds to create an OpenFlow instance on the switch associating all VLAN's belonging to the experiment with this instance. The SDN service then initializes a controller on an additional node in the experiment and binds the experiment's OpenFlow instance on each switch to the controller. The controller receives OpenFlow requests from the user and serves as an authenticator for the requests. Though the controller node is associated with the experiment, the user is not given access to the controller node. Isolation of the controller node and having dedicated IP ranges prevents experiments from affecting each other.

Authentication: Every request to the experiment's controller must be authenticated before accessing SDN switches in DeterLab. The experiment's controller authenticates requests generated by user using remote procedure call to the SDN service, which identifies the requesting user using his testbed certificate. This authentication check ensures that only authorized users can manipulate an experiment's settings. Next, the controller validates the user's OpenFlow message. The controller identifies the type of the OpenFlow message and checks if the request arguments (link identifiers, IP addresses) belong to the experiment. In case of messages, which modify the switch's rule space, the controller performs checks on available rule space quota for the experiment. If all the checks are passed successfully, the controller proceeds to replace the identifier in the request with the VLAN identifier. If more than one VLAN are associated with a given link, the message will be cloned once for each VLAN. Finally, the controller forwards the OpenFlow message to the SDN switch. Figure 4 illustrates an SDN experiment where the SDN Switch and two nodes are on the same VLAN. All OpenFlow messages to the controller are authenticated and verified for available rule space before being forwarded to the SDN Switch. The controller supports all Openflow 1.3 messages which are forwarded to it. Unfortunately, authentication and validity checks introduce overhead that would not exist if a switch were dedicated to a single user. This limits the performance tests that could be done with SDN, but it is necessary to ensure safety to DeterLab's experiments. Authentication checks could be performed once per

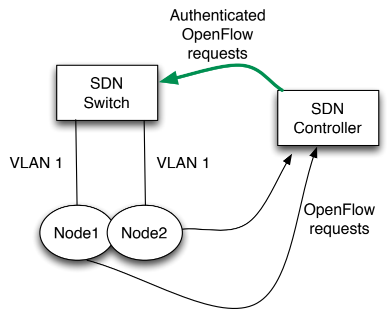

Figure 4: Testbed setup of an experiment sharing SDN switch outside the experiment space.

experiment and results cached for use in subsequent SDN requests. Validity checks require request parsing and database lookup, and introduce sub-second delays.

Allocation of Rule Space SDN switches, which are shared among experiments have limited rule space. Without adopting some means of fair allocation of rule space, experiments may become starved. There are three approaches which could be adopted to allocate rule spaces to experiments.

Fixed Allocation. Rule space on each switch can be allocated in fixed chunks to each SDN experiment allocated to the switch. We could project, based on usage history, the expected number of experiments per switch and use that to calculate the size of the fixed chunk. If we underestimate the number of active experiments, we could refuse to allocate further SDN experiments to the given switch. This policy would lead to some underutilization of rule space but would create stable conditions for users, where rules once inserted can only be deleted by the user.

Dynamic Allocation. Rule space on each switch can be equally divided between active SDN experiments on that switch. If a new experiment is swapped in, the quotas of the existing experiments would be reduced accordingly. This policy would lead to less amount of underutilized rule space but would create unforeseen and uncontrollable effects on the existing experiments. For example they could not predict the size of their quota, nor guarantee that rules they have inserted will persist in time.

On-demand Allocation. Some experiments may not use up the entire fair share of their rule space. It would be possible to design a fair-sharing approach similar to those in other resource allocation domains (e.g., fair sharing of router queues) but this would create poor user experience. For example, if an experiment A does not use its current rule space and it gets re-allocated to the experiment B, if A later wanted to insert some rules we would have to reclaim the space from B. This would potentially lead to deletion of rules that the user may not control, and is not user-friendly.

3.2.2 NCL Approach

In the NCL approach, we modify the topology specified in the NS file so that we can provision SDN resources flexibly, while still using the existing DeterLab's resource allocation algorithm [11]. SDN capability is provided to the users' experiments by choosing to connect them over a separate experimental network, through SDN-enabled switches, as shown in the Figure 2. The SDN switches are configured to interact with a Network Hypervisor (OpenVirteX [12]) which is connected to the core switches. We have modified the OpenVirtex to perform flow translation where we attach a TOS bit for each tenant's flow, and we call this new instance OpenVirteXNCL (OVXN). Since, we use OVXN, the users can use any SDN controller of their choice, and still not interfere with each other.

In order to leverage the current DeterLab's resource allocation [11] to provision both SDN experiments and normal experiments seamlessly, we model SDN switches in the following way:

1. Each SDN switch-port connected to host is defined as a special node called *ofedge*, and a switch-port that is connected to other switches is defined as *ofcore*. *ofedge* represent host-links and *ofcore* represent core-links in the topology.

2. The special node (ofedge/ofcore) consists of two virtual ports eth0 and eth1, where eth0 is the external port connected to the host/core, and eth1 is connected to the SDN switch.

3. When a user specifies an SDN switch in the topology, we automatically convert this specification into the format where we enumerate all the ports of the switch as either ofedge/ ofcore depending on the whether the port is connected to host/switch.

4. Also, another node is provisioned as the SDN controller of the experiment. The SDN controller will be directly connected to the shared OVXN node.

An example of a NS file showing two nodes connected by an SDN switch is shown below:

```
set n1 [$ns node ]
set n2 [$ns node ]
set switch1 [$ns node]
tb-set-hardware $switch1 ofswitch
set link1 [$ns duplex-link $switch1 $n1 ...]
set link2 [$ns duplex-link $switch1 $n2 ...]
```

The topology spec converted into the supported format, by our SDN-aware pre-processor looks as follows:

```
set n1 [$ns node ]
set n2 [$ns node ]
# Below Auto-generated
set switch1p1 [$ns node]
tb-set-hardware $switch1p1 ofedge
set switch1p2 [$ns node]
tb-set-hardware $switch1p2 ofedge
set ofswitchswitch2 [$ns make-lan
                    "$switch1p1 $switch1p2" ...]
set link1 [$ns duplex-link $switch1p1 $n1 ...]
set link2 [$ns duplex-link $switch1p2 $n2 ...]
set ctrl [$ns node]
tb-set-hardware $ctrl ofcontrol
set ovxn [$ns node]
tb-set-hardware $ovxn ovxctl
set ofc-ovxlink [$ns duplex-link $ctrl $ovxn ...]
```

In the original NS file (given by user), we identify an SDN switch using the hardware-type ”ofswitch”. Once a switch is identified as SDN switch, its ports are enumerated as switch1p1 and switch1p2. since switch1 has only two connections in the NS files. Since both the ports are connected to host, their type is ”ofedge”. Also, the host-links are changed to link between hosts and specific ports instead of the complete switch. Finally an SDN controller node is reserved, and it is connected to OVXN. We do not require that all the links in an experiment be either SDN-enabled or regular. It is possible to have a mixture of regular and SDN links in the same experiment.

Controller Isolation: Since OVXN (being the Network Hypervisor) is a shared node, which is directly connected to controllers in multiple SDN experiments, connection between the experiment's SDN controller and OVXN is made using VLAN in the experiment network (not SDN network). Hence, this connection is made using assignment of VLANs to the corresponding interfaces in the experiment network. The OVXN's interface to the experiment network is a trunk port which can simultaneously have virtual links with each SDN controller, which belong to different experiments. In order for OVXN to communicate with each SDN controller over a different VLAN, we implement a tagging mechanism using the following steps.

1. We setup an Open vSwitch [10] bridge at the external interface to OVXN, which connects to the experiment network.
2. The OVS bridge is configured with a flow to strip VLAN from any incoming packet before sending it to the host.
3. An agent in OVXN receives information (tuple) from boss (main provisioning server) <ControllerIP, ControllerMAC,VLAN> for each experiment during the creation.
4. Based on the tuple data, a flow rule is added to the OVS bridge to push vlan based on the destination IP. For example, if the SDN controller's IP is 10.1.1.1, and VLAN number is 15, the corresponding rule would be <Match:ipv4_src:”10.1.1.1”, Actions:VLAN:15, Output:1>.
5. A static ARP entry is added at the OVXN for the SDN controller to prevent arp broadcast.

In this way, isolation is guaranteed between individual SDN controllers with a shared node. Also, in terms of bandwidth, we have 2 X 10G interfaces towards the OVXN, which is sufficiently provisioned for multiple SDN controllers. We illustrate the SDN experiment isolation in Figure 5 where two identical SDN experiments specified in the above NS file (Expt1 and Expt2) are running. The OVXN connects to the controllers over a shared experiment network, however it tags the packets based on the controller that will be the packets' destination.

Rule-space Isolation: Since, we do not restrict the IP-address space, and also allow any OpenFlow rules to be specified by the experimenters, it can be challenging to truly isolate each experiment's packets as they share the same infrastructure (sometimes the same switch). To do this, we use the IP Type of Service (ToS) bit. We tag each packet with a specific ToS tag determined by the tenant-id (or experiment-id) based on its incoming port. Also, the user's rules are modified to contain the ToS bit in match, and action. However, when the packet leaves the network, ToS is stripped off.

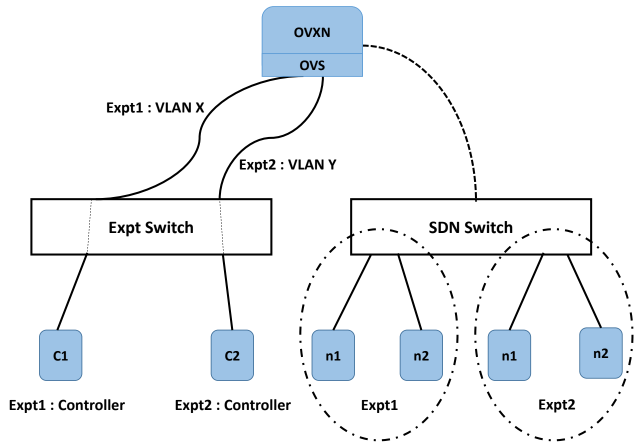

Figure 5: NCL SDN experiment setup to demonstrate controller isolation.

We use the ToS bit to perform isolation since the HP-3810 switches used in NCL infrastructure do not support VLAN strip action. However, in other supporting switches VLAN identifiers could be used for isolation. Our approach has the advantage that a user is allocated physical hardware resources and the experiment provides true performance test. Our current approach is limited in terms of its scalability in that the network topology that can be supported is limited by the actual (or subset of) physical topology. We are looking to improve the scalability of our approach to support any arbitrary topology in future work.

3.3 Smart NICs with SDN

Host-based networking includes implementation of network functions such as virtualization and switching to be operated on a standard commercial off the shelf hardware equipped with x86 CPUs. This type of deployment is common in the industry to enable significant efficiency gains, and gives complete control of the networking stack to the network operator.

To alleviate the burden on host system for packet processing, network functions such as switching can be offloaded to smart NICs. Smart NICs have architectures supporting flow and packet processing by having hardware-based accelerators to handle repetitive or specialized functions such as hashing and cryptography. Smart NICs are fully programmable, energy-efficient multi-core processors on which many packet processing functions including a full-blown software switch (e.g., Open vSwitch) can run.

We have purchased and installed several Netronome smart NICs on DeterLab, which run Open vSwitch and provide multiple virtual interfaces supporting multiple virtual machines. One can thus run many virtual machines on a node with a smart NIC running Open vSwitch and interconnect them into various topologies. One of the virtual machines can run the user's SDN controller, which can be connected to the virtual switch running in the smart NIC.

Similarly to the software-based Open vSwitch solution, this technique provides complete isolation to the user but has much better performance. However, the user is restricted on the number of virtual nodes and the topologies, which can be initialized on one physical node.

4. RELATED WORK

Testbeds other than DeterLab and the NCL testbed have provided SDN experimentation capabilities to users. The GENI [8] testbed's support is based on FlowVisor [13] slicing, where users of an experiment may opt-in to any slice of their choice. FlowVisor can compromise isolation in case of any misconfiguration of the slices. Also, the topology is fixed to the entire global physical topology (or a subset), and not to the experimenter's choice of any arbitrary topology. Hardware-based approaches adopted by our testbeds minimize the constraints on topology provided to the user.

Recently, other GENI-like testbeds started supporting SDN, like OFelia [14], ESNet [15], and Felix [16]. They give access to SDN experiments using switches where selective ports are controlled by OpenFlow and connected together to generate limited types of topologies. Also, they connect multiple sites using federation. CloudLab [17] provides a customizable SDN environment, using 100 G dedicated SDN-based inter-connects to communicate with other sites. Most of the testbeds are WAN network, which span across multiple sites. This may not be suitable for certain experiments, which try to emulate a data-center with low inter-node delay. Our solutions are more flexible and support a wider range of topologies. Further, current SDN solutions on testbeds do not address isolation and sharing issues, while our work focuses on these.

5. CONCLUSIONS

There are many advantages to supporting SDN on testbeds, but there are also notable challenges in implementing a high-performance solution that is also fair and secure. We have described three possible approaches for SDN support on testbeds and discussed their pros and cons. Our future work will be to implement and to evaluate these approaches in rigorous tests of performance, security and usability.

Acknowledgement. This research is supported by the National Research Foundation, Prime Minister's O ce, Singapore under its National Cybersecurity R&D Program (award No. NRF2015-NCR-NCR002-001) and administered by the National Cybersecurity R&D Directorate. This project is also the result of funding provided by the Science and Technology Directorate of the United States Department of Homeland Security under contract number D15PC00184. The views and conclusions contained herein are those of the authors and should not be interpreted necessarily representing the official policies or endorsements, either expressed or implied, of the Department of Homeland Security or the US Government

6. REFERENCES

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[2] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an sdn platform for cloud network services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, 2013.

[3] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: a software defined internet exchange," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 551–562, 2015.

[4] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 19, ACM, 2010.

[5] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, (Boston, MA), pp. 255–270, USENIX Association, Dec. 2002.

[6] R. Bajcsy, T. Benzel, M. Bishop, *et al.*, "Cyber Defense Technology Networking and Evaluation," *Commun. ACM*, vol. 47, no. 3, 2004.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: An overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 3–12, July 2003.

[8] 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*, vol. 61, pp. 5 – 23, 2014.

[9] "National Security R & D Laboratories." http://ncl.sg.

[10] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer.," in *Hotnets*, 2009.

[11] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 65–81, Apr. 2003.

[12] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: Make your virtual sdns programmable," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), pp. 25–30, ACM, 2014.

[13] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?," in *Proceedings of the 9th OSDI*, (Berkeley, CA, USA), pp. 365–378, USENIX Association, 2010.

[14] M. S. et al., "Design and implementation of the {OFELIA} {FP7} facility: The european openflow testbed," *Computer Networks*, vol. 61, pp. 132 – 150, 2014.

[15] "ESNet." http://www.es.net/network-r-and-d/experimental-network-testbeds/100g-sdn-testbed/.

[16] G. Carrozzo, R. Monno, B. Belter, R. Krzywania, K. Pentikousis, M. Broadbent, T. Kudoh, A. Takefusa, A. Vieo-Oton, C. Fernandez, *et al.*, "Large-scale SDN experiments in federated environments," in *Smart Communications in Network Technologies (SaCoNeT) Conference*, pp. 1–6, IEEE, 2014.

[17] R. Ricci and E. e. a. Eide, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *;login: the magazine of USENIX & SAGE*, vol. 39, no. 6, pp. 36–38, 2014.