OpenVirteX: Make Your Virtual SDNs Programmable

Ali Al-Shabibi, Marc De Leenheer,
Ayaka Koshibe, Guru Parulkar
and Bill Snow
Open Networking Laboratory
Menlo Park, CA 94025, US
{ali,marc,ayaka,guru,bill}@onlab.us

Matteo Gerola and Elio Salvadori
CREATE-NET
Povo, 38123 TN, Italy
{mgerola,esalvadori}@create-net.org

ABSTRACT

We present OpenVirteX, a network virtualization platform that enables operators to create and manage virtual Software Defined Networks (vSDNs). Tenants are free to specify the topology and addressing scheme of their vSDN, and run their own Network Operating System (NOS) to control it. Since OpenVirteX logically decouples vSDNs from the infrastructure, it also enables the introduction of features such as link and switch resiliency, and network snapshotting and migration of these tenant networks. OpenVirteX builds on the design of FlowVisor, and functions as an OpenFlow controller proxy between an operator's network and the tenants' network OSes. Our evaluations of this implementation show that i) OpenVirteX is capable of presenting tenants with configurable vSDNs while incurring a modest overhead to the control channel, and ii) that our architecture enables the introduction of features and enhancements such as link resilience to tenant networks.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.3 [**Computer-Communication Networks**]: Network Operations

Keywords

OpenFlow; Software-Defined Network; Network Virtualization; Virtual Links; Resiliency; Address Virtualization; Topology Virtualization; Control Function Virtualization

1. INTRODUCTION

Network virtualization has emerged as one of the key capabilities of our computing and networking infrastructure. Many approaches are being actively pursued with different tradeoffs. In this paper, we present OpenVirteX (OVX), which delivers Infrastructure as a Service (IaaS) in the context of Software Defined Networks. It is highly desirable to decouple the network from its physical manifestation to provide virtualized network resources. These virtual networks can offer strong isolation and have the ability to migrate, snapshot, and to customise topology at instantiation time. Such virtual networks could be instantiated along with the compute resources to deliver true infrastructure on-demand.

Infrastructure providers are increasingly looking towards network virtualization using SDN to better utilise their networking resources. In particular, network virtualization enabled by SDN allows infrastructure owners to lower the management complexity of their networks, while customising their tenants' network to better serve their needs.

Network Virtualization provides the concept of a virtual network which is decoupled from the underlying physical infrastructure, and both physical and virtual networks can continue to use existing abstractions and protocols such as TCP/IP. However network virtualization allows new features or operations. For example, virtual networks can be treated as services that can be instantiated, migrated, and deleted. We created OVX as a network virtualization platform that can i) provide address virtualization to keep tenant traffic separate, ii) provide topology virtualization to enable tenants to specify their topology, and iii) deliver each virtual network to the tenants' NOS as Infrastructure on-demand.

We present OVX as a network hypervisor that enables operators to provide this form of network virtualization to their customers. In specific, OVX builds on OpenFlow [13], and our experiences with FlowVisor [15]. Like FlowVisor, OVX functions as a proxy within the control channel, presenting OpenFlow networks to tenants, while controlling the underlying physical infrastructure via the southbound OpenFlow interface as shown in Figure 1.

By exposing OpenFlow networks, OpenVirteX allows tenants to use their own NOS to control the network resources corresponding to their virtual network. In other words, OpenVirteX creates multiple virtual software defined networks out of one. Unlike FlowVisor, which simply slices the entire flowspace amongst the tenants, OVX provides each tenant with a fully virtualized network featuring a tenant-specified topology and a full header space.

The rest of this paper is organized as follows. Section 2 presents our motivation for developing OVX, described with respect to Network Function Virtualization and cloud computing, while Section 3 presents related work. Section 4 discusses the architecture and core virtualization abilities of OVX. The following Section 5 describes the main features exposed by OVX. Section 6 presents preliminary per-

HotSDN'14, August 22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2620728.2620741 .

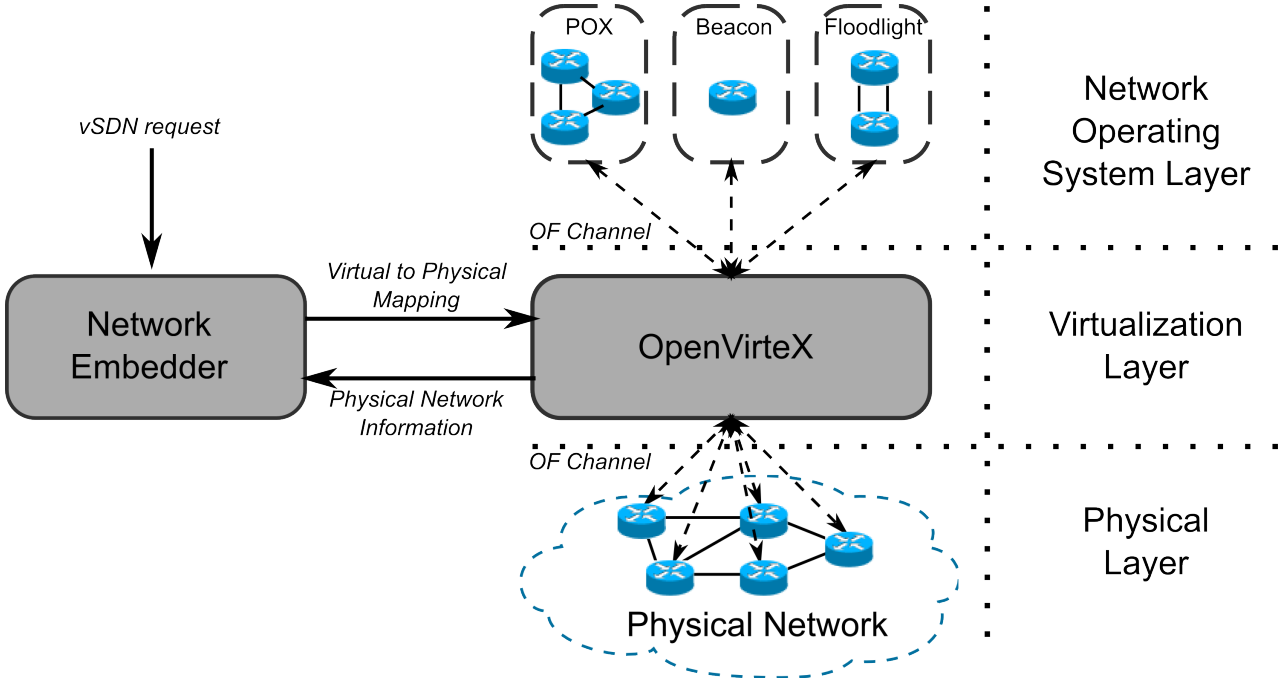

Figure 1: System architecture.

formance results. Finally, Section 7 closes the paper with discussions of future work.

2. USE CASES

Nowadays network functions, such as firewall, DPI, load-balancing and authentication, are implemented in dedicated hardware appliances. Network Function Virtualization(NFV) [7] [1] is a new and promising approach to offering such network functions in software/VMs to reduce capex and opex and allow more flexibility. We believe network functions will be implemented as scalable elastic services realized using a virtual network of VMs. As such, NFV will require dynamic virtual networks management. OVX aims to provide such an unified service orchestration layer.

Another key motivation for OVX is to facilitate transition to the cloud. Consider the scenario where an enterprise has constructed a large network service complete with a management and monitoring solution, and is now considering moving its physical infrastructure to the cloud. Porting its management and monitoring solution to fit the resources available in the cloud may be cost-prohibitive for the enterprise. Ideally, the enterprise would be able to create or instantiate a virtual network that is identical to the physical network that it has built in-house. This requirement is at the core of OVX's philosophy. Indeed, the ability to instantiate virtual networks of arbitrary topology as well as provide custom addressing is the main requirement around which OVX was built.

3. PREVIOUS WORK

FlowVisor [15] is a software platform for slicing an OpenFlow network into multiple resource pools, or slices. Each slice has its own NOS and is associated with a subset of the network resources encompassing some or all of the switches and links. Network slicing with FlowVisor does have several limitations. All slices essentially share the same flow or address space, and thus a slice does not have a completely separate and independent address space. FlowVisor also does not allow a slice to have an arbitrary (virtual) network topology; it can only offer (a subset of) the physical topology. Additionally, FlowVisor provides isolation amongst the slices that it controls by allocating non-overlapping sections of the packet header space to each slice's flow space. Unfortunately, misconfiguration can cause the flow spaces to overlap, in which case FlowVisor is unable to provide isolation for affected slices.

VeRTIGO [9] is an extension to FlowVisor which provides topology virtualization. It allows the tenant to specify virtual links in the network slice. Since Vertigo is based on FlowVisor it is unable to provide each tenant with a full isolated address space.

Another approach to network virtualization is described in FlowN [10]. This platform leverages a database storage system to maintain a mapping between the physical and virtual realms. However, since each tenant's logic must be embedded in in the FlowN controller, a tenant is constrained to developing their application with the FlowN framework. OVX allows the tenant to use his own NOS by exposing an OpenFlow interface northbound. More importantly, FlowN encapsulates each tenants' traffic into a VLAN to achieve address space isolation, thereby removing the VLAN header from the headerspace available to the tenant.

4. ARCHITECTURE

OVX is a network virtualization platform capable of spawning virtual networks with OpenFlow semantics. These virtual networks may have arbitrary topology and addressing schemes, configured as per tenant request. Requests are con-

veyed via API calls to OVX, with a tool such as a network embedder. Figure 1 shows how a network embedder and OVX would work in concert to realise a requested virtual network. First, a user specifies a virtual network's addressing scheme, topology, and NOS link (e.g. vSDN request) to the embedder, which generates a virtual-to-physical mapping using information from OVX. Next, this mapping is passed to OVX, which in turn instantiates the virtual network on the physical topology.

The decision to separate the problem of embedding the virtual network onto the physical one from that of network virtualization was made early on for the following reasons:

1. There has been significant research in the area of network embedding algorithms (such as D-Vine, R-Vine [8] and VTPlanner [14]) which we can leverage, rather than try to add to this field;

2. We want OVX to have the ability to run in a standalone manner; and

3. We want our network hypervisor to be completely focused on efficient and correct network virtualization.

4.1 Internal OVX Architecture

Internally, OVX relies on a loose decoupling of virtual elements from the physical counterparts. In order to achieve this, OVX models all virtual and physical elements and maintains the mapping between them as shown in Figure 2. This mapping is provisioned by the embedder. The area in Figure 2 above the dotted line deals only in virtual terms while the area below deals only in physical terms. All virtual network elements are mapped to at least one physical element. The mapping itself does not specify how the virtualization is actually implemented. In fact, each element implements its virtualization how it desires, i.e. a virtual link may use MPLS tags or MAC rewriting as we have currently implemented. As each virtual element is simply a pointer onto some real network element; we can disable, enable, modify and/or reorganize virtual network elements at runtime.

4.2 Topology Virtualization

As mentioned earlier, OVX allows the tenant or user of the system to specify his own arbitrary topology. This topology could be simple, i.e., a big switch, or a more elaborate network with many alternative paths for fault tolerance. In essence, these topologies do not have to correspond to the actual physical network, but rather, exactly match what the tenant desires. By design, the only limitation enforced by OVX is that a single physical switch cannot be partitioned into multiple virtual switches. To expose a virtual topology, OVX resolves LLDP messages coming from the NOS. In particular, when an LLDP message arrives at a virtual switch element with a certain outport specified in its body, OVX "knows" where the other end of that link is. Therefore, OVX forges an LLDP "response" packet and sends it back to the NOS, thereby creating the illusion of a link at the NOS. This LLDP mechanism is beneficial for the following reasons:

1. The number of LLDP packets travelling in the data plane remains constant no matter how many virtual networks exist, and

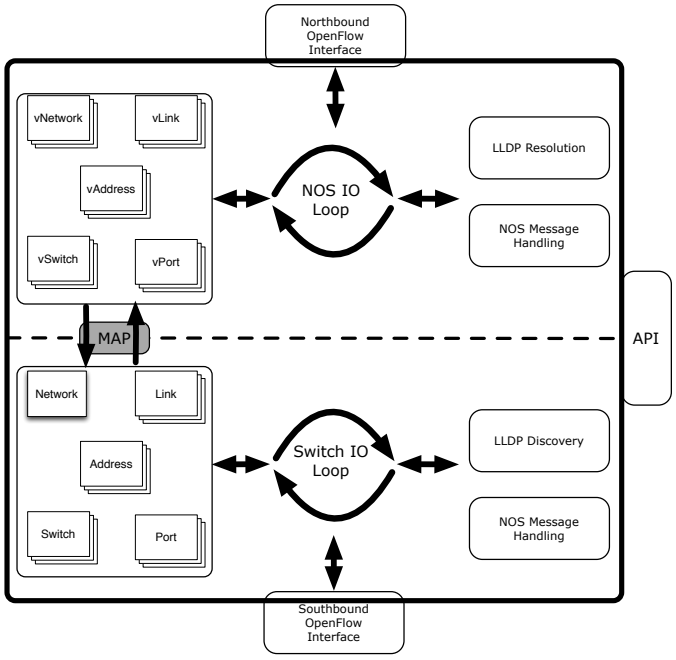

Figure 2: Internal OVX architecture.

2. Enables OVX to expose any virtual topology to the NOS.

4.3 Address Virtualization

OVX grants tenants the ability to choose address assignments for their end hosts, allowing multiple, potentially overlapping IP address blocks to exist in the same physical network. To differentiate hosts, OVX generates globally unique tenant IDs for each tenant, and for each host, a physical address that encodes the host's membership using the tenant ID. The physical address used is a combination of MAC and IP headers. We use IP rewriting when the NOS pushes layer 3 rules and MAC rewriting when the NOS pushes layer 2 rules. This is done for the following reasons:

1. We do not want to modify the semantics of the rules pushes by NOSes, and

2. We want to support both layer 2 and 3 virtual networks.

Collisions of addresses are avoided by installing flow rules to rewrite addresses at the edge switches of the network, from tenant-assigned address to physical IP address at the ingress edge, and vice-versa at the egress edge.

As OVX rewrites packets at the edge of the network, it imposes a negligible overhead on the dataplane. Evidently, we cannot support the scenario where all tenants want to use the entire IP or MAC space since we reserve some bits in those headers to encode the tenant identifier. The translation process is illustrated in Figure 3.

Importantly, this procedure is invisible to a tenant's NOS or hosts, implying that a NOS used to control a virtual network created by OVX doesn't need to be modified in any way to properly function. In addition to preventing address aliasing in a transparent way, the mapping created by the

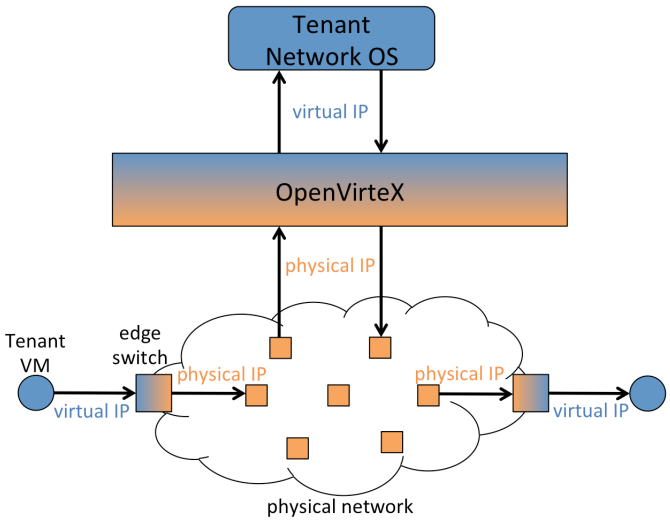

Figure 3: The IP translation enforced by OVX happens into the edge switches for data packets and between OVX and the NOS for control packets.

procedure is used by OVX to demultiplex northbound control messages so that they reach the correct tenant networks.

4.4 Control Function Virtualization

We enable each virtual network to have its own NOS and applications that can program the virtual network switches. OVX is responsible for mapping various control functions for the virtual network on to the corresponding physical network. In some cases OVX would translate a relatively simple control operation on a virtual network into multiple actions on the physical control plane. OVX can do this by leveraging its advantageous position in the control network, i.e., it can intercept control packets, and therefore multiplex different control planes onto one. This gives the "illusion" that every NOS is the only OS running in the system.

5. FEATURES

The design decisions in OVX are intended to easily provide extensibility. The loose coupling of virtual and physical components in the form of N:1 key-value mappings, introduces significant amounts of flexibility. The liberties that can be taken in how virtual components can be mapped to physical counterparts is key to a few core vSDN features:

- *Topology customization:* vSDN topologies need not be isomorphic to the infrastructure, or be restricted to its subgraph. A virtual link may encompass multiple contiguous hops, and virtual switches may abstract away parts or all of a network.
- *Resiliency:* A virtual link or switch can be mapped onto multiple physical components to provide redundancy. A resilient virtual link is characterized by multiple physical paths between the points corresponding to the virtual link endpoints. A virtual switch that abstracts away portions of the physical network may take advantage of redundancies in the topology (e.g. multiple paths) to provide multiple paths between its ports.
- *Dynamic vSDN reconfiguration:* The reconfiguration of a tenant network is reduced the manipulation of key-value pairs. This is a simple operation since the mapping is logically centralized in the global map, and key-value pairs employ references as opposed to storing the component representations. Since the mappings themselves do not store any network state, these can be changed at runtime.

  In addition, it is implied that vSDNs can be made portable across different operator networks given that they i) support the same control protocol as OVX, and ii) the vSDN topology can be mapped onto the network without partitioning switches.

5.1 Persistence

Each network and network element maintains a collection of preservable attributes. This enables persistence by allowing OVX to record and store this information in persistent storage, so that vSDNs can be torn down and re-created at a later time. In addition to giving OVX the ability to "save" and "recover" vSDN configurations, persistence serves as a first step towards the ability to snapshot tenant networks.

5.2 Troubleshooting

The extensive rewriting of control messages complicates the already difficult process of troubleshooting networks. However, the central position of OVX in the control channel enables it to integrate with network debuggers such as NetSight [11] by providing connections into vSDNs and instantiating special-purpose ports on virtual switches. These ports can be used to mirror control messages directly to NetSight or a similar tool, and as with other virtual constructs provided by OVX, can be dynamically created and configured.

6. RESULTS

We present only some preliminary results for our solution, tested in a controlled environment. The tests discussed here focus on control channel overhead introduced by OVX, and virtual network generation time. To measure overhead, we used cbench [2], a control plane benchmarking tool. In this test, cbench was configured to spawn a number of switches equal to the number of virtual networks, each switch having five hosts with unique MAC addresses. Every virtual network is an exact copy of the physical one, in order to have a clear comparison with FlowVisor and FlowN. The tests have been run on a commodity laptop equipped with 2.3GHz quad-core Intel Core i7 (turbo boost up to 3.5GHz) with 16GB 1600MHz memory and 512GB PCI-e based flash storage.

Figure 4 shows the latency introduced by OVX, FlowVisor, and FlowN, and the reference case where no virtualization software is used and the packets are sent directly from the switch to the controller (Floodlight). The results show that, compared with other platforms, OVX offers better performance than similar virtualization platforms and only adds around 0.2 ms latency to the control channel. Importantly, the delay added is independent of the size of the physical network and the number of virtual networks.

Table 1 shows the time needed to create and configure a virtual network up to the point where the network OS has connected. The physical network is the Internet2 NDDI

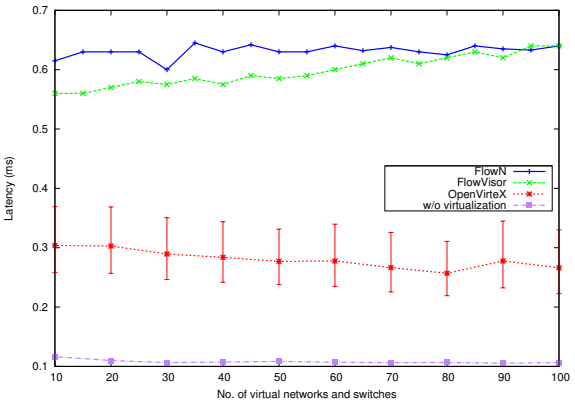

Figure 4: Latency comparison of OVX, FlowVisor, FlowN and Floodlight.

No. of switches	Physical clone		Big switch	
	API	DB	API	DB
21	1.363	.761	.418	.301
53	3.679	1.908	.787	.459
121	10.099	4.18	.847	1.078

Table 1: Virtual network provisioning time in seconds for various sizes of the physical topology. Virtual networks can be created through the API or loaded from persistent storage (DB).

topology, which has 11 core switches spread across the US and fat tree-based datacenter networks (with varying number of switches) attached to the cores. Rapid creation of virtual networks is required to support IaaS environments, which OVX clearly can offer with provisioning times in the order of several (tens of) seconds. Also note how the creation time of a big switch virtual network encompassing the complete physical network does not depend on the size of the physical topology. Contrast this to the virtual clone of the physical network, where we have to recreate the one-to-one mapping from virtual to physical for each individual network element.

Finally, we have deployed OVX on Internet 2 (I2), a nation-wide research network located in the US, to demonstrate its capabilities in a live physical WAN. We run two vSDNs, one a copy of the I2 topology and the other, a single giant switch. Each vSDN was controlled by its own NOS, ONOS [4] and Floodlight, respectively, which ran unique functions that applied accross the I2 WAN.

7. CONCLUSION & FUTURE WORK

In this paper, we presented OVX which provides programmable vSDNs. These vSDNs are customizable in terms of topology and addressing scheme used, and each tenant can control his virtual SDN with the NOS of his choice.

Although the current version of OVX presents all the core features expected from a network virtualization platform, we are currently designing three additional functionalities that will improve the usability and the performance of the tool. In particular, we are focusing on:

- *Snapshotting and migration of vSDNs:* in a virtualization environment, the ability to preserve the state and data of a Virtual Machine (VM) at a specific point in time has become a key functionality. This allows not only fast recovery in case of failure, but also eases the migration and duplication of a working VM in other locations [12]. The information stored in OVX allows to reproduce this feature into the network infrastructure, not only saving the configuration of the vSDNs but also the status of the data plane, e.g., all the OpenFlow rules active in all the devices at a certain point of time, thus allowing the tenant to step back to a previous working state and/or to migrate the vSDN to another location without reconfiguring it.

- *Evolving beyond OF1.0:* we are replacing the standard java openflowj library, strictly tied to OF1.0, with LOXI [5]. LOXI is a marshalling and unmarshalling engine that generates version-agnostic OpenFlow libraries in multiple languages. Leveraging on the clear separation between virtual and physical environment implemented in OVX, we plan to implement OF1.3 in the southbound interface, while exposing OF1.x to the tenants, thus allowing common open source OF1.0 controllers like NOX and Floodlight to run on an OF1.3 network.

- *vSDN-based Quality of Service:* the only solution to limit bandwidth usage for a specific vSDN in OF1.0 is to statically assign a specific port queue to the virtual network. This approach is not feasible in a real network, since the configuration of minimum and maximum bandwidth per queue is static and vendor dependent. However, once the migration to LOXI, and thus to OF1.3, will be completed, OVX will use flow-based (e.g. vSDN-based) meters to enforce QoS for the virtual networks, thus offering to the tenants a fully isolated environment with performance guarantees.

We believe OVX has a promising future in the network virtualization space. To maximize its impact, we will release OVX, along with the network embedder, as open source software in the foreseeable future. Furthermore, work is in progress to integrate OVX with OpenStack [6], and create a powerful IaaS platform that can orchestrate compute, storage and network resources. This will finally lead to deployment of OVX in OpenCloud [1], where it will be a key building block to realize NFV functionality at scale.

8. REFERENCES

[1] OpenCloud website - http://www.opencloud.us.
[2] Oflops website - http://www.openflow.org/wk/index.php/Oflops.
[3] Floodlight website - http://www.projectfloodlight.org/floodlight/.
[4] ONOS website - http://tools.onlab.us/onos.html.
[5] LOXI website - https://github.com/floodlight/loxigen/wiki/OpenFlowJ-Loxi.
[6] OpenStack website - https://www.openstack.org.

[7] M. Chiosi et. al. Network functions virtualisation. Technical report, ETSI, October 2012.

[8] M. Chowdhury, M. R. Rahman, and R. Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking*, 20(1):206–219, February 2012.

[9] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori. Vertigo: Network virtualization and beyond. In *EWSDN 2012*, October 2012.

[10] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. In *IEEE Internet Computing*, March/April 2013.

[11] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 71–85, Seattle, WA, 2014. USENIX.

[12] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford. Live migration of an entire network (and its hosts). In *11th ACM Workshop on Hot Topics in Networks*, 2012.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 32(2):69–74, April 2008.

[14] R. Riggio, F. De Pellegrini, E. Savadori, M. Gerola, and R. Doriguzzi Corin. Progressive virtual topology embedding in openflow networks. In *The Fifth IFIP/IEEE International Workshop on Management of the Future Internet (ManFI)*, 2013.

[15] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Operating Systems Design and Implementation*, October 2010.