HybNET: Network Manager for a Hybrid Network Infrastructure

Hui Lu[∓*], Nipun Arora[†], Hui Zhang[†], Cristian Lumezanu[†], Junghwan Rhee[†], Guofei Jiang[†]
[∓]Purdue University [†]NEC Laboratories America
lu220@purdue.edu {nipun,huizhang,lume,rhee,gfj}@nec-labs.com

ABSTRACT

The emergence of Software-Defined Networking(SDN) has led to a paradigm shift in network management. SDN has the capability to provide clear and easy management of complex operational challenges in large scale networks. However, most of the existing work in SDN network management assumes a full deployment of SDN enabled network switches. Due to both practical and financial limitation real implementations are likely to transition through a partial deployment. In this paper, we describe our experience in the design of HybNET a framework for automated network management of a hybrid network infrastructure (both SDN and legacy network infrastructure). We discuss some of the challenges we encountered, and provide a best-effort solution in providing compatibility between legacy and SDN switches while retaining some of the advantages and flexibility of SDN enabled switches. We have tested our tool on small hybrid network infrastructure, and applied it to manage the OpenStack Neutron interface a well known open-source IaaS provider.

Categories and Subject Descriptors

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

General Terms

Design, Management

Keywords

Network architecture, Cloud, Virtualization, SDN

1. INTRODUCTION

Software-defined networking (SDN) enables efficient and expressive network management in enterprises and data centers by (1) separating the high-level network policy specification (*e.g.*, for isolation, security, failure recovery) from its low-level implementation on network devices (*e.g.*, forwarding rules), and (2) allowing fine-grained visibility and *almost instant* configuration updates of the network. Network operators realize SDN by deploying switches whose forwarding table is programmable remotely from a logically centralized controller using a specialized protocol (*e.g.*, OpenFlow [22]).

The majority of OpenFlow-enabled networks deployed in practice, either in academic test-beds [20, 16, 14, 23] or part of large data centers [19, 17], assume a fully evolved SDN ecosystem, where all legacy network configuration has migrated to OpenFlow. In reality, however, the transition from the legacy network to an OpenFlow-enabled network does not happen overnight. Due to both practical and financial limitations the network is likely to transition through a *hybrid* deployment with both legacy and OpenFlow switches and with the same high-level policy implemented through different low-level mechanisms. For example, ensuring communication isolation among a set of hosts is likely implemented with VLANs on legacy switches and fine-grained forwarding rules on OpenFlow switches. The challenge, identified by others as well [22, 21, 15] is to ensure seamless network operation, even when the configuration mechanisms of the network or not compatible.

We present HybNET, a network management framework for *hybrid* OpenFlow-legacy networks. HybNET provides the network operator centralized control and visualization across the whole network, similar to how a controller in a pure OpenFlow network would. HybNET hides the dissonance between the SDN and legacy network configurations by transparently translating the legacy network configuration into OpenFlow configuration and providing a common configuration mechanism for both SDN and legacy switch configuration. In essence, our framework views the connections between OpenFlow switches as *virtual links* composed by multiple links between physical switches. Thus, OpenFlow switches deal with the task of intelligently applying the configuration, while legacy switches are limited to providing forwarding.

The contribution of this paper is two-fold. Firstly, we provide a basic design of a hybrid network manager which allows for seamless connectivity in the network via a common centralized configuration interface. Secondly, we provide a mechanism for network virtualization functionality for a set of hosts. This functionality is provided using VLANs in traditional networks and using fine-grained forwarding rules

*This work was done when the author was an intern at NEC Laboratories America

inserted by the OpenFlow controller. HybNET provides a proxy service that translates the VLAN configuration into OpenFlow rules on the fly by .

We have implemented the automation management prototype on top of the popular open source cloud computing platform, OpenStack [8]. We adopt Neutron [3], the network service of OpenStack, for host-side network virtualization and construct the hybrid physical infrastructure by using legacy switches as well as SDN switches (supporting OpenFlow). The cloud infrastructure was hosted a mininetwork infrastructure, with both SDN and legacy switches. HybNET was successfully able to manage, add, delete and modify virtual tenant networks for users and network operators, and provided network isolation for all tenants. We believe that HybNET is the first of it's kind in providing a practical management mechanism for hybrid networks, and can be easily used by network operators without any special hardware requirements.

The rest of the paper is organized as follows: Section 2 introduces the design of HybNET. Section 3 describes implementation of our prototype. Section 4 shows the evaluation in terms of infrastructure performance. Section 5 discusses the related work. Finally, in section 6, we summarize the work.

2. DESIGN

2.1 Architecture Overview

As shown in Figure 1, our design consists primarily of three components: physical infrastructure, path finder, and controller (described in section 2.2). The physical infrastructure describes the basic mechanism that HybNET connects to the network infrastructure, and the path-finder provides a basic algorithms to find viable paths in the network for connectivity. The main component, and work-flow is a part of the controller which manages and orchestrates the entire management framework.

Each network operator request is dealt as a *transaction* and is logically atomic in nature. HybNET checks if the rules have been updated successfully, and reports any errors while maintaining state consistency at all times. Additionally, HybNET supplies a common API for network operators to use to process transactions. For each transaction, configurations are prepared and sent to legacy switches or SDN controllers through remote communication mechanism, such as RPC and REST calls, and a persistent state view is kept in the mapping database.

2.2 Components

Physical infrastructure/topology: A Hybrid Network is composed of both legacy and SDN switches, which can be fabricated into various topologies based on requirements [15, 21]. However, to provide a generic framework, HybNET does not assume any underlying topology. At the same time, we assume that we require complete knowledge of the physical topology, which can be provided either by the network administrator or by an automatic network discovery service (*e.g.*Link Layer Discovery Protocol (LLDP)[2]).

Additionally, similar to OpenFlow controllers HybNET requires management access to control every switch in the

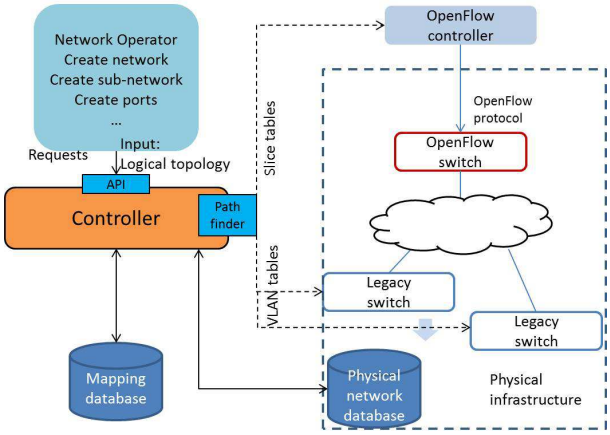

Figure 1: Hybrid-controller architecture overview

network. Access can be implemented either in-band (management traffic share the network with data traffic) or outband (management traffic is designated to separate physical network). All network requests are communicated to the HybNET controller, which computes the network operations and segregates the updates that need to be performed in the legacy part of the network and the OpenFlow part of the network. The resulting changes are then disseminated to the OpenFlow controller (which then manages all SDN switches), and the relevant legacy switches. HybNET talks to OpenFlow controller via REST calls and subsequently the controller manages OpenFlow switches via OpenFlow protocols. While the management of the OpenFlow controller is standardized, legacy switches have several network management mechanisms, such as SNMP[9] and NETCONF[18]. For the purposes of this paper, we have demonstrated our proof of concept tool by using the NETCONF protocol (we believe that other mechanisms can also be adapted to such a management framework). Provided aforementioned communication protocols, HybNET will be able to control every switch in hybrid network.

Path finder: As stated earlier the primary goal of any network management solution is to provide end-to-end connectivity. To this end, HybNET offers path finding algorithms to calculate suitable paths to connect end-hosts whose logical topologies belong to the same network. Figure 2 shows an example of a cloud infrastructure which wants to provision two VMs, one is on Host 1 and the other is on Host 4 on the same logical network and ensure connectivity between them. Viable paths between them can go through both legacy and SDN switches, hence existing tools cannot be directly applied. The path-finder can find a suitable path along the red line as shown in Figure 2 by using an appropriate algorithm. Once calculated this path is used to identify the relevant switches that need to be configured, and this information is forwarded to the HybNET controller.

Within enterprise network or data-center networks, network topologies can be much more complex. Thus, intelligent path finding algorithms [12, 13, 24] are required to manage aspects such as efficiency, isolation, load balancing, QoS, etc. However, for the purposes of this paper, our implementation

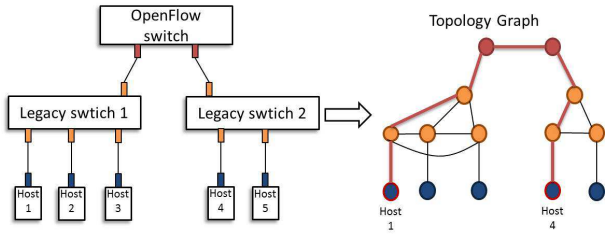

Figure 2: Topology graph generation

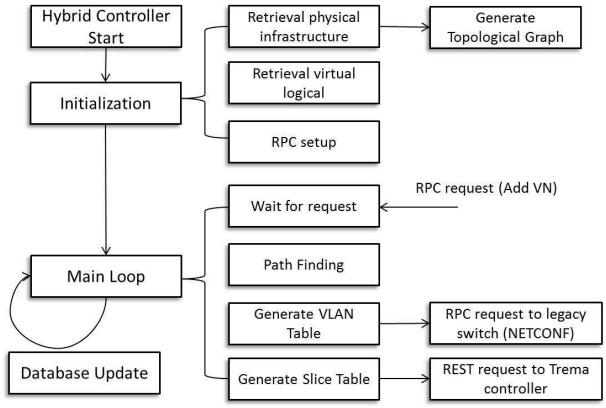

Figure 3: Hybrid-controller working flow

on adopts a simple shortest path algorithm.

Controller: The controller is the main component of HybNET and manages most of the logic. The work-flow (see Figure 3) of the controller can be divided into two phases: an initialization phase, and a main loop which manages everything at run-time. In the initialization stage, controller retrieves physical topology from the network and generates a topological graph. Additionally, it registers and sets up RPC callback functions so that they can receive and process network operator requests.

The main loop function of controller, manages all run-time network requests. For example, when the user requires to provision a new VM, a logical network is created from the host side. Then an *"adding"* request with the requisite information is sent to the controller by the network operator with help of HybNET supplied APIs. After receiving this request, controller will parse input parameters to check tenant id, user id, logical network information (i.e., VLAN configurations), and VM information (i.e., host machines). Then path finding algorithm is applied to detect available paths. If the path does not exist, an unreachable error will occur. We mark this transaction as *"failed"* and send back an error message.

If paths are generated successfully, HybNET prepares configuration files for legacy switches and OpenFlow switches along the path. Our design provides general APIs to allow developers or switch vendors to design drivers to talk to legacy switches or applications, while control of OpenFlow switches is designated to standard OpenFlow controllers. If the return status of each switch after configuration shows SUCCESS, the desired connection is established, otherwise a configuration error will occur. We mark this transaction *"failed"* and all switches are rolled back to previous configuration state.

Another important aspect is that any changes made to the physical infrastructure need to be reported to HybNET. If physical topology changes, the database is updated correspondingly, and if required the changes are made updating rules on corresponding switches.

2.3 Virtual links

Network virtualization is the process of combining hardware and software network functionality into a single software-based administrative entity called a virtual network. One of the key features offered by network virtualization is the logical separation of systems physically attached to the same local network into different virtual networks. Grouping hosts with a common set of requirements regardless of their physical location can greatly simplify network design. This enables network operators or infrastructure providers the capability to manage network resources being used by different tenants/applications in isolation.

In legacy network switches, a common way to ensure network isolation and provide virtualization is the use of VLAN tags. VLAN tags can create multiple layer 3 networks by providing layer 2 isolation on the same physical network. When VLAN tagged packets are to simply pass through an intermediate switch via two pass-through ports, only the two ports are a member of the VLAN and are configured to allow tagged packets. The sliceable switch [10] application allows for network isolation in OpenFlow networks, by introducing another level of redirection using the concept of *slice*. *Slice* is a software enforced concept, which segregates end-user mac addresses or switch ports in different *slices*. Before a new flow is created, packet-in frames are sent to the controller. If the packet-in frames belong to the registered slice on that port, the corresponding flow is added to the relevant slice (else it is rejected). A sliceable switch can be used to create a large scale services (theoretically unlimited) within a single domain, unlike VLAN (limited to 4096 tags). It can also create a service that spans multiple domains without considering mesh connectivity between devices. Given these two mainstream isolation methods for traditional network and SDN, the critical problem faced by hybrid network is how to maintain compatibility between them.

In this paper, we propose the idea of *"virtualization in virtualization"* to allow for network isolation, the first virtualization here refers to SDN slices, and the next to virtualization via VLAN's. As discussed in section 2.2 HybNET has a global view of the physical topology, but offers a SDN-view network topology to the SDN controller as shown in Figure 4. The SDN-view network topology includes the OpenFlow switches connected by *"virtual links"*, which are composed of a succession of legacy switches. This method leaves the network intelligence to the SDN switches, while limiting the primary purpose of legacy switches to packet transport. Isolation can still be applied in virtual links by using VLAN

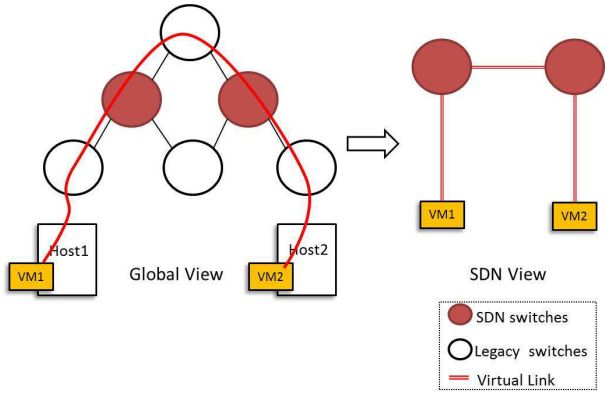

Figure 4: Convert a global view network topology to SDN view network topology by "Virtualization in virtualization" mechanism

tags, and mapping them to slice-id's.

One of the key limitations in mapping VLAN tags to slice ID's is that VLAN only supports a maximum of 4096 VLAN-ID's, while sliceable switch application does not have any such limitation. Our current solution is simplistic in the sense that we have a one-to-one mapping between VLAN ID and slice ID, this provides a clear abstraction for network virtualization. This solution can work for any hybrid topology, but it is constrained by the range of tags allowed by VLAN and cannot support more than 4096 VLAN's. Another simplistic mechanism we have tested is to provide network isolation at the OpenFlow layer using sliceable switch and provide no isolation guarantees in the "virtual links" (i.e. VLAN tags are not used in legacy switches, and packets are simply forwarded without any layer 2 isolation). This solution assumes, that either all top-of-the-rack switches are OpenFlow switches, or that each end-host machine has an OpenVSwitch[6] (software OpenFlow switch). Hence, each end-user(VM or end-host) can only be accessed via an OpenFlow enabled switch, which can implement network slices using sliceable switch application thus ensuring isolation. While this solution provides end-user isolation, it does not provide any isolation in the legacy switches, and requires certain restrictions in the physical topology.

2.4 General API

As mentioned earlier, HybNET supplies a common API for network operators to use to process transactions and configure hybrid network infrastructure across boundaries.

For example, say a cloud operator wants to create a new Virtual Tenant Network for company A, and assign some VM's to it. The network operator would first need to add a HybNET slice by calling the function *add_slice(slice_id)*. A successful execution of this function will return a new HybNET slice_id which corresponds an OpenFlow slice and a VLAN tag. Further the operator can add a VM to the given slice by calling the function *add_vm_to_slice(tenant_info, ins_info, net_list)*. As specified, this function requires the information about the owner, VM information (vm id, mac address, host ip etc.), and the slice id generated in the previous call: a successful execution of this function results in HybNET adding the list of VM's to the specified slice. The following are some of the common API functions provided by HybNET.

add_slice(slice_id):
Create a slice in sliceable switch database
@slice_id: unique id for a slice

delete_slice(slice_id):
Delete a slice from sliceable switch database
@slice_id: existing slice_id to be deleted

add_vm_to_slice(tenant_info, ins_info, net_list):
Add a VM to an existing slice and create the network connectivity
@Tenant_info: provides the owner information of this request.
@Ins_info: specifies VM information, such as id, mac, host ip etc.
@Net_list: provides logical topology information including VLAN-ID and Slice-ID.

delete_vm_to_slice(ins_info):
Remove a VM from an existing slice and delete the network connectivity
@Ins_info: specifies VM information.

3. IMPLEMENTATION

In this section, we describe the implementation of HybNET, and it's integration with a well known cloud service provider to test out our system. HybNET has primarily been built in python with the majority of the implementation focused on the controller: this includes the logic to maintain physical and logical topologies in back-end databases, path finding algorithms, and driver interfaces used to communicate with legacy switches and OpenFlow controllers. We also use ruby which is mainly used to manage NETCONF[18] enabled legacy switch configuration.

The prototype of HybNET includes a command line interface which allows administrators to issue network requests. Currently, we have integrated HybNET with OpenStack (Grizzly) an open-source cloud computing platform. Specifically, HybNET works in tandem with Neutron the network service manager of OpenStack to provide hybrid network management capability. Further, in our proof-of-concept implementation each end-host is configured with an OpenvSwitch (OVS) which allows for node level virtualization. The OVS is a software switch based on the OpenFlow protocol, and is connected to HybNET via Neutron. OVS provides node level virtualization by leveraging TAP[1] devices and KVM. All VMs running on the same compute host attach their TAP devices to an OVS integration bridge. Integration bridge isolates attached virtual ports by their logical network tags assigned by network operator and establishes corresponding flow rules for packet forwarding. With the help of OVS, multiple logical networks are able to share one or multiple physical interfaces in an isolated manner. Network requests forwarded to Neutron are intercepted by HybNET as input from the network administrator. Subsequently, HybNET computes and forwards rule updates to Trema[11] OpenFlow controller, and NETCONF based RPC scripts.

[1]further explanation of the exact design can be found at `http://docs.openstack.org/trunk/openstack-network/admin/content/under_the_hood_openvswitch.html`

4. EVALUATION & CASE STUDY

In this section, we describe a real world implementation of HybNET in managing a cloud infrastructure. Our test-bed consists of compute nodes running on 4-core 3.4 GHz servers and 8 GB memory. Each server contains a number of virtual machines connected to an OVS. The physical network infrastructure consists of Juniper EX2200 series switches(legacy switches) and NEC Programmable-Flow PF5240 switches, which support OpenFlow protocol. We have tested HybNET by integrating it with OpenStack and tested it's feasibility on a mini-network. Additionally, to test performance of HybNET, we simulate a large network environment and check on our frameworks efficiency.

Since HybNET performs no run-time rule update, we believe that it will have no impact in network latency and throughput. The only job performed by HybNET is when configuring the network infrastructure for managing cloud admin, and tenant requests. Hence, we focus on measuring the performance of HybNET only in terms of performing network operations. We adopt Infrastructure Response Time (IRT)[1] (latency between placing and completing a *transaction* on the network infrastructure) as a key metric to evaluate infrastructure performance. The remainder of this section, discusses the automation and flexibility of HybNET as well as the latency of network administration tasks.

In order to measure overhead, we first connect a real mini-network and beef up the database by simulating input of a much larger fat-tree data-center topology(see Figure 5). The advantage of this scheme is that we have a large mock network which increases the complexity substantially, while a real mini-network which can be used to test out the performance and feasibility of HybNET. Our fat-tree is built with k-port switches supporting $k^3/4$ hosts, where k = 16 to simulate 320 switches and 1,000+ hosts. The simulated data structures (switches, hosts and interfaces) are stored in the physical infrastructure database, used by hybrid-controller to generate mock physical topology graph at initialization stage. The core switches are OpenFlow switches while the aggregation and edge switches are legacy switches[2]. We simulate such a network topology to measure how long hybrid-controller will take to establish connectivity paths in a real world network environment.

To showcase the feasibility of HybNET we profile VM network provisioning in OpenStack, which we found to be the most time consuming task, and which requires connectivity establishment by our controller. IRT of provisioning a VM can be broken down into the following paths: image template fetching, resource preparation (*e.g.*, virtual network provision, storage provision), connectivity establishment (carried out by hybrid-controller) and VM spawning (VM creation on the host). We configure the network request such that the connectivity path (shortest path from our path-finder) is generated along the real physical switches comprising of both legacy and OpenFlow switches. Figure 6 shows the latency breakdown of our profiling results when provisioning a VM on top of OpenStack infrastructure. The OpenStack's compute API (Nova) handles the provision request from an end user within less than 1 seconds, including

[2]please note HybNET is topology agnostic hence it does not matter if we use any other topology

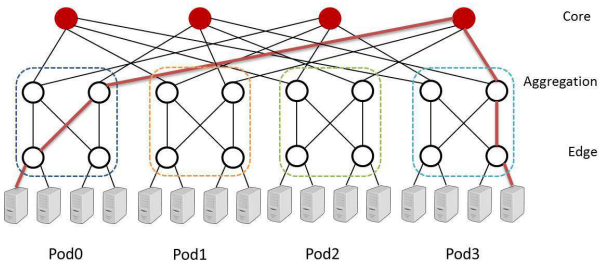

Figure 5: Example of Fat-Tree topology with k = 4

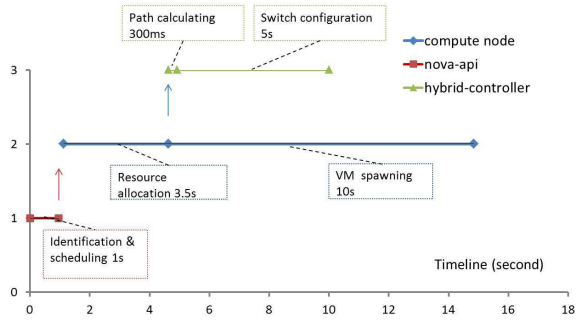

Figure 6: Latency breakdown for provisioning a VM on top of OpenStack

identification and scheduling tasks. The total VM provision time is about 13.5 seconds (we didn't count the VM booting time), among which 3.5 seconds are for resource preparation and 10 seconds are for VM spawning. Hybrid-controller consumes less than 6 seconds in total. This result is the worst case measurement by choosing the longest possible path in the network (hence it requires configuration of the maximum number of switches). The algorithm takes around 300 milliseconds for calculating shortest path, consisting of 4 legacy switches and 1 OpenFlow switch. Even though legacy switch configuration is parallelized, we found that the configuration of legacy switches was the biggest bottleneck as it takes around 5 seconds to finish setting up new configurations. Unfortunately, this is a limitation of the hardware and it's network configuration capabilities. While not relevant directly, it should also be noted that in the case of OpenStack since the time consumed by hybrid-controller is much smaller than VM spawning, it's latency is hidden from the end-user.

Thus hybrid-controller supplies automation management without obviously influencing original infrastructure provision performance. We believe that HybNET is highly beneficial as it provides complete automation, as well as network virtualization with best-effort performance based on the given physical infrastructure.

5. RELATED WORK

SDN controller frameworks such as Trema[11], NOX[5], and academic projects such as OpenDaylight[7] provide useful

mechanisms for network administrators to write controllers and management interfaces to simply the network management process. Trema is an open source framework written in Ruby and C, which provides a rich library to the developer to design a OpenFlow controller. It contains a rich library of existing applications, and a large open-source community contributing updates. NOX is one of the early SDN controllers, developed by Nicira[4] and has since been the basis for many research projects and early exploration in the SDN space. OpenDaylight is a community-led, open, industry-supported framework, for accelerating adoption of SDN in the industry and academia. However, all of these existing mechanisms focus on a full deployment of SDN infrastructure, and cannot be directly applied to a hybird network.

Recently, Google described B4 [17] it's mechanism to transition from it's pure legacy network infrastructure to an SDN controlled fabric for managing connections between it's datacenters. While an interesting insight into the challenges faced in managing hybrid infrastructures, B4 focuses on network transition specifically for Google's network infrastructure and cannot be generalized. Fabric[15] and Panopticon[21] discuss various topologies that can be used in designing a hybrid network, which can provide some of the high level network functionalities and advantages provided by SDN (such as isolation, access control and load balancing etc.). On the other hand, HybNET is topology agnostic and focuses on providing a framework for management of a hybrid network. In a way, approaches such as Fabric and Panopticon[15, 21] are complimentary to our goals and showcase some interesting designs which can be leveraged by HybNET to provide richer network functionalities.

6. CONCLUSION

In this paper, we provide HybNET a network management framework for providing automation in configuring hybrid networks composed of both legacy and OpenFlow switches. Additionally, our system provides compatibility between VLAN and sliceable switch application to provide seamless network virtualization across boundaries. Our tool aims to retain some of the advantages of SDN networks while making network automation feasible across different network infrastructures. We showcase our system by integrating it with OpenStack a well known open-source cloud service provider, and have tested this tool on a real hybrid network infrastructure to show its feasibility and performance.

7. REFERENCES

[1] Infrastructure performance management for virtualized systems. `http://www.ca.com/us/~/media/files/whitepapers/infrastr-perform-mgmt_238145.aspx`.

[2] Link layer discovery protocol. `http://www.extremenetworks.com/libraries/products/LLDP_TB.pdf`.

[3] Neutron. `http://wiki.openstack.org/wiki/Neutron`.

[4] Nicira. `http://nicira.com/`.

[5] Nox. `http://www.noxrepo.org/nox/about-nox/`.

[6] Open vswitch. `http://openvswitch.org/`.

[7] Opendaylight. `http://www.opendaylight.org/project`.

[8] Openstack. `http://www.openstack.org/`.

[9] Simple network management protocol. `http://net-snmp.sourceforge.net/`.

[10] Sliceable switch. `https://github.com/trema/apps/wiki/sliceable\_switch\_tutorial`.

[11] Trema. `http://trema.github.io/trema/`.

[12] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan. Best-path vs. multi-path overlay routing. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, IMC '03, pages 91–100, New York, NY, USA, 2003. ACM.

[13] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 8:1–8:12, New York, NY, USA, 2011. ACM.

[14] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the network forwarding plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, page 8. ACM, 2010.

[15] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: a retrospective on evolving sdn. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 85–90. ACM, 2012.

[16] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM, 2011.

[17] S. J. et al. B4: Experience with a globally-deployed software defined wan. 2013.

[18] B. Hedstrom, A. Watwe, and S. Sakthidharan. *Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions*. PhD thesis, MasterâĂŹs thesis, University of Colorado, 2011.

[19] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving High Utilization with Software-Driven WAN. In *ACM Sigcomm*, 2013.

[20] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[21] D. Levin, M. Canini, S. Schmid, and A. Feldmann. Toward transitional sdn deployment in enterprise networks.

[22] 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*, 38(2):69–74, 2008.

[23] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent updates for software-defined networks: Change you can believe in! In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 7. ACM, 2011.

[24] F. P. Tso, G. Hamilton, R. Weber, C. Perkins, and D. Pezaros. Longer is better: exploiting path diversity in data center networks. 2013.