Software-Defined Network Solutions for Science Scenarios: Performance Testing Framework and Measurements

N. Rao, D. Katramatos

Submitted to the International Conference on Distributed Computing and Networking Conference
to be held at Varanasi, India
January 04 - 07, 2018

January 2018

Computational Science Initiative

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Advanced Scientific Computing Research (SC-21)

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or any third party's use or the results of such use of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof or its contractors or subcontractors. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Software-Defined Network Solutions for Science Scenarios: Performance Testing Framework and Measurements

Nageswara S.V. Rao, Qiang Liu, Satyabrata Sen
Oak Ridge National Laboratory
Oak Ridge, TN, USA
{raons,liuq1,sens}@ornl.gov

Raj Kettimuthu, Josh Boley
Argonne National Laboratory
Argonne, IL, USA
{kettimut,jboley}@mcs.anl.gov

Bradley W. Settlemyer, Hsing B. Chen
Los Alamos National Laboratory
Los Alamos, NM, USA
{bws,hbchen}@lanl.gov

Dimitrios Katramatos, Dantong Yu
Brookhaven National Laboratory
Upton, NY, USA
{dkat,dtyu}@bnl.gov

ABSTRACT

High-performance scientific workflows utilize supercomputers, scientific instruments, and large storage systems. Their executions require fast setup of a small number of dedicated network connections across the geographically distributed facility sites. We present Software-Defined Network (SDN) solutions consisting of site daemons that use dpctl, Floodlight, ONOS, or OpenDaylight controllers to set up these connections. The development of these SDN solutions could be quite disruptive to the infrastructure, while requiring a close coordination among multiple sites; in addition, the large number of possible controller and device combinations to investigate could make the infrastructure unavailable to regular users for extended periods of time. In response, we develop a Virtual Science Network Environment (VSNE) using virtual machines, Mininet, and custom scripts that support the development, testing, and evaluation of SDN solutions, without the constraints and expenses of multi-site physical infrastructures; furthermore, the chosen solutions can be directly transferred to production deployments. By complementing VSNE with a physical testbed, we conduct targeted performance tests of various SDN solutions to help choose the best candidates. In addition, we propose a switching response method to assess the setup times and throughput performances of different SDN solutions, and present experimental results that show their advantages and limitations.

CCS CONCEPTS

• **Networks → Network measurement**; **Network performance evaluation**;

KEYWORDS

Software-Defined networks, dedicated connections, science scenarios, controllers, switching response method

1 INTRODUCTION

Today's science infrastructures consist of several scientific instruments, supercomputers, storage/file systems, and custom visualization facilities [5], which are located at geographically dispersed sites. These sites are typically connected over a combination of shared and dedicated high-performance networks. The scientific workflows span these distributed facilities, and their executions require a small number of high-performance network flows among various remote sites for specific time periods. These network flows in turn require dedicated high-capacity, low-latency, and/or low-jitter connections among the sites. Currently, such network connections are composed and configured manually by site and Wide-Area Network (WAN) operators, with typical lead times of days or even longer.

Recent advancements in Software-Defined Network (SDN) solutions [8, 18] promise fast and automatic provisioning within seconds. We present SDN solutions to set up these connections using a network of site daemons operating on a control plane over the shared network. They utilize custom scripts and controllers to coordinate the path setup and teardown operations over the dedicated high-performance data plane. Once a connection request is received from the users or automated scripts, a site daemon translates and communicates the request to other site daemons and the WAN daemon. Then, these connection requests are translated into commands for setting up the site network elements and sent to the site controller, which then installs suitable flow entries on site switches. These solutions utilize custom dpctl scripts [10], OpenDaylight (ODL) [3], Floodlight [1], or ONOS [2] controllers to maintain network connections by inserting, modifying, and deleting flow entries on OpenFlow devices.

The development, testing, and evaluation of these SDN solutions over science network infrastructures are extremely challenging because they require: (i) close coordination and allocation of infrastructures at multiple sites, (ii) potential infrastructure disruptions during the initial testing of SDN codes, and (iii) long testing periods to assess solution choices from a large number of possible combinations of SDN devices and controllers, during which the infrastructure is unavailable to the regular users. To overcome these challenges, in this paper, we develop a Virtual Science Network Environment (VSNE), wherein various sites and WAN configurations are emulated using Virtual Machines (VMs), and their sub-networks are emulated using Mininet within VMs. Building on this VSNE, we

ICDCN '18, January 4–7, 2018, Varanasi, India

ACM ISBN 978-1-4503-6372-3/18/01…$15.00
https://doi.org/10.1145/3154273.3154336

implement our SDN solutions to set up the dedicated connections using a network of site daemons. Thus, VSNE enables the development and testing of SDN solutions, including highly disruptive ones, while the physical infrastructure is being built, and it does not require multi-site physical infrastructure allocation and coordination. In particular, our VSNE emulates the infrastructure that is currently being built among four national laboratories, namely, Argonne National Laboratory (ANL), Brookhaven National Laboratory (BNL), Los Alamos National Laboratory (LANL), and Oak Ridge National Laboratory (ORNL). VSNE runs on a workstation and is replicated at all four sites so that their respective SDN site-solutions can be tested and synchronized, and upon maturity, can be rolled into the respective site physical infrastructure.

In addition to SDN functionality testing, VSNE also supports certain performance tests, for example, to assess path setup and teardown times of controllers, and to help choose among the solutions. However, due to the underlying Mininet emulations, VSNE is subject to bandwidth constraints that limit our performance studies to smaller connection capacities. Hence, we develop a complementary physical testbed to conduct throughput tests. We collect ping and Transmission Control Protocol (TCP) throughput time traces using both VSNE connections and physical 10GigE connections to assess the responsiveness of SDN solutions for connection setup and teardown operations. The ping measurements capture the connection setup and teardown times in both cases. However, somewhat unexpectedly, TCP dynamics over VSNE connections deviate significantly from those on physical connections, and hence we use the latter in throughput tests. Our measurements from targeted tests provide valuable insights into the strengths and limitations of different SDN solutions. For example, using HP 5604zl switches, under legacy OpenFlow 1.0, dpctl scripts provide faster responses than ODL Hydrogen controllers by over a second; but under OpenFlow 1.3, their performances are comparable to those using Floodlight and ONOS controllers.

Individual SDN solutions composed of switches, together with controllers and their corresponding custom northbound codes, can be quite varied, e.g., HP switches with dpctl scripts and Cisco switches with ODL controllers. To objectively compare the performances of disparate SDN solutions, we propose the *switching response* method. In general, SDN solutions can be composed by choosing from a wide variety of controller and device combinations; they may include open-source and vendor-specific SDN controllers, along with different OpenFlow implementations by device vendors, ranging from building additional software layers on existing products to developing completely native implementations. An ideal response to a connection setup request would be an immediate ping return and an instantaneous TCP throughput maximization. However, in practice, the responses lag and may contain complex transients that depend on site daemons, controllers, and network devices. The switching response method enables us to quantify the performance of SDN solutions using responses to a train of connection setup and teardown events, and objectively compare and choose suitable candidates for a given infrastructure.

The organization of this paper is as follows. In Section 2, we briefly describe scientific workflows to motivate the needed SDN solutions, and then present a daemon-based SDN solution. Details of VSNE and physical testbed are presented in Section 3. The measurement analysis and switching response method are described in Section 4. The paper concludes in Section 5.

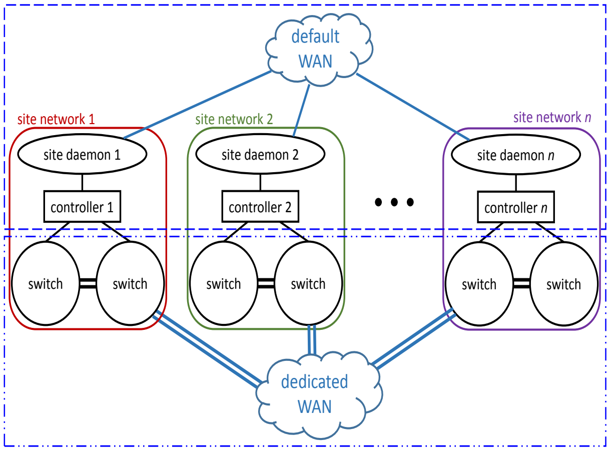


Figure 1: SDN solution using interconnected site daemons.

2 SDN SOLUTIONS FOR SCIENTIFIC WORKFLOWS

Effective implementations of network connections for scientific workflows are critical to successful collaborations among researchers in many disciplines [5]. At present, custom-designed Local Area Network (LAN) and WAN connections are provisioned manually by different network engineer teams, often requiring days. Furthermore, valuable resources are often over-provisioned to meet peak transient needs. Recent developments in SDN, network function virtualization (NFV), and related technologies [8, 9] hold an enormous promise for fast automatic provisioning of network connections. However, the science network flows represent a different set of challenges compared to data center environments, which have been mainly targeted by many rapidly developing SDN technologies. The adoption and deployment of these technologies have been somewhat slow in science networks due to operational and security considerations [14]. One predominant feature of science network flows is that a small number of them originate from known sites, and they involve high-volume and high-precision flows over multi-domain WAN, LAN, and storage area networks. In terms of network infrastructures, science environments provide two types of services: (i) a default IP network to support user login and other services, and (ii) dedicated high-bandwidth data connections either on demand or through advance reservation.

2.1 Site-Service Daemon Framework

For the science scenarios considered here, a single controller solution is not effective, although such approaches with stable control-plane connections have been effective in path/flow switching over local area networks using OpenFlow [13, 15] and cross-country networks using customized methods [11, 12]. In our SDN solution, a set of *site daemons* provide basic connectivity among the sites over

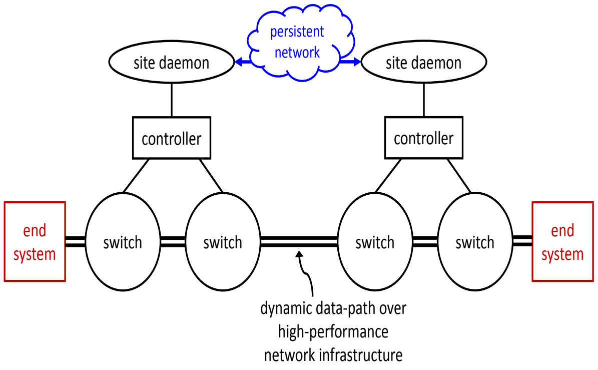


Figure 2: Illustration of path setup between two sites.

the default IP network. These site daemons maintain persistent connectivity among themselves and also with the local site controllers, switches, and science users, as shown in Figure 1. At each site, the local daemon receives connection requests (setup/teardown) from either local users, or automated workflow agents, or remote site daemons. Then, in response, the local daemon performs two tasks: (i) invokes custom scripts to setup or teardown connections within the site by interacting with the site controllers and switches, and (ii) generates specifications for WAN and remote site connections, and communicates them to WAN and remote site daemons. We show an example of two-site path setup in Figure 2.

2.2 Controllers

We consider two types of site/WAN controllers using dpctl scripts and one of the open-source controllers, namely, ODL, Floodlight, and ONOS, as described in the following.

2.2.1 dpctl Controller. Some vendors support dpctl API, as part of OpenFlow implementation, which enables hosts to communicate with switches to insert new flows, query the status of flows, and delete existing flows. We utilize custom dpctl controller scripts for flow setup and teardown by directly communicating with the switches. The execution path of this code is very simple as it involves direct communications with the switch, as shown in Figure 3(a). A similar solution was developed to support an automatic fail-over of long-haul connections in [10].

2.2.2 Open-Source Controllers. The SDN controllers, including ODL, Floodlight, and ONOS, communicate with OpenFlow switches to install, query, and delete flow entries on their southbound interface. The site-service daemon communicates with site controllers via the northbound interface. Specifically, we use curl or Python scripts to communicate flow setup and teardown operations to the controller via REST interfaces, as illustrated in Figure 3(b); the contents of these flow entries are identical to the ones used in the above dpctl controller solution. However, compared to dpctl scripts, these controller codes are much more complex to analyze, because they involve not only the REST scripts but also the controller software stack which by itself is fairly complex. Thus, both the software and execution paths are much more complicated compared to the dpctl method, and these controllers are also required to run constantly on the servers at end sites.

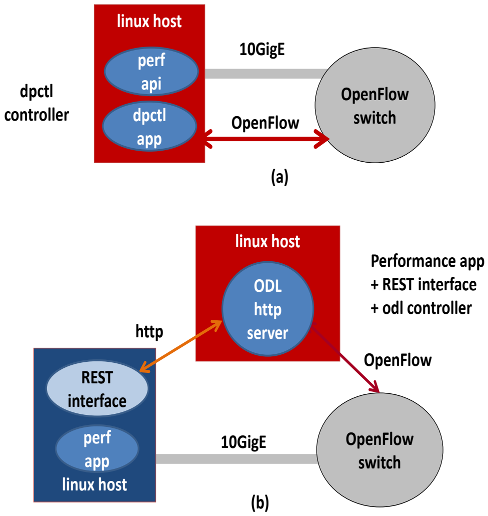


Figure 3: Configurations of dpctl and controllers in physical testbed.

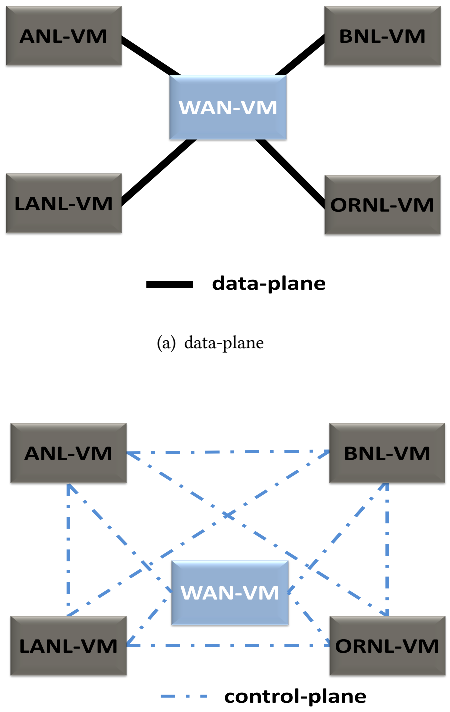
3 EXPERIMENTAL AND VIRTUAL DEVELOPMENT AND TEST ENVIRONMENT

In this section, we describe VSNE and testbed hardware used for evaluating various SDN solutions. In particular, the VSNE is primarily used for code development and functionality testing, whereas the testbed is used in targeted tests for complementary performance assessment, more specifically, without being constrained by the capacity limits and non-representative TCP dynamics of VSNE connections.

3.1 Virtual Science Network Environment

We develop a VSNE with five VMs, four of which emulate the sites, ANL, BNL, LANL, and ORNL, and a fifth one emulates the dedicated ESnet [7] WAN connections among these sites, as shown in Figure 4; the overall framework is general and can emulate multi-site networked infrastructures. The VMs run under VirtualBox 5.1 environment on a Linux host with RHEL 7.2 kernel. For each site VM, two internal interfaces are enabled, one for control-plane communications among all site daemon codes, and the other for dedicated data-plane connections between site VMs and WAN VM. The former emulates the persistent network over default WAN among site daemons shown in Figure 1, and is implemented using internal networking feature of VirtualBox VMs, which supports communications between processes running within all VMs on a host.

Figure 5 illustrates the configuration of a site VM. Using Python scripts running in the Mininet environment, we create two hosts $h1$ and $h2$ that are connected to a switch $s2$, which is in turn connected to another switch $s1$. Both $s1$ and $s2$ are Open vSwitches whose flows can be dynamically configured as needed. These switches in our VSNE framework represent the deployed OpenFlow hardware



(a) data-plane

(b) control-plane

Figure 4: Data-plane and control-plane connections.

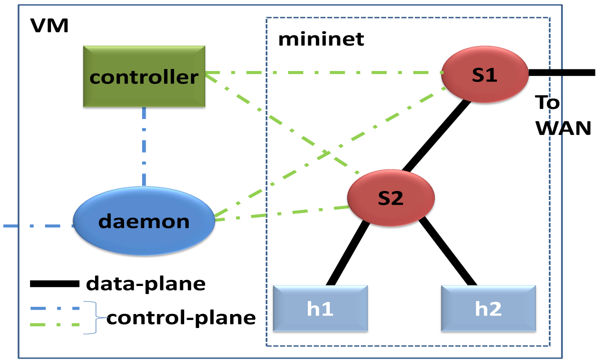


Figure 5: Site VM configuration with two hosts and two switches.

switches, and in particular, they support both dpctl and various controllers so that VSNE codes can be transferred to physical networks. Also shown in Figure 5 are a site daemon and an SDN controller. The connections among Mininet switches and hosts constitute the data-plane, and the northbound connections between site daemons and SDN controller and southbound connections between the controller and switches constitute the control plane. In the case of dpctl scripts, the site daemons directly control the switches via the control-plane connections.

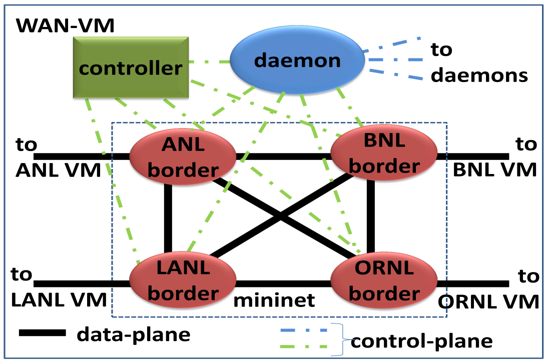


Figure 6: WAN VM configuration for four sites.

Table 1: Configurations of long-haul link latency on WAN VM

Link	Latency (ms)
ANL–BNL	24
ANL–LANL	80
ANL–ORNL	13
BNL–LANL	90
BNL–ORNL	48
LANL–ORNL	80

On the WAN VM, four "gateway" Open vSwitches are created in Mininet and the links between these switches emulate the long-haul physical connections, such as OSCARS [4] circuits between the sites. Figure 6 illustrates both the data-plane connections among the switches and to WAN, and the control-plane connections between controllers and/or the WAN daemon. We emulate long-haul link latency between the site-pairs in the Mininet environment using delay parameters shown in Table 1. A simple rate control mechanism is also implemented in the same script where the maximum bandwidth of the links is set to be 20 Mbps. The site configurations and connections to WAN for a multi-site infrastructure are illustrated in Figure 7.

The switching response method to be described in Section 4 entails flow insertion and deletion on the switches. We run connectivity tests using the command **ovs-ofctl**, considered as a specific version of the dpctl method for Open vSwitches. In particular, the **add-flow** and **del-flows** methods are respectively used to add and delete flows on a particular switch. Alternatively, we dynamically add or delete flows using curl scripts while the SDN controller (ODL, Floodlight, or ONOS) is running. Once all flows are added, we run **ping** to test the connectivity and the RTT, where ICMP packets are exchanged between end hosts. On the other hand, we run TCP **iperf** on end hosts to test the instantaneous transfer rates between the host pair. A first version of VSNE spanning the four laboratory sites has been implemented and replicated at the sites, and functionality tests have been carried out.

3.2 OpenFlow Testbed

An experimental testbed is set up for targeted performance tests and early code transition from VSNE, which consists of two site

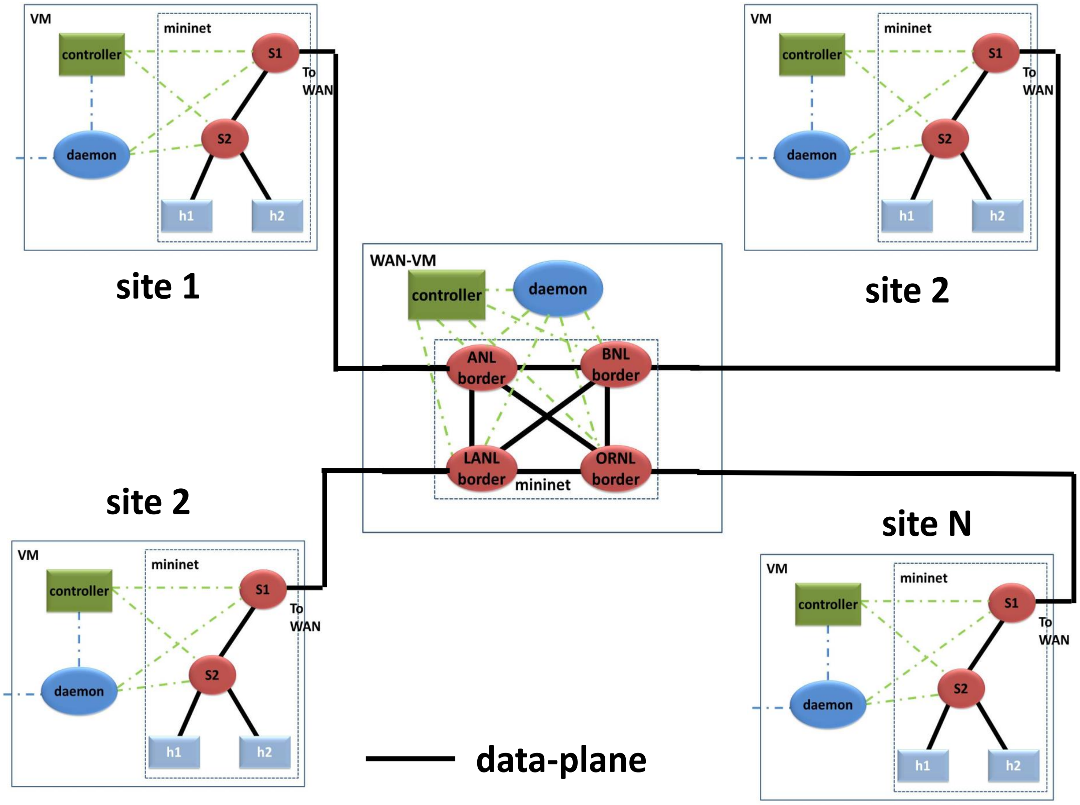


Figure 7: Data-plane connections within the sites and WAN, and between sites and WAN are shown as solid lines, and control-plane connections within the sites and WAN are shown as dashed lines.

LANs, each with multiple hosts connected via 10GigE NICs to the site's border switch. The testbed, as shown in Figure 8 for dpctl and ODL options, is a network connection-level representation of the physical infrastructure spanning ANL, BNL, LANL, and ORNL. The border switches are connected to one another via local fiber connection of a few meters in length and ANUE devices that emulate long-haul connections in hardware. Tests are performed in configurations that use pairs of HP 5064zl as border switches, which are OpenFlow-enabled and support dpctl.

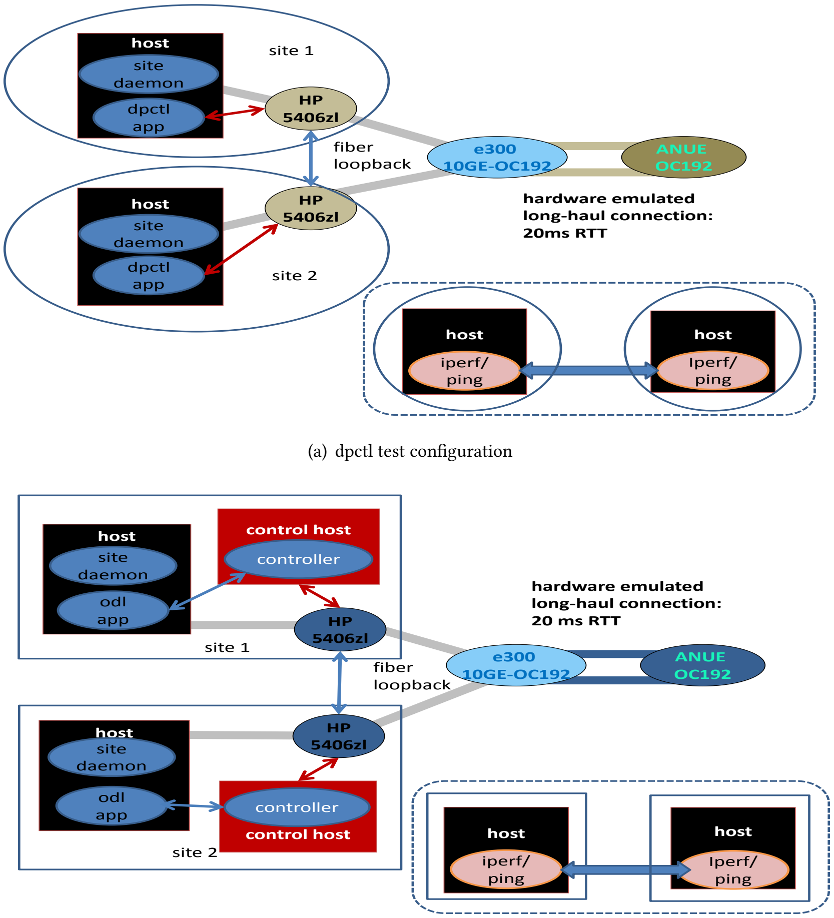
Two classes of Linux hosts are connected to border switches. The controller hosts (e.g., "feynman 1" and "feynman 2") are utilized to run ODL/ONOS/Floodlight controllers, and client and server hosts (e.g., "feynman 3" and "feynman 4") are used to execute site modules along with client server codes, for example, iperf clients and servers. The *dpctl* tests utilize only client and server hosts to execute both monitoring and site/WAN daemon codes. For ODL tests shown in Figure 8(b), site codes on client/server hosts utilize REST interfaces of controllers to communicate flow messages needed for connection setup and teardown operations. We employ open source controllers running on controller hosts and site daemon modules running on server/client hosts.

4 SWITCHING RESPONSE METHOD

We characterize the performance of an SDN solution by its ping and TCP throughput responses to periodic "switching" events that trigger flow manipulations on the switches, namely, insertion and/or deletion of flow entries. Each event under a network configuration X engages: (i) software components including SDN controllers, northbound scripts, and site daemons, and (ii) hardware components including switches, connections, and host systems. We propose the *switching response* method that utilizes averages from ping and/or TCP throughput traces collected under a switching sequence, wherein the connection is periodically switched at the beginning of a fixed interval period T, and after duration $T_D < T$.

4.1 TCP and ping Measurements

We apply two different switching sequences for performance tests; we primarily focus on ping for VSNE due to its limitations on TCP dynamics as described below. For VSNE, the switching sequence corresponds to periodic datapath setup and teardown as shown in Figure 9(a), along with expected ping and iperf responses shown in Figures 9(b) and 9(c) respectively. For illustration purposes and also to unify the analysis of TCP and ping measurements, ping values under a torn-down connection are shown as small negative values



(a) dpctl test configuration

(b) ODL test configuration

Figure 8: Test configurations for dpctl and ODL SDN solutions.

close to zero. For the testbed, we additionally consider switching between a direct fiber connection with 0 ms RTT and an emulated connection with 20 ms RTT (which would have yielded somewhat of a “flipped” version of the ideal ping response in Figure 9(b)). In all cases, the ping response is fairly fast, resulting in rectangular-shaped responses shown in Figure 9(b). On the other hand, TCP throughput reacts somewhat slower to connection changes as it recovers, leading to trapezoidal-shaped responses abstracted in Figure 9(c).

4.1.1 Testbed Measurements. Two sets of experiments are run using the testbed configuration as shown in Figure 8. The first set utilizes OpenFlow 1.0, and the traces alternate between 50 sec local-connection periods and 10 sec no-connection periods as shown in Figure 10. For both dpctl and ODL Hydrogen methods, the throughput quickly reaches the connection capacity for the local connection and drops to zero upon teardown. However, the average recovery time for ODL is about one second slower (see Figure 11), and these results are consistent with similar results discussed in [10].

The second set of experiments use OpenFlow 1.3 with dpctl and Floodlight/ONOS controllers, where the paths are switched between 0 ms and 20 ms connections. Table 2 displays time epochs that correspond to the change-points in the 500 sec TCP iperf traces (in 0.1 sec resolution) in response to path setup and teardown events scheduled at 50 sec intervals, and Figure 12 shows two sample traces of the switching response with Floodlight. The peak throughput is lower than in previous experiments because of the different connection configuration and default TCP buffers. Interestingly, the advantage of dpctl’s faster response disappears in these tests. A closer examination reveals that the performance differences between controllers and dpctl method are less obvious, i.e., within 1 sec on average; however, the effect of response lag here is cumulative in that the lag increases over time as the test continues. As shown in Table 2, compared to the response times in the “1st 0ms” column, where the data-path connection is set up for the first time, four cycles later, the values in the “5th 0ms” column exhibit variable increases in response time: 0.5 sec for Floodlight, 1 sec for ONOS, and 1.5 sec for dpctl. The average differences among the response times are within 1 sec over the five periods.

These targeted tests highlight the performance differences in OpenFlow versions, and also reflect certain practical aspects. For

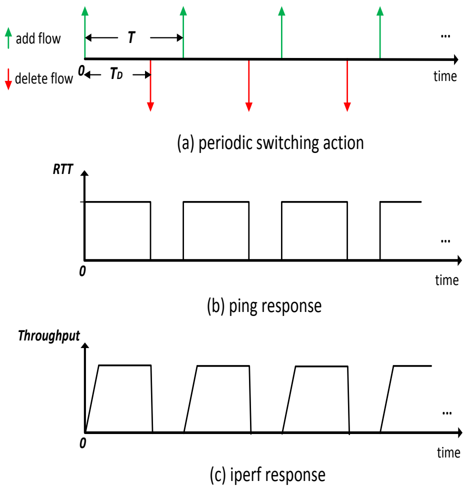


Figure 9: Illustration of Flow actions and switching responses using ping and iperf.

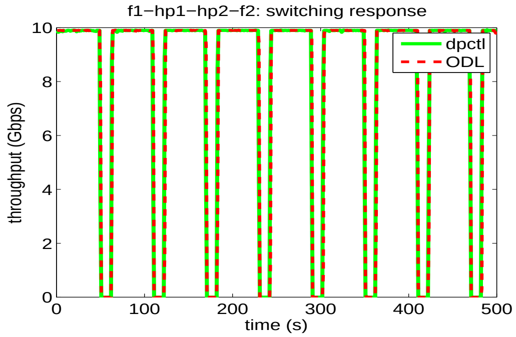


Figure 10: Testbed iperf switching response running OpenFlow 1.0.

OpenFlow 1.3, HP switches employ a more complex implementation and flow table structure compared to OpenFlow 1.0, and dpctl code base is also re-written. Their combined effects contribute to the somewhat slower response of the dpctl method. The OpenFlow 1.0 results are still of interest as it remains in use at some sites, while the transition to OpenFlow 1.3 is being evaluated from operations and security perspectives. Even in our more flexible testbed environment, this transition has been time-consuming, as it involves re-writing of dpctl scripts and customization of Floodlight and ONOS southbound interfaces to match the upgraded HP switches.

4.1.2 VSNE Measurements. We collected switching responses for ODL Beryllium, Floodlight, and ONOS controllers over VSNE. Here we present the results for ODL since statistically the others are quite similar. Each test run comprises of 5 flow-event periods with $T = 30$ sec and $T_D = 20$ sec. The site daemon code is invoked periodically by the sender site VM that connects a local virtual host to a remote virtual host located on another site VM. In contrast to testbed TCP throughput traces shown in Figure 10, the switching responses here exhibit significant delay and variations across repeated test runs, as shown in Figure 13 for the ANL-ORNL connection. The throughput recovery time after the flows are restored is at least 6 sec in this case, and it is not uncommon to encounter 10 sec or longer lag times. In addition, although the link capacity (20 Mbps) is eventually reached, the throughput traces exhibit significant instability, reflected by a large proportion of throughput rates in 10-15 Mbps range, especially for the ovs-ofctl configuration. We conjecture that such out-of-sync phenomena for TCP packets are the combined effect of complex TCP dynamics and limitations of Open vSwitches (under OpenFlow 1.3) in Mininet emulation environment. On the other hand, ICMP packets used by ping have less complex dynamics, as reflected in much more well-behaved traces, and provide insights into the switching performance as shown next.

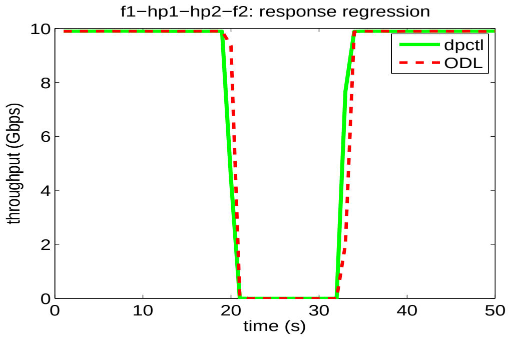


Figure 11: Testbed iperf switching response regression running OpenFlow 1.0.

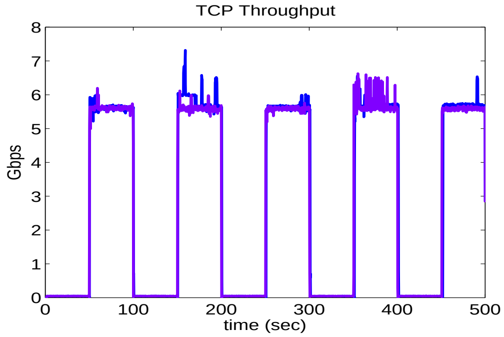


Figure 12: Testbed iperf switching response with Floodlight and OpenFlow 1.3.

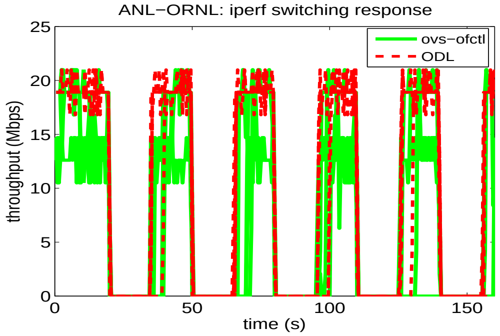
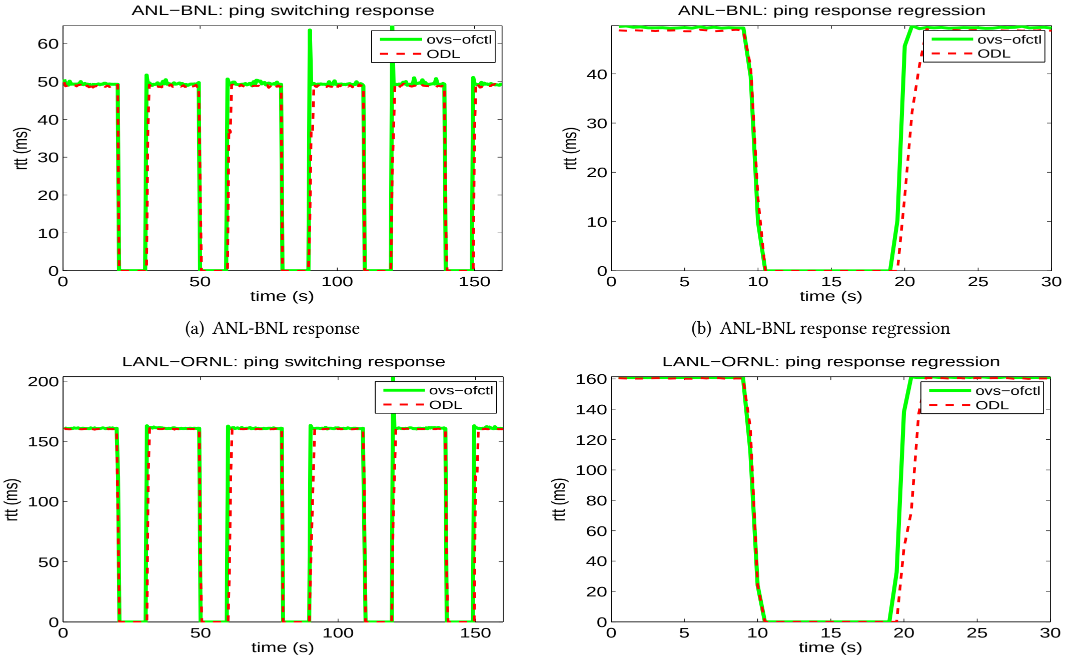


Figure 13: ANL–ORNL sample iperf switching response in VSNE.

Table 2: TCP switching response time epochs (in seconds) in hardware testbed running OpenFlow 1.3

Options	1st 0ms	1st 20ms	2nd 0ms	2nd 20ms	3rd 0ms	3rd 20ms	4th 0ms	4th 20ms	5th 0ms
dpctl	50.4	100.4	150.8	200.8	251.2	301.2	351.6	401.6	451.9
Floodlight	50.3	100.2	150.4	200.3	250.5	300.4	350.7	400.6	450.8
ONOS	50.3	100.2	150.6	200.5	250.8	300.8	351.1	401.0	451.3



(a) ANL-BNL response

(b) ANL-BNL response regression

(c) LANL-ORNL response

(d) LANL-ORNL response regression

Figure 14: Select virtual connection ping switching response and regression.

Table 3: Ping switching response and packet delivery performance over virtual connections

Link	Response Time (s)		Packet loss rate	
	ovs-ofctl	ODL	ovs-ofctl	ODL
ANL–BNL	0.500	1.100	31.25%	32.89%
ANL–LANL	0.500	1.150	31.33%	33.28%
ANL–ORNL	0.550	1.275	31.33%	33.59%
BNL–LANL	0.650	1.225	31.56%	33.75%
BNL–ORNL	0.500	1.225	31.25%	33.52%
LANL–ORNL	0.600	1.250	31.48%	33.67%

In Figure 14, the average switching responses, represented by the 160 sec ping trace plots (with the interval set to be 0.5 sec), and the response regression plots for a single 30 sec cycle, are shown for select site pair connections (a small negative number, shown as zero here, represent the torn-down connection RTT). In particular, from the regression plots, one can observe: (i) the trailing edge of the ovs-ofctl response curve, as when the flows are deleted, largely coincides with that of ODL; and (ii) the leading edge of the ovs-ofctl curve, shortly after the flows are added, precedes that of the ODL curve, demonstrating the shorter response time of the former.

The response delays listed in Table 3 indicate that it takes over 1 sec to recover using ODL, more than twice the amount of time with the ovs-ofctl option for most scenarios. The overall longer response time with ODL is also reflected in the aggregated packet loss rates over the 160 sec period as shown in Table 3, where over 2% of all the packets are additionally lost using ODL (compared to ovs-ofctl) because of the relatively slower response after the flows are restored.

4.2 Response Regression Function and Finite Sample Analysis

We provide an analytical justification for the conclusions reached in the previous subsection based on the responses averaged over T sec switching cycles. The main consideration is that both TCP throughput and ping estimates are overall monotone functions of time around the switching events, although they vary in opposite directions. When the current connection is switched to a shorter one, the ping estimate decreases quite rapidly whereas TCP throughput increases overall but could exhibit complex dynamics, for example, as shown in Figure 12. We utilize this underlying monotonicity property to justify our earlier conclusions by adapting the statistical method developed for long-haul connection fail-over scenarios in [10]. Such an analytical justification is important in view of various hardware and software factors that contribute to complex measured traces.

A configuration X is specified by its software consisting of SDN controllers and northbound scripts, orchestrator modules, and hardware consisting of switches, routers, and host systems. Let $\delta(t - iT), i = 0, 1, \ldots, n$ denote the sequence of connection setup and teardown commands, where t represents time and T is the *period.* Let $\mathbf{T}^X(t)$ denote the parameter of interest at time t, namely, TCP throughput or RTT estimates returned by ping as shown in Figures 10, 13, and 14, respectively. Let $R^X(t) = \mathbf{B} - \mathbf{T}^X(t)$ denote the *parameter* trace that captures the "unused" portion of the peak parameter value $\mathbf{B}$. For TCP traces over a connection with capacity $\mathbf{B}$, it is the residual bandwidth at time t above TCP throughput $\mathbf{T}^X(t)$. For data transfers, the desirable TCP response is to achieve throughput close to B during the $[0, T_D]$ interval in each period T. To unify the analysis of ping and TCP throughput measurements, we denote the RTT over a torn-down connection by a small negative number (shown as zero in Figure 14), and let B represent the RTT of the connection that has been set up. The parameter trace is close to zero when throughput is close to the peak or when the RTT estimate is accurate, and close to $\mathbf{B}$ during the $[T_D, T]$ portion of each cycle.

We define the *setup response function* as

$$R_i^X(t) = R^X(t - iT), t \in [0, T_D)$$

which is the response to the ith setup $\delta(t - iT)$, for $i = 0, 1, \ldots, n$. A response function is an impulse function that represents the instantaneous path setup and ping/TCP throughput recovery. As seen in measured traces, $R_i^X(t)$ is a somewhat "flattened" impulse function with significant statistical variations; in general, the narrower this function, the quicker the setup.

The *response regression* of configuration X is defined as

$$\bar{R}^X(t) = E\left[R_i^X(t)\right] = \int R_i^X(t) d\mathbf{P}_{R_i^X(t)},$$

for $t \in [0, T_D]$. The underlying distribution $\mathbf{P}_{R_i^X(t)}$ is in general quite complex since it depends on the dynamics of controllers and TCP or ping mechanisms. In general, we assume that it exhibits an overall decreasing profile for $t \in [0, T_D]$. For example, following the teardown period $[T_D, T]$, TCP throughput increases as it recovers from zero. For ping measurements, the transition is sharper as it becomes RTT B at $t = 0$ and drops to 0 when ping returns an accurate RTT estimate. The duration in which it stays near B is a measure of the time needed for the flows to be installed.

We define the *response mean* $\hat{R}_i(t)$ of $\bar{R}_i(t)$ using the discrete measurements collected at times $t = j\delta, j = 0, 1, ..., T/\delta$, as

$$\hat{R}^X(j\delta) = \frac{1}{n}\sum_{i=1}^{n}\left(R_i^X(j\delta)\right)$$

which captures the average profile. Based on the mean differences in TCP traces for physical connection in Figure 10, the dpctl method responds about a second faster than the ODL method. A similar relationship can be drawn from the ping traces collected over emulated VSNE connections in Figure 14; but, the VSNE TCP traces, such as those shown in Figure 13, are less conclusive, thereby indicating limitations of VSNE for more complicated TCP dynamics. From an engineering perspective, the above performance comparisons based on the measurements seem intuitively justified, and in the next subsection, we provide a statistical justification for the use of response mean $\hat{R}(t)$ by exploiting the underlying monotonic properties of the performance parameter, namely, the recovery of ping or TCP throughput following a connection setup. The derivation is a special case of a more general result presented in [10], and we provide these details for completeness.

4.3 Finite Sample Statistical Analysis

A generic empirical estimate $\tilde{R}^X(t)$ of $\bar{R}(t)$ based on discrete measurements collected at times $t = j\delta, j = 0, 1, ..., T_D/\delta$, is given by

$$\tilde{R}^X(j\delta) = \frac{1}{n}\sum_{i=1}^{n}\left[g\left(R_i^X(j\delta)\right)\right]$$

for an estimator function g from function class $\mathcal{M}$ of non-decreasing, namely, monotone functions. For ease of notation, we also denote $\tilde{R}^X(.)$ by $f \in \mathcal{M}$. The *expected error* $I(f)$ of the estimator f is given by

$$I(f) = \int [f(t) - R_i^X(t)]^2 d\mathbf{P}_{R_i^X(t),t}.$$

The *best expected estimator* $f^* \in \mathcal{M}$ minimizes the expected error $I(.)$; that is, $I(f^*) = \min_{f \in \mathcal{M}} I(f)$. The *empirical error* of an estimator f is given by

$$\hat{I}(f) = \frac{\delta}{T_D n}\sum_{i=1}^{n}\sum_{j=1}^{T/\delta}\left[f(j\delta) - \left(R_i^X(j\delta)\right)\right]^2.$$

The *best empirical estimator* $\hat{f} \in \mathcal{M}$ minimizes the empirical error $\hat{I}(.)$; that is, $\hat{I}(\hat{f}) = \min_{f \in \mathcal{M}} \hat{I}(f)$. Since the response mean $\hat{R}(t)$ is the mean at each observation time $j\delta$, it achieves zero mean error, which in turn leads to zero empirical error, i.e., $\hat{I}\left(\hat{R}\right) = 0$; thus, it is the best empirical estimator. By ignoring minor variations, $\hat{R}$ is closely approximated as a non-decreasing function, as indicated by the response means of dpctl and ODL methods, as shown in Figure 14 for ping measurements, and in Figures 10 and 13 for TCP throughput traces respectively.

In what follows, we will show that the response mean $\hat{R}(t)$ is a good approximation of the response regression $\bar{R}(t)$ as indicated by the Vapnik-Chervonenkis theory [16]. Furthermore, this performance guarantee is distribution-free, i.e., independent of the underlying joint distributions of controllers and switches, and is valid under very general conditions [17] on the variations of performance parameter (such as TCP throughput or ping RTT estimate) measurements. We emphasize that the underlying distributions are quite complicated and generally unknown, since they depend on complex interactions between controller software and switches.

First, we show that the error of estimator $\hat{R}$, given by $I\left(\hat{R}\right)$, is within ϵ of the optimal error $I(f^*)$ with a probability that improves with the number of observations n. In particular, the probability

$$\mathbf{P}\left\{I\left(\hat{R}\right) - I(f^*) > \epsilon\right\}$$

decreases with n, independent of the distributions of the controllers, switches, and orchestrator modules. In the first step, the uniform convergence property of the expected and empirical errors over the function class $\mathcal{M}$ shows that

$$\begin{aligned} &\mathbf{P}\left\{I\left(\hat{R}\right) - I(f^*) > \epsilon\right\} \\ &\quad \leq \quad \mathbf{P}\left\{\max_{h\in\mathcal{M}} \left|I\left(h\right) - \hat{I}(h)\right| > \epsilon/2\right\} \end{aligned}$$

Then, by applying the uniform bound ([6], p. 143) provided by Vapnik-Chervonenkis theory, we obtain

$$\begin{aligned} &\mathbf{P}\left\{I\left(\hat{R}\right) - I(f^*) > \epsilon\right\} \\ &\quad \leq \quad 16\mathcal{N}_\infty\left(\frac{\epsilon}{\mathbf{B}}, \mathcal{M}\right) ne^{-\epsilon^2 n/(4\mathbf{B})^2} \end{aligned}$$

where $\mathcal{N}_\infty\left(\epsilon, \mathcal{A}\right)$ is the ϵ-cover size of function class $\mathcal{A}$ under L_∞ norm. The ϵ-cover size is a deterministic quantity that depends entirely on the function class, which in turn makes the above probability bounds distribution-free in that they are valid for any joint distribution of controllers, switches, and orchestrator modules at various sites.

Next, the monotonicity of functions in $\mathcal{M}$ establishes that their total variation is upper-bounded by $\mathbf{B}$. This property in turn provides the following upper bound for the ϵ-cover size of $\mathcal{M}$ [6]:

$$\mathcal{N}_\infty\left(\frac{\epsilon}{\mathbf{B}}, \mathcal{M}\right) < 2\left(\frac{n}{\epsilon^2}\right)^{(1+\mathbf{B}/\epsilon)\log_2(4\epsilon/\mathbf{B})}.$$

By using this bound, we obtain

$$\begin{aligned} &\mathbf{P}\left\{I\left(\hat{R}_i\right) - I(f^*) > \epsilon\right\} \\ &\quad < 32\left(\frac{n}{\epsilon^2}\right)^{(1+\mathbf{B}/\epsilon)\log_2(4\epsilon/\mathbf{B})} ne^{-\epsilon^2 n/(2\mathbf{B})^2}. \end{aligned}$$

This bound provides qualitative insights into this approach when a "sufficient" number of measurements are available. The exponential term on the right-hand side decays faster in n than the growth in other terms, and hence for sufficiently large n it can be made smaller than a given probability α. Thus, the expected error $I(\hat{R})$ of the response mean used in the previous subsection is within ϵ of the optimal error $I(f^*)$ with a probability that increases with the number of observations. Furthermore, this performance guarantee is distribution-free, i.e., independent of the underlying joint distributions of controllers and switches, and is valid under very general conditions [17] on the variations of TCP throughput or ping RTT estimates.

5 CONCLUSIONS

We presented SDN solutions to set up dedicated connections needed for multi-site high-performance scientific workflows. They utilize site-service daemons to coordinate these connections, using dpctl, ODL, Floodlight, or ONOS controllers. The VSNE consisting of VMs, Mininet, and custom scripts emulates various network environments and supports the development and testing of SDN solutions while the multi-site physical infrastructure is being built. We proposed the switching response method to assess the connection setup performance of these SDN solutions. The SDN solutions are directly transferable to physical networks, but the performance test results require complementing VSNE with a physical testbed. While ping measurements are consistent, TCP traces show significant differences between the emulated and physical connections. Targeted tests using both VSNE and physical testbed provide insights into the performance of various SDN controllers, devices, and technologies.

It would be of future interest to investigate the correlations between TCP dynamics observed in physical and Mininet environments. More generally, it would be useful to gain further insights into the limitations of VSNE performance tests. Future work may also include the development of a baseline test harness for science environments wherein a controller or a switch can be plugged into a known, fixed science configuration to assess the performance. In particular, the switching responses can be generated for different combinations of controllers, switches, and other SDN components under various configurations to objectively evaluate their relative performances.

ACKNOWLEDGMENTS

This work is funded by the High-Performance Networking Program and the Applied Mathematics Program, Office of Advanced Computing Research, U.S. Department of Energy, and by Extreme Scale Systems Center, sponsored by U. S. Department of Defense, and performed at Oak Ridge National Laboratory managed by UT-Battelle, LLC for U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Floodlight. http://www.projectfloodlight.org.

[2] ONOS. http://www.onosproject.org.

[3] Opendaylight. www.opendaylight.org.

[4] OSCARS: On-demand Secure Circuits and Advance Reservation System. http://www.es.net/oscars.

[5] Software defined networking for extreme-scale science: Data, compute, and instrument facilities. DOE ASCR Workshop, Bethesda, MD, Aug. 2014.

[6] M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[7] Energy Sciences Network. http://www.es.net.

[8] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[9] Y. D. Lin, D. Pitt, D. Hausheer, E. Johnson, and Y. B. Lin. Software-defined networking: Standardization for cloud computing's second wave. *Computer*, 47(11):19–21, Nov. 2014.

[10] N. S. V. Rao. SDN solutions for switching dedicated long-haul connections: Measurements and comparative analysis. *International Journal on Advances in Networks and Services*, 9(3-4), 2016.

[11] N. S. V. Rao, S. E. Hicks, S. W. Poole, and P. Newman. Testbed and experiments for high-performance networking. In *Tridentcom: International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, 2010.

[12] N. S. V. Rao, W. R. Wing, , S. M. Carter, and Q. Wu. Ultrascience net: Network testbed for large-scale science applications. *IEEE Communications Magazine*, 43(11):s12–s17, 2005.

[13] C. E. Rothenberg, R. Chua, J. Bailey, M. Winter, C. N. A. Correa, S. C. de Lucena, M. R. Salvador, and T. D. Nadeau. When open source meets network control planes. *IEEE Computer*, pages 46–53, November 2014.

[14] S. Scott-Hayward, S. Natarajan, and S. Sezer. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18(1):623–654, First Quarter 2016.

[15] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit. SDN and OpenFlow evoltion: A standards perspecitve. *IEEE Computer*, pages 22–29, November 2014.

[16] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[17] V. N. Vapnik. *Statistical Learning Theory*. John-Wiley and Sons, New York, 1998.

[18] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. A survey on software-defined networking. *IEEE Communication Surveys & Tutorials*, 17(1):27–51, First Quarter 2015.