Virtual Environment for Testing Software-Defined Networking Solutions for Scientific Workflows

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

Joshua M. Boley, Rajkumar Kettimuthu
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL, USA
{jboley,kettimut}@anl.gov

Bradley W. Settlemyer, Hsing-Bung Chen
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, NM, USA
{bws,hbchen}@lanl.gov

Dimitrios Katramatos
Computational Science Initiative
Brookhaven National Laboratory
Upton, NY, USA
dkat@bnl.gov

ABSTRACT
Recent developments in software-defined infrastructures promise that scientific workflows utilizing supercomputers, instruments, and storage systems will be dynamically composed and orchestrated using software at unprecedented speed and scale in the near future. Testing of the underlying networking software, particularly during initial exploratory stages, remains a challenge due to potential disruptions, and resource allocation and coordination needed over the multi-domain physical infrastructure. To overcome these challenges, we develop the Virtual Science Network Environment (VSNE) that emulates the multi-site host, storage, and network infrastructure using Virtual Machines (VMs), wherein the production and nascent software can be tested. Within each VM, which represents a site, the hosts and local-area networks are emulated using Mininet, and the Software-Defined Network (SDN) controllers and service daemon codes are natively run to support dynamic provisioning of network connections. Additionally, Lustre filesystem support at the sites and an emulation of the long-haul network using Mininet, are provided using separate VMs. As case studies, we describe Lustre file transfers using XDD, Red5 streaming service demonstration, and an emulated experiment with remote monitoring and steering modules, all supported over dynamically configured connections using SDN controllers.

CCS CONCEPTS
- Networks → Network services; Network experimentation;

KEYWORDS
Software-defined infrastructure, software-defined networking, virtual science network environment, scientific workflows.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

AI-Science’18, June 11, 2018, Tempe, AZ, USA. © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-5862-0/18/06 ... $15.00.
https://doi.org/10.1145/3217197.3217202

1 INTRODUCTION
Scientific workflows are supported by an infrastructure of scientific instruments, supercomputers, storage systems, and custom visualization facilities [1] that are located at geographically dispersed sites and are connected over high-performance networks. To accomplish extreme-scale science by utilizing such an infrastructure, scientific workflows often require large data transfers and remote monitoring, streaming, and steering operations. These tasks in turn require dedicated high-capacity, low-latency, and low-jitter network connections during specified periods. Such connections are typically composed of both site Local-Area Network (LAN) segments and Wide-Area Network (WAN) connections over a dedicated high-capacity network infrastructure for data flows, as well as a persistent shared network among these sites for control traffic. Currently, the dedicated connections are provisioned manually by site and WAN operators, with typical lead times of days or longer. However, recent developments in Software-Defined Infrastructure (SDI) promise advanced capabilities such that end-systems and network paths can be composed and orchestrated entirely by software at unprecedented speed and scale. In particular, Software-Defined Networking (SDN) technologies [9, 25] enable automatic provisioning of these dedicated connections with much faster setup times, for example, within a few seconds.

The development of networking software modules for scientific workflows and an assessment of their impacts often require installation and evaluation of untested technologies. Performing these tasks over production-grade physical infrastructure remains extremely challenging, since it requires substantial resource allocations and often labor-intensive coordinations among site and network operations teams. For early functionality tests and proof-of-principle demonstrations, such expenses are not justified, particularly, if the
Challenges, we develop the *Virtual Science Network Environment* (VSNE) that emulates the host, storage, and network infrastructure of multiple sites using Virtual Machines (VMs). A site is represented by a VM, wherein the hosts and LANs are emulated using Mininet, and SDN controllers and site service daemons are executed natively to support dynamic provisioning of network connections. In addition, Lustre filesystems are supported at these sites using a server VM, and the long-haul network connections are emulated using Mininet on WAN VM. Applications such as file transfer, streaming, and experiment steering, can be installed on VMs and made available to all emulated site hosts.

Thus, without the expensive physical infrastructure, VSNE can be run on a workstation to develop and test various software components. In collaborative projects, VSNE can be replicated at all participating sites so that their respective solutions can be independently developed and tested, and uploaded to other sites. Upon maturity, these codes can be rolled into production physical infrastructure. Therefore, VSNE provides several advantages: (i) it enables early testing of workflow components, such as SDN scripts and applications, which could be potentially disruptive; (ii) it does not require the expenses of multi-site collaboration and physical resources; and (iii) solutions can be independently and concurrently developed before or while the physical infrastructure is being built. In particular, the VSNE described in this paper emulates the infrastructure that is currently being built to span four Department of Energy (DOE) national laboratories, namely, Argonne National Laboratory (ANL), Brookhaven National Laboratory (BNL), Los Alamos National Laboratory (LANL), and Oak Ridge National Laboratory (ORNL).

In addition to developing VSNE, we also present SDN codes to set up dedicated connections among the sites using a network of site-service daemons that are connected over the shared network. These daemons utilize custom scripts and northbound controller interfaces to coordinate the setup and teardown of dedicated network links. Once a connection request is received from the users, a site daemon translates and communicates subsequent requests to other site- and WAN-daemons, which are in turn translated into commands for setting up the site network elements. We have considered SDN solutions using custom dpctl scripts and open-source controllers including OpenDaylight (ODL) [14], Floodlight [7], and ONOS [15] in our previous study [19, 20] and investigated the TCP transport dynamics with these controllers.

We describe three scientific use cases using VSNE with dedicated connections provisioned using the above-mentioned SDN solutions. First, we illustrate Lustre file transfers using XDD [24] over dynamically provisioned connections. Second, we demonstrate the use of Red5 streaming framework [21] for data-streaming applications, which shows its applicability to scientific workflows. Third, we present an emulated experiment in which remote monitoring and parameter steering capabilities are demonstrated. These use cases illustrate that such workflows are compatible with dynamically SDN-provisioned network paths, so that further development and detailed performance testing can be carried out, possibly, over the physical infrastructure. In addition to establishing the utility of VSNE for functionality tests, these tests also reveal some limitations in accurately emulating the network transport dynamics.

The organization of this paper is as follows. In Section 2, we provide two examples of scientific workflows to motivate the needed SDN capabilities. In Section 3, we describe VSNE in detail, including its capabilities, a site-service daemon framework that connects the elements together, and its implementation using VMs and Mininet. In Section 4, we present the use cases of XDD file transfers, Red5 streaming framework, and remote instrument monitoring and steering. The paper concludes in Section 5.

## 2 SCIENCE SCENARIOS

Scientific workflows are realized by composing and automating complex scientific applications to enable collaborations among researchers in many disciplines, such as biology, astronomy, environmental science, materials science, nuclear science, among others [1]. By using flexible high-level abstraction languages, these workflows can mask the complexity of execution infrastructure, and allow scientists to execute simulations on remote systems, retrieve data from an instrument or database, process the data, and run data analysis tools, while automating data movement between various stages of the workflow processing [5]. We present below a brief overview of two representative categories of scientific workflows, and discuss the challenges associated with state-of-the-art networking and services.

### 2.1 Scientific Workflow Drivers

**A. Workflows for Near-Real-Time Computations:** Data generated at an instrument system, during or after the experiments, are often transported to a remote supercomputer site for near-real-time analysis and computation. For example, in cosmology, the raw data generated by the Palomar Transient Factory (PTF) [10] survey are processed by a near-real-time computational code at a remote national laboratory to identify optical transients within minutes of images being taken. Similarly, in material science, to facilitate near-real-time analysis of organic photovoltaics (OPV) using x-ray scattering, Lawrence Berkeley National Laboratory (LBNL) Advanced Light Source (ALS) data need to be moved to ORNL because currently only Titan has the computational capability to run the required analysis tool HipGISAXS [2].

**B. Workflows for Dynamic Monitoring and Control:** Data generated from a running computation and/or an instrument often need to be dynamically monitored at local or remote facilities to understand whether the simulation/experiment is functioning properly. Analysis and control of these intermediate datasets are critical to drive the next simulation/experiment configurations at various science facilities [1], including ALS, Spallation Neutron Source (SNS), Large Synoptic Survey Telescope (LSST), and others.

### 2.2 State-of-the-Art Approaches and Challenges

Both workflows mentioned above require data movement, streaming, and control operations to be supported over wide-area networks, in particular, by on-demand/advance-reserved dedicated
connections with high bandwidth and low jitter. Currently, custom-designed science network connections are typically composed and configured by teams of experts. For example, LAN and WAN connections are set up by network engineers, and dedicated I/O resources and host systems are configured by systems administrators. Multiple valuable resources are often over-provisioned to meet the peak transient needs. Thus, although highly desired, many promising capabilities, including on-demand computation/instrument monitoring, interactive steering, etc., are not implemented. In the future, the hosts, storage, and networks are expected to become increasingly sophisticated, and the number of possible combinations of parameters to be optimized for a complex scientific workflow will increase exponentially, and will be beyond the limits of manual operations. Recent developments in SDN and related technologies [9, 11] hold an enormous promise in developing fast automatic provisioning of the underlying network paths. However, the testing of new components of workflows, such as XDD optimized for Lustre and custom Red5 streaming apps for science, requires multi-site networked infrastructure, and our VM-based VSNE provides an alternative development environment for them.

Moreover, it is to be noted that the network flows associated with science applications represent a different set of challenges from the data center and cloud environments, where the current SDN technologies are being developed. The predominant feature of scientific workflows is that they are small in numbers originating from known sites, and they involve dedicated, precision flows over multi-domain wide/local/storage area networks; these features are supported in our VSNE using VMs and Mininet. In contrast to other VM-based projects, such as OpenGENI [16] for networked OpenStack components, Chameleon [4] for cloud infrastructure, Jetstream [8] incorporating both, ViNO orchestration service [3] for creating network topologies, and DOT [22] for low cost and scalable network emulation, our VSNE is specifically tailored to science environments by using Mininet to emulate site hosts and networks, with additional support for Lustre filesystem and a suite of controllers, as described in the next section.

3 VIRTUAL SCIENCE NETWORK ENVIRONMENT

In this section, we provide a detailed description of the virtual science network environment (VSNE), including its functionality and implementation. Particularly, we highlight its capability to emulate the network and storage infrastructure in real-world scientific workflow applications. The site-service daemon framework is shown to facilitate persistent control-plane connections. Virtual machine configurations, Mininet network topologies, and the interconnected site service daemon framework are presented for a four-site VSNE.

3.1 VSNE Capabilities

Our VSNE enables early testing of SDN solutions (some of which are potentially disruptive) before or while the physical infrastructure is being built. In particular, a number of virtual site hosts, switches, and network links are created on a virtual machine (VM) where custom applications and filesystems are also installed and made available to every virtual site host, since the latter can inherit the execution environment from the site VM where it resides. Our functionality testing examples to be presented in Section 4 are run from the perspective of a site host. For storage, we support the Lustre filesystem [12] on the site hosts by utilizing a Lustre server on a dedicated VM. Site hosts run Lustre clients to mount the filesystem and utilize assigned sub-directories for individual sites. The Lustre VM is connected to site VMs over an internal network which represents the site storage network. Each site VM contains hosts and LANs emulated using Mininet, and the site VMs are interconnected using a WAN VM. SDN controllers and service daemons support dynamic provisioning of network connections, whose details are discussed below.

3.2 Site Service Daemon Framework

In our SDN solution, a set of site-service daemons provide connectivity among the sites over the default IP network. These site-service daemons maintain persistent connectivity among themselves and also with the local site-controllers, switches, and users, as shown in Figure 1. The local site-service daemon receives connection requests (setup/teardown) from local users, automated workflow agents, or remote site-service daemons. In response, it invokes custom scripts to set up or tear down connections within the site, and generates specifications for WAN and remote site connections, and communicates them to WAN and remote site-service daemons. We show an illustration of two-site path setup in Figure 2. While the SDN controllers communicate with switches to install, query, and delete flow entries on their southbound interfaces, the site-service daemons talk with SDN controllers via the northbound interface. Examples of using these service daemons are shown in Section 4.

3.3 Virtual Machine Configurations

We use a total of six VMs, four of which represent the sites, namely, ANL, BNL, LANL and ORNL, and a fifth VM emulates the dedicated ESN (E) WAN connections among these sites, as shown in Figure 3. An additional VM provides the Lustre filesystem to be mounted onto all four sites. The VMs run under the VirtualBox 5.1 environment on a Linux host with RHEL 7.2 kernel. For each site VM, three interfaces are enabled – Network Address Translation (NAT) interface and two internal interfaces. The host OS re-routes

\[ \text{Figure 1: Framework of interconnected site daemons} \]
and re-sends all the data sent from the guest VM via the NAT interface, the control-plane communications among all the sites using the site daemon codes are executed via one internal interface, and the dedicated data-plane connection among the site VMs and WAN VM is implemented via the other internal interface.

### 3.4 Network Emulation

We use Mininet [13] to create custom parameterized topologies on site and WAN VMs using Python code. Figure 4 illustrates the configuration on any of the four site VMs. We create two virtual hosts $h_1$ and $h_2$ that are connected to a virtual switch $s_2$; then, another “gateway” switch $s_1$ serves to link $s_2$ and the outside world. Both $s_1$ and $s_2$ are Open vSwitches whose flows can be dynamically orchestrated as needed; these switches represent the deployed OpenFlow hardware switches, and in particular, they support both dpctl and open-source controllers so that VSNE codes can be transferred to physical networks. Also shown as components on the site VM in Figure 4 are an SDN controller and a site-service daemon, where the latter is used for communications of control-plane messages with other sites.

On the WAN VM, four Open vSwitches are created in Mininet that emulate the physical circuits – for example, similar to those in OSCARS [17] – required to access the long-haul links among the sites. Figure 5 illustrates both the data-plane connections among the switches and to the outside world, and the control-plane connections involving the controller and/or the WAN daemon. In addition, we incorporate the actual long-haul link latency between physical sites in the Mininet environment, by imposing various delay parameters, between the “border router” site-pairs in the Python script. For instance, the one-way latency between ANL and ORNL is set to be 6 ms. A simple rate control mechanism is also implemented in the same script where the maximum link-bandwidth is set to be 20 Mbps.

### 3.5 Path Request via Science User Web Interface

We design a web2py-based [23] web service stack that is integrated into the site-service daemon framework so that an end user can directly send datapath reservation requests (to another remote host) via the client web interface, as shown in Figure 6. The web2py stack and the daemon framework are tightly integrated so that the user request on the web is redirected to the daemon as CLI requests. Once the requested path is successfully reserved and set up later, we can run simple `ping` and `iperf` commands to confirm the connectivity between the two remote hosts.

### 4 USE CASES

In this section, we develop three use cases by utilizing the VSNE framework described in Section 3. The first case is a simple XDD file transfer from one site host to another site host located on a different VM, using the Lustre filesystem. In the second case, we test the viability of Red5 open-source streaming framework that supports a
Then, we can see the details of this file:

```
> ls -l /mnt/knotfs/anl_lustre_dir
 total 1
-rw-r--r-- 1 root 5242880 Jun 14 12:29 test_file_anl
> ls -l /mnt/knotfs/bnl_lustre_dir
 total 1
-rw-r--r-- 1 root 5242880 Jun 14 12:33 test_file_anl_copy
```

The file-transfer then starts if the datapath between the host pair has been set up; once complete, we can run the following command on h2_bnl to verify the file has indeed been transferred successfully:

```
> ls -l /mnt/knotfs/bnl_lustre_dir
 total 1
-rw-r--r-- 1 root 5242880 Jun 14 12:33 test_file_anl_copy
```

Detailed information about the transfer is also displayed at both the sender and receiver. The outputs are rather long, but the last few lines show some of the most important metrics of interest, as displayed in Figure 7, including the transfer size, elapsed time, and bandwidth/throughput. Interestingly, here both file read- and write-rates are under 1 Mbps, much slower than the link capacity set as 20 Mbps.

### 4.2 Streaming Applications

The open source Red5 streaming framework supports a wide variety of streaming applications, for example, chat, video, and data streaming applications over the Real Time Messaging Protocol (RTMP) supported over TCP. While extensively used for streaming applications over a variety of platforms, its use in scientific workflows has been very limited. This framework has a potential for leading to effective streaming applications tailor-made for science collaborations; however, its effectiveness over dynamically provisioned networks has not been well established. We now show its viability by demonstrating a scenario where users at different sites dynamically set-up a virtual collaborative meeting to exchange messages and stream images over a dedicated science network infrastructure.

Consider a workflow involving computations run on a supercomputer, whose outputs are sent to a remote visualization site and its images are streamed to the team to discuss and arrive at parameters for subsequent computations. Dedicated network connections with appropriate capacity and latency will be needed between the sites to match the workflow requirements. This workflow is demonstrated between ORNL and BNL by initiating Red5 server on ORNL VM to support both text messaging (chat) and video streaming applications on browsers at both sites. The BNL-ORNL connections are set up between hosts that use flash-enabled browsers to connect to the server, and the video streaming app is launched at both ends. Whereas the ORNL app is connected locally to Red5 server, the BNL streaming app is connected over the provisioned data-plane connections. The screenshots in Figure 8 show the media files stored on ORNL VM streaming to both sites. And, these can be cooperatively analyzed using the concurrent chat application.

The above video streaming use case requires only the installation of the Red5 server (under an hour) and no additional coding. They demonstrate the ease of deployment and effectiveness of the Red5 app for the science environment, thereby providing a quick proof of the concept. Several other streaming tasks of scientific workflows may be supported by simple extensions or customizations of existing apps. For example, outputs from a running computational task can be posted to a simple chat window using curl scripts, thus transforming the chat interface into a simple computational monitoring app.
4.3 Instrument Monitoring and Steering

Our third use case captures the essence of scientific workflows associated with dynamic monitoring and remote steering of an instrument. Monitoring the intermediate results from an instrument is often essential to understanding the proper functioning of the running experiment. Similarly, at times, sending control commands to the instrument is required to steer the next experimental configurations. In the following, we demonstrate the implementation of instrument monitoring and steering operations from a remote site using our VSNE.

First of all, we need to emulate a scientific instrument by extending the Mininet environment of a particular site VM and modifying the python script that creates the required switches, hosts, and network links associated with the instrument. Specifically, we modify the ANL site VM to incorporate an emulation of APS (Advanced Photon Source) instrument, as shown in Figure 9. We consider that the instrument is connected to the switch $s_2$, and all the flow-paths have already been set up to route the generated experimental data from any of the three instrument hosts to switch $s_2$. Now, if this data stream is to be monitored from a remote host, we have to first establish a dedicated datapath from $s_2$ to that particular host. For demonstration purposes, we consider that instru1 is in operation, and we set up paths from $s_2_{anl}$ to $h_2_{ornl}$ by adding appropriate flows in ANL, WAN, and ORNL switches such that $h_2_{ornl}$ can remotely monitor instru1.

The instrument data flow is emulated by running TCP iperf on the instrument host and the monitoring host. A script, instr_iperf, on the instrument hosts emulates non-contiguous data flows, broken (as desired) by periods of no traffic, as

```
./instr_iperf.sh -c [ip] -i [i1 i2 ...] -t [intrv]
```
Figure 10: Instrument monitoring and steering operation details

Figure 10 depicts in detail one illustrative example of instrument monitoring and steering. To start with, ANL’s instru1 host initiates the iperf transfers at alternate intervals with intrv = 30 seconds as

```
./instr_iperf.sh -c <h2_ornl> -i 0 2 4 6 8 10 -t 30
```

This is highlighted with an orange bubble in Figure 10(a), and the corresponding iperf at the monitoring host, h2_ornl, is also shown with an orange bubble in Figure 10(b). Consequently, we notice on instru1 that a transfer starts and lasts for 30 seconds (shown by an orange arrow), and on h2_ornl we observe the received data stream.

In Figure 11, we plot these received iperf data (averaged over 10

Next, to emulate the remote instrument steering mechanism, we send different values of intrv (as emulated control commands) from the monitoring host to the instrument, which in turn uses those newer intrv values in subsequent iperf transfers. We utilize the default IP network among the sites to convey the newer intrv values from the monitoring site VM to the instrument site VM as

```
./instrument_control.sh <instr-vm-ip> <new-intrv>
```

where instr-vm-ip is the IP of the instrument site VM, and new-intrv is a new time-duration value to be used by the instrument.

Figure 10 depicts in detail one illustrative example of instrument monitoring and steering operation workflows. To start with, ANL’s instru1 host initiates the iperf transfers at alternate intervals with intrv = 30 seconds as

```
./instr_iperf.sh -c <h2_ornl> -i 0 2 4 6 8 10 -t 30
```

This is highlighted with an orange bubble in Figure 10(a), and the corresponding iperf at the monitoring host, h2_ornl, is also shown with an orange bubble in Figure 10(b). Consequently, we notice on instru1 that a transfer starts and lasts for 30 seconds (shown by an orange arrow), and on h2_ornl we observe the received data stream.

In Figure 11, we plot these received iperf data (averaged over 10
repetitive runs) as a function of time, and an orange arrow with -t30 indicates the specified time duration value.

While this first iperf transfer is ongoing, the monitoring host decides to send a control command to the instrument site for changing the time-duration value to 5 seconds. This is communicated from the ORNL site VM as

```
./instrument_control.sh <ANL-site-vm-ip> 5
```

This is marked by a red box in Figure 10(b), and the corresponding acknowledgment at the ANL site VM is also marked by a red box in Figure 10(a). In Figure 11, we use a red arrow to mark the time instance of this new control command with -t5. As a consequence, in the next transfer interval, instrul starts a new iperf transfer that lasts only 5 seconds, which is evident from a thinner profile of iperf transfer in Figure 11.

We continue this monitoring and steering procedure by sending two more control commands for modifying the time-duration parameter to 20 and 10 seconds, which are respectively indicated by green and yellow boxes/arrows in Figures 10 and 11. Subsequent iperf transfers from instrul show that these steering parameters are correctly received and used (see green and yellow arrows in Figure 10(a)). The averaged iperf trace in Figure 11 also confirms the proper use of the most-recent time-duration parameter in the following iperf transfers. As no further change is made to the time-duration value after -t10, each of the last three iperf transfers continues for about 10 seconds (see Figures 10(a) and 11).

It is to be noted here that the alternate transfer and non-transfer intervals used in this example is not a fixed scheme; our instrul_iperf script allows for any generic specification of transfer intervals. For example, in Figure 12, we show the iperf profile of another example of instrument monitoring and steering, which is initiated from instrul as

```
./instrul_iperf.sh -c <H2_ornl> -i 1 2 4 6 9 10 -t 30
```

and then steered by ORNL site VM by sending three new values of the time-duration parameter as 5, 20, and 10. At the beginning of this transfer, we notice a wider iperf profile (see Figure 12), because there is no non-transfer interval in between the first two consecutive transfer intervals, respectively with -t30 and -t5 parameters. Similarly, the last two transfer intervals indexed with 9 and 10 (each with -t10 specification) occur back-to-back, and as a result another wider iperf profile is observed toward the end.

5 CONCLUSIONS

We presented a Virtual Science Network Environment (VSNE) for wider iperf profile is observed toward the end.

```
This is marked by a red box in Figure 10(b), and the corresponding acknowledgment at the ANL site VM is also marked by a red box in Figure 10(a). In Figure 11, we use a red arrow to mark the time instance of this new control command with -t5. As a consequence, in the next transfer interval, instrul starts a new iperf transfer that lasts only 5 seconds, which is evident from a thinner profile of iperf transfer in Figure 11.

We continue this monitoring and steering procedure by sending two more control commands for modifying the time-duration parameter to 20 and 10 seconds, which are respectively indicated by green and yellow boxes/arrows in Figures 10 and 11. Subsequent iperf transfers from instrul show that these steering parameters are correctly received and used (see green and yellow arrows in Figure 10(a)). The averaged iperf trace in Figure 11 also confirms the proper use of the most-recent time-duration parameter in the following iperf transfers. As no further change is made to the time-duration value after -t10, each of the last three iperf transfers continues for about 10 seconds (see Figures 10(a) and 11).

It is to be noted here that the alternate transfer and non-transfer intervals used in this example is not a fixed scheme; our instrul_iperf script allows for any generic specification of transfer intervals. For example, in Figure 12, we show the iperf profile of another example of instrument monitoring and steering, which is initiated from instrul as

```
./instrul_iperf.sh -c <H2_ornl> -i 1 2 4 6 9 10 -t 30
```

and then steered by ORNL site VM by sending three new values of the time-duration parameter as 5, 20, and 10. At the beginning of this transfer, we notice a wider iperf profile (see Figure 12), because there is no non-transfer interval in between the first two consecutive transfer intervals, respectively with -t30 and -t5 parameters. Similarly, the last two transfer intervals indexed with 9 and 10 (each with -t10 specification) occur back-to-back, and as a result another wider iperf profile is observed toward the end.

5 CONCLUSIONS

We presented a Virtual Science Network Environment (VSNE) for early testing of SDN functionalities for multi-site scientific workflow applications without the need for immediate deployment of the physical infrastructure. VSNE utilizes virtual machines, Mininet topologies, custom scripts with dpctl or SDN controllers, and the site service daemon framework to coordinate among various sites. We also described three use cases to demonstrate the viability of the VSNE, where the software solutions are directly transferable to physical networks.

It would be of future interest to further investigate the VSNE performance by incorporating it into a physical testbed with actual hosts, OpenFlow switches, and long-haul links. This will allow us to explore the correlations between performance metrics observed in physical and virtual environments so that more insights can be gained into the limitations of performance tests conducted under the VSNE environment. Future work may also include the development of other proof-of-the-principle functionality tests as well as a baseline test harness for scientific workflows wherein a controller and/or a switch can be plugged into a known, fixed configuration to assess their performances under various settings.

Acknowledgments

This work is funded by the High-Performance Networking 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


