The USENIX Journal of Education in System Administration
Volume 2, Number 1 • November 2016
www.usenix.org/jesa/0201

Software Defined Networking for Systems and Network Administration Programs

Bruce Hartpence, Rochester Institute of Technology
Rossi Rosario, Rochester Institute of Technology

Academic programs can be very successful when they include industry best practices, innovations and techniques in addition to theory and background. This approach has historically been a tenet of the networking and systems administration program at the Rochester Institute of Technology. Software-defined networking is an excellent example of a technology which combines theory and emerging practice. Software Defined Networking or SDN includes components that stretch across networking and systems administration curricula including servers or controllers, virtualization, OpenFlow enabled network elements, communication pathways, opportunities for automation, telemetry from the network, dynamic response to system demand and more.

These characteristics, and because SDN experiments and courses can be implemented in either virtual or non-virtual facilities, make SDN an outstanding platform for teaching the principles of network and systems administration. Graduate students can also take advantage of the environment encompassed by SDN topologies to further their understanding of systems design, management, testing and communication protocols. This paper will describe some of the SDN projects run at the Rochester Institute of Technology (RIT), the impact on curriculum and some of the environments used. The challenges associated with running the projects and courses within a lab environment will also be illustrated. How and why many of the ideas and new industrial developments were integrated into the classroom will be central to the ideas presented.

1. INTRODUCTION

Understanding what is meant by the term Software Defined Networks (SDN) is not always a simple task as the term has been applied to so many different ideas. For the purposes of this paper, Software Defined Networking will be used to describe an innovative approach to network construction, configuration and monitoring. While it is relatively new (most of the ground work was completed in Stanford circa 2008) it is transforming our thinking regarding communication architectures [McKeown et al. 2008]. SDN is a novel network architecture in that it provides a segmented design where the controlling entities of the network (control plane) are detached from the forwarding functionality (data plane), allowing for efficient and granular control over the data flow. As result, a logical centralization of the network is possible. This centralization is in the form of a controller that communicates with network elements which can simplify network management and improve visibility into the network. One improvement is that SDN provides a much more dynamic and informed view into the network via flow tables. SDN also provides a mechanism through which networks may be shared among different tenants, each having their own control over network switches and routers [Sherwood et al. 2010]. An important point is that the controller may be able to dynamically make changes to network elements based on feedback or code without human intervention.

These improvements are difficult to achieve in traditional networks. Traditional networking and systems administration refers to the static and laborious techniques for designing, managing and deploying network elements, servers and services. Examples include the need to interface with each element manually, attempting to determine traffic patterns via telemetry from servers or protocols, deploying virtual machines and applying security mechanisms.

SDN networks are logically segmented on three general regions: Application layer the management plane where the network programming portion takes place, Control Layer in which the controllers or network brains reside, and finally the forwarding devices or Openvswitches (OVS) - the nodes that execute the forwarding decisions according to the flows rules dictated by the controllers [Kreutz et al. 2015]. This is ilustrated in Figure 1. This concept breaks with traditional paradigms of network design and operation, opening up a whole new path for research exploration.

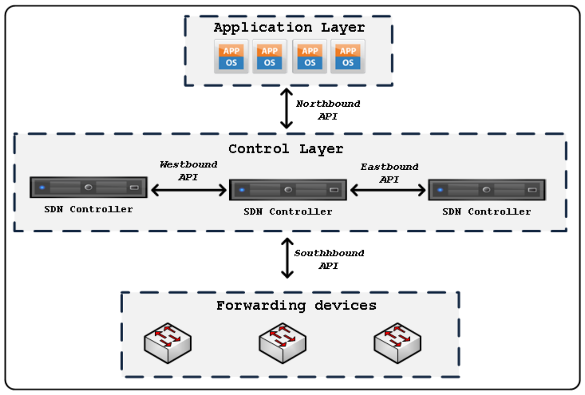


Fig. 1: SDN segmentation

The communication channel between the controller and the network elements (southbound interface) is made possible via OpenFlow protocol [Specification 2011]. OpenFlow is the first standardized communication interface for programmable networks. It was envisioned to provide an open and programmable platform in which flow table entries could be inserted and modified easily, without worrying about vendor specific configurations [McKeown et al. 2008]. Although it started just as the research vehicle for programmable networks, nowadays, most of the controllers support this communication protocol. Figure 2 depicts the OpenFlow channel between the controller and network element. SDN provides an excellent opportunity for educators to

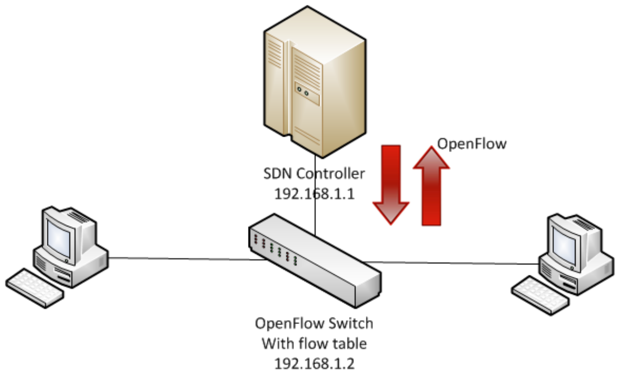


Fig. 2: OpenFlow Channel

bridge theoretical concepts and in lab experiences. This is not only because of the nature of SDN architectures but also because SDN (or the idea of being software defined with a controller) has expanded into areas such as storage (SDS), wide area networking (SD-WAN) and data centers (SDDC). SDN also leverages several other technologies such as virtualization thus integrating virtual and traditional networks. The separation of control and data planes are well documented in the literature as are the problems associated with traditional network design and operation. These problems extend into the administrator space because historically servers and the network have been unable to respond dynamically to changes or provide visibility into processes. With the creation of open source controllers, such as Pox, and the the ability to virtualize elements of the topology, these problems and their potential solutions can be deployed and observed in the lab.

While controller software such as NOX or POX [Gude et al. 2008] is open source and stable, making it relatively simple to work with, it does not represent the current state of SDN research or industry. Fortunately, the SDN structure is modular which allows the educator to easily shift from one to another. This is demonstrated in one of the projects described herein. Thus it is possible to integrate the latest changes from production work. In addition, central components such as Open vSwitch [Pfaff et al. 2015] can be held at a particular patch level or embrace the latest developer modification. The projects described in this paper will not only discuss their construction and results but some of the challenges associated with the management of resources. These challenges vary when the architecture is constructed on-site as opposed to in a remote facility such as the Global Environment for Network Innovation or GENI [Elliott 2008].

One can make the case that the software-defined approach can be applied to network and systems administration programs by simply looking at the components and skills necessary to deploy the architecture. However, innovative and practical use cases can be found in industry as well. Facebook is an organization that is willing to share some of its project work. The Facebook network, database and infrastructure teams were faced with not only complex problems associated with performance and network traffic but massive log files reporting hardware and software failures. To address these challenges, Facebook engineering created the Facebook Auto Remediation service or FBAR [Hoose 2011]. FBAR has the ability to automatically run code in response to these failures. The result is that the system adopts a software defined automation approach that saves on man hours, time and money. Examples like this show the interrelated nature of problems and how the software defined approach can used by multidisciplinary teams.

The rest of this paper is organized as follows; section 2 provides some background on standard network/sys admin coursework and how SDN integrates into these areas. Section 3 discusses SDN components and the lab environment used to support courses and labs deploying these topologies. Section 4 is an overview of two major projects run at RIT, their results and challenges. Section 5 will be a discussion of our experiences and the potential for related courses and labs. Section 6 concludes.

2. SYSTEMS/NETWORK ADMINISTRATION CURRICULUM TOPICS

As SDN is a relatively new topic, it is fair to question where it would fit and why it should be adopted. Information Technology (IT) and related programs commonly contain courses covering key ideas and best practices for the industry into which graduates will soon enter. The goals of this coursework include preparing students for their careers and instilling the ability to be lifetime learners and solve problems. That is, to be able to grab new technologies or ideas and incorporate them into current and new generation architectures.

RIT is no different. Having a core of networking and systems administration courses, we have also added courses that embrace trends in industry and capture important methodologies. SDN provides an excellent opportunity to further these ideals, especially when combined with virtualization. A brief description of our foundation courses follows. While these may read like a section of the course catalog, the shortened descriptions are included here to indicate what we consider to be core tenets of the curriculum and the additions necessary to keep current and provide a vibrant learning environment.

— Task Automation Using Interpretive Languages - An introduction to the Unix operating system and scripting in the Perl and Unix shell languages. Includes basic user-level commands, control and data structures in Perl.

- Systems Administration I - This course is designed to give students an understanding of the role of the system administrator in large organizations. The technologies discussed in this course include: operating systems, system security, and service deployment strategies.
- Systems Administration II - This course goal is to help students understand the roles and responsibilities of system administrators in modern enterprise organizations. The responsibility of the System Administrator is to maintain, manage, secure and monitor the network and the services it provides.

These first three comprise the core of the systems administration courses with the next group describing the network based coursework.

- Intro to Routing and Switching - This is an introduction to wired network infrastructures, topologies, technologies and protocols required for effective end-to-end communication. Basic security concepts are also introduced at the local area network communication level.
- Wireless Networking - This course is designed to provide the student with an understanding of the protocols, principles and concepts of radio and optical communication as they apply to wireless data networking for local area networks and peripherals.
- Network Services - An investigation of the tasks of selecting, configuring and administering services in an internetworking environment. Topics include the TCP/IP protocol suite, service administration including DHCP, DNS, SSH, Kerberos, and an introduction to directory services.

These brief descriptions mirror what is seen in many programs and the courses have been extended by a collection of courses designed to incorporate critical areas.

These newer courses also addressed a problem encountered by some of our incoming graduate and undergraduate students; some technologies have become such an integral part of our programs that students may not have had an adequate background before being expected to hit the ground running in a lab environment. The best example of this is virtualization. It is difficult to separate virtualization from networking or systems administration in most modern communication architectures. Similarly, we had integrated virtualization into many of our courses, assuming that students would understand what they were to do and how to use the tools. We discovered that this was not always the case. To address this challenge we created a virtualization course and adopted a full stack approach.

- Virtualization - This course will take the students through the evolution of virtualization. The course begins with virtual network topologies such as VLANs, trunks and virtual routing and forwarding. From there we examine the various host-based virtualization platforms, bare metal hypervisors, server virtualization, storage and cloud computing.
- Configuration Management - This course teaches students advanced techniques in the Perl language. Techniques include the use and construction of object oriented scripts, user administration and monitoring, file system walking and checking, and computer and network security issues.
- Storage Architectures - This course provides students with a theoretical as well as hands-on exposure to enterprise scale storage technologies such as storage area networks and network attached storage. Students will study SCSI, Fibre Channel, IP Storage, Infiniband, and Fibre Channel over Ethernet.
- Seminar in Software Defined Networking - Students will take traditional network architectures and, utilizing a combination of virtualized platforms, logical network topologies and emerging standards, will complete construction of a network based on controllers, OpenFlow and Open Virtual Switch (OVS).

Additions and modifications to the courses are made based on input from industrial advisory boards, careers fair participants, industry best practices and research projects. Software defined networking is a topic that regularly comes up as it embodies next generation structures along with so many of the needs we see today. These include an increased emphasis on tool creation, orchestration, visibility and integration or collaboration between elements.

This paper is not about the success or lessons learned from a particular course; these listings indicate our desire to incorporate aspects gleaned from research projects and interaction with industry partners. SDN serves as an excellent example of how this comprehensive architecture can provide students with not only excellent learning opportunities, but a forward looking research agenda. As controller and Openflow based SDN is an emerging technology, there are ample research topics and targets for graduate capstone projects.

2.1. The Sys and Net Admin relationship to SDN

Having discussed the content of the program, we might now investigate the relationship between SDN architectures and systems/network administration coursework. Why might it be a good fit? Many of our systems and network administration topics can be thought of as applied, meaning that they represent very real, job oriented tasks. But included in program goals is the need to provide the background, reasons and higher thinking associated with the practice. Similarly, a complete understanding of SDN requires that the researcher or student examine the traditional methods used in network operation and the problems that are created. Moving to the new paradigm means that we will be using new skills and thinking about the forwarding process in a different way. Several of the courses in this list were either created or updated as a direct result of our experiences with SDN topologies and an over-reliance on assumed skills.

Like so many areas within the industry, there is increase in the amount of scripting and device administration for normal operation. Examples include the call for greater automation and orchestration platforms. Networking tasks are also becoming more systems or server administration like. In addition, the separation of the various planes and the virtual nature of elements requires that we understand more of the reasons as to why something is done and the under-the-hood operation. SDN is a multifaceted, comprehensive architecture that can be multipurpose and complex. Operating an SDN architecture requires that nearly the entire collection of systems administration skills be brought to bear.

As discussed previously, SDN architectures include both hardware and software components. While vendors continue to claim that SDN can still be managed using mechanisms that are similar to the console and command line interfaces so common to routers and switches, it is hard to avoid the realization that working with virtualized elements, many of which are open source, is very like the tasks we see in systems administration. Management of the controller and the virtual environment are very similar to running any other server. In fact, many network devices behave and operate like hypervisors. As will be discussed later in this paper, running an SDN research or test project is very much like a development environment in that the connections and nodes are remote and virtual. In addition, the resources supporting the SDN build are a practical implementation of our systems administration topics. Several examples can be found in configurations and revision history, the use of git repositories, interaction with the open source community, management of the hypervisor software and chassis, patch or updates to the systems, software installs to the virtual machines and desktops, automation or scripting of tasks, processing of data, managing SSH connections and keys, performance testing, user accounts and access.

If we use the examples from Facebook, is an integration between teams and ideas. While we might not choose to convert a routing course to controller based SDN, we might choose to adopt management techniques and dynamic configuration as a part of the class. In systems administration we see tasks such as log file growth and monitoring of servers or intrusion detection systems taking up a greater percentage of our time. So, we choose to adopt automation to address a subset of the task to free up valuable human resources. The FBAR system is full of ideas overlapping other areas that we have also chosen to adopt. These include Devops and orchestration tools such as

Kupernetes [Bernstein 2014] and Chef.

SDN offers several benefits over the traditional network architecture but it does comes with some added complexity and the potential to demand a greater level of programming and management. Someone, local or remote, must build, provide and support SDN projects. Add to this that the elements are running in virtual machines and we see an increased set of administrative tasks. From this point of view, SDN architectures and peripheral components dovetail very nicely with standard systems and network administration curricula because the students and faculty can be directly engaged with an architecture requiring all of their skills and knowledge.

3. IMPLEMENTING SDN PROJECT AND COURSE WORK

In order to experiment with SDN architectures, a project might start with a couple of very basic topologies. Perhaps the easiest would be using Mininet which is a self-contained SDN environment that runs in a virtual machine [Lantz et al. 2010]. Mininet enables the construction of a wide variety of SDN topologies from the command line interface. Mininet can also be scripted facilitating expansive testing and experimentation. However, this self-contained structure limits student and faculty exposure to actual operating network elements, even though Mininet constructs can be accessed via the command line. Thus, all links and traffic are simulated. For our purposes, the decision was made to use actual physical and virtual SDN deployments in projects and courses. The first project described in section 4 was built by a graduate student for her capstone. The second was built by faculty as a test case for coursework and research projects. As a result of these successes, further research was promoted and several components were assigned to students for course laboratory builds.

3.1. Physical and Hybrid Topologies

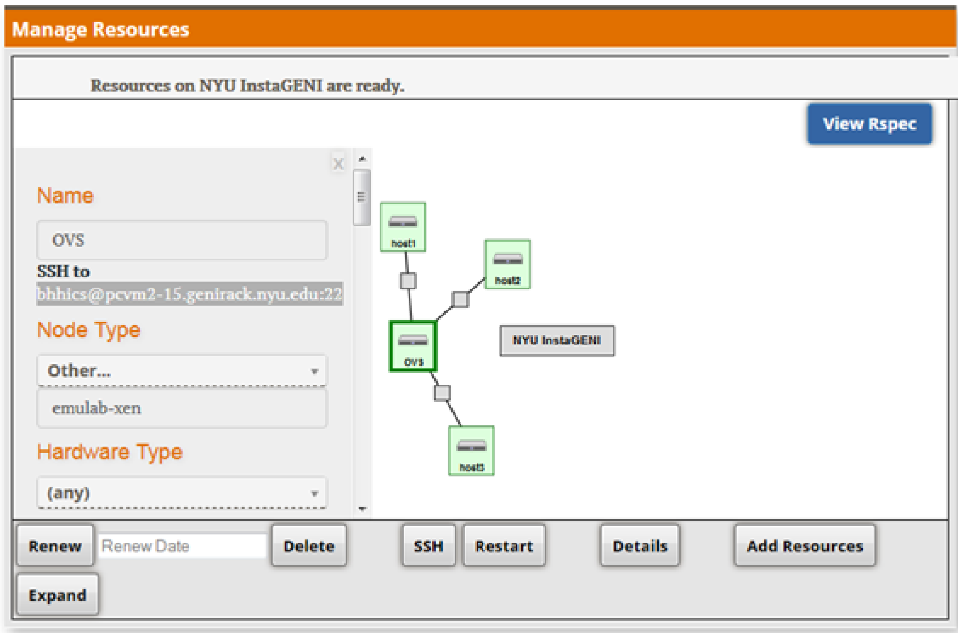
As one might imagine, the construction of a physical topology requires a greater commitment of resources. In the base topology, this would require four physical boxes; two for network nodes, one for the controller and one for the network switch. These computers need not be powerful and in fact, our initial topologies were built with cast off equipment from the department. We started with physical hardware only and allocated four machines per SDN topology. Other than the need for machines, the central node required 3-4 network interface cards, depending on the management interface configuration. For example, the computer running openvswitch (OVS) was connected to all three of the other nodes but eventually a fourth management connection was added. The management connections allowed remote configuration and patch updates. From an educational point of view, building a physical topology is rewarding because the student or research must touch and configure everything. Problems, configuration or otherwise, must be solved by the project members. In addition, results, network traffic, server needs and the devices themselves were immediately accessible. From a project management perspective, space becomes a real concern.

As with any physical deployment, heat and ventilation can also be a concern. Over the course of this work, there were times when the entire experiment had to be moved to an office space that under normal conditions has sufficient environmental controls. However, the equipment shown in Figures 4 and 7 can give off a surprising amount of heat and noise. Temperature in the space went above eighty-five degrees and the noise made working in the space untenable.

As we will see later in this paper, some of these can be virtualized resulting in a hybrid topology. Even with the virtual infrastructure, the hypervisor chassis, any external nodes and the cabling are all tangible items that must be allocated. In this second variation, only a single computer has to be allocated, alleviating many of the space and resource concerns. However, this box should have a minimum of 8GB of RAM and a hard drive sufficient for stored install media and virtual machines. In this configuration the network nodes, controller and OVS are running in virtual machines. The computer will be the hypervisor chassis which adds an additional systems administration task. Another externally connected management node is also required although this can be the researcher or student laptop. In our build we used VMWare ESXi 5.5 and vSphere for the hypervisor and management components.

3.2. Virtual Topology

It is possible to run the entire project at a remote site which removes the need for physical equipment. The only local requirement would be an access or management node. GENI is an engineering and educational environment that allows students to create any number of virtual machines and their associated connections. Each virtual machine has an external management connection. GENI is a fully scriptable environment that also allows the installation of additional software. Those familiar with Planetlab, Emulab or Amazon Web Services based virtual machines will have a good mental image of the system. We have used GENI virtual machines and connections in a wide variety of activities including capstones, small experiments and even running entire courses. An example of the GENI interface used by researchers and students can be seen in Figure 3.


Manage Resources

Resources on NYU InstaGENI are ready.

Name: OVS

SSH to: bhhics@pcvm2-15.genirack.nyu.edu:22

Node Type: Other...

emulab-xen

Hardware Type: (any)

Fig. 3: GENI jacks interface

The advantage of using GENI was that experiments were not limited by the number of VMs or their connections. In addition, there was little to manage locally. On occasion, the system does experience slower performance but on the whole our experiences with the system were very good. There is a certain intangible satisfaction when the researcher is able to walk into the lab and directly interface with the testbed but this may be outweighed by the benefits.

Undergraduate students that are already familiar with our lab infrastructure have been given the SDN physical topology to build as part of the seminar. About half of the class reached all of the topology goals but these results are inconclusive because of the variation in student background. What was clear from the seminar was that the skills required came from both systems administration and networking. Graduate students are assigned the virtual build on GENI. This gives them the opportunity to work from several locations (many of our students are distance) and they do not need to maintain a lab presence for the entire semester. This is a regular part of our Enterprise

Networking graduate course. Each of the projects described in the next section will provide details on the equipment requirements and the challenges encountered. In the second project, a comparison between experiments run on GENI and the physical environments is also included.

4. COMPLETED RESEARCH PROJECTS

The following projects were run on campus during the last two years. They form the basis of the work going forward and the modules integrated into the systems and network administration coursework.

4.1. Traffic flow comparison between SDN controllers: Pox, Floodlight and OpenDaylight

A vast variety of SDN controllers have been deployed to date, mainly distinguished by the programming language employed, the protocols used to interact with the forwarding layer and the support of distributed cluster designs. This particular graduate capstone project examines the traffic behavior and network performance of multiple SDN controllers, using specific network setups while distinctive simulated TCP and UDP data streams traverse the network. This assessment intended to provide some insights about the operational features of different SDN controllers, but mainly a comparison analysis to distinguish traffic flow patterns using similar network environments among the controllers. Similarly to previous research, this project included the following:

— Deployment of the virtualized infrastructure, using an in house bare metal server and GENI
— SDN deployment: Controllers, virtual switches and end devices
— Definition of traffic generation mechanisms and tools for data gathering
— Definition of benchmarks for analysis

In overall, six independent experiments were performed to evaluate the performance statistics and traffic flow behavior between the controllers. These experiments were based, for the most part, on throughput (Mbps), delay (ms) and jitter (ms) measurements, taking into consideration the infrastructure employed.

These experiments were mainly instantiated in a hybrid infrastructure comprised by a physical box running VMware 5.5. As shown in Table I below, the system was not robust, but provided a stable and reliable environment while the research was underway.

Table I: SDN Virtualization Equipment

Description	Value
Operating System	VMware ESXi 5.5.0 build-2068190
System Model	Power Edge R520
Installed memory	16GB
Disk	Raid10 - 4 x 931GB = 1.81TB
Processor	4 x Intel(R) Xeon (R) CPU E5-2403 0 @ 1.80GHz

The network architecture consisted, mostly, in six virtual machines from where we used three as controllers, two as clients and one as openvswitch. Other nodes were used for routing and testing purposes, as described in the Figure 4.

The secondary system, in which we also performed several tests, was built in GENI. This topology mirrored the design elaborated in the lab, except for the routing and management devices. The virtual nodes were instantiated in multiple independent slices, which are isolated network instances in GENI cloud, as observed in Figure 5

Both the local and remote environments used a base installation of Ubuntu LTS 14.X, a Linux flavor with well known repositories and stable responses for OVS, SDN controllers, and OpenFlow in general. A clean installation of the virtual machine, prior SDN configuration,

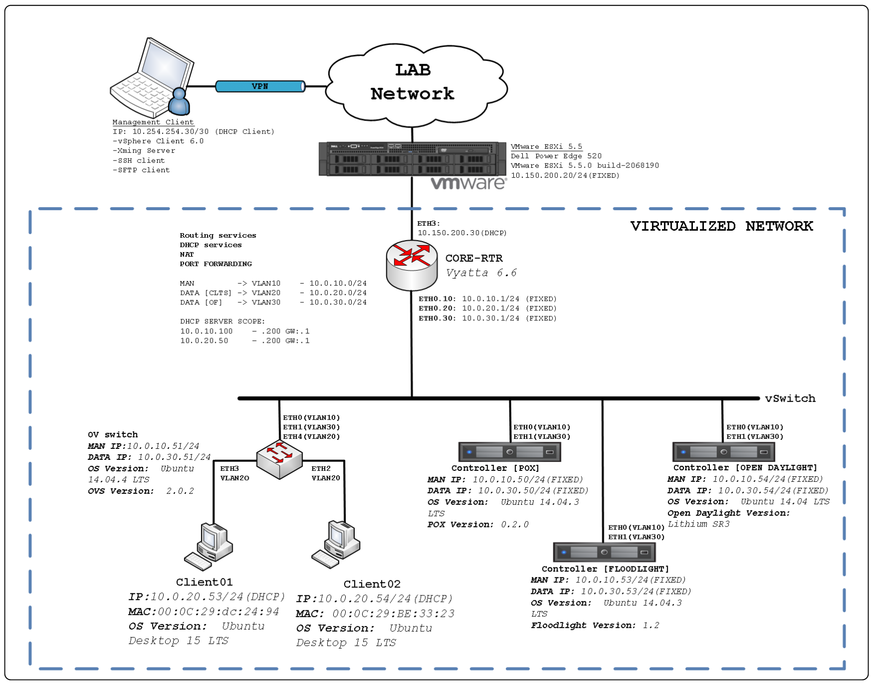


Fig. 4: Local network architecture

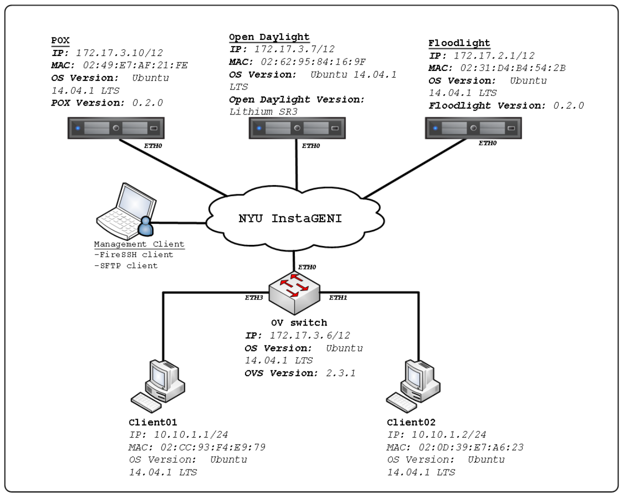


Fig. 5: Remote network architecture

contained the VWware tools, OpenSSH and updated repositories.

In summary, more than 300,000 samples were gathered among all tests to be able to provide an accurate comparison analysis between the controllers. Some traffic generation tools that made this research possible were IPERF and Ping, for TCP/UDP flows and ICMP packets.

After observing and analyzing the results, we were able to identify some interesting patterns in all controllers when dealing with queuing and processing flow times. We concluded that the injected delays by controllers are not relevant once the flow entries are installed in the forwarding device. Even though the flow construction set up times varies drastically from controller to controller, as well as their expiry times, the performance of the network measured in throughput and latency with respect to the clients, is consistently similar among all controllers.

Building this type of project allow students understand the concepts of network virtualization and SDN, as they are not given with a pre-fixed deployment, such as Mininet, as all the components are built from scratch. In our particular case, we ran into few difficulties even before dealing with our intended scope (SDN) -network isolation via virtual switches (in VMware hypervisor) for ensuring communication between the nodes and openvswitch was not possible in the first attempt, network segmentation for management traffic was not part of the initial design and providing a clean and functional installation of nodes and controllers was not an easy task either. Using a virtual environment resulted favorable, as the work could be backed up before initiating a new phase.

4.2. SDN testbed construction

The testbed described here was build to investigate the efficacy of both physical and virtual architectures for use in other projects and student work. Performance metrics were also examined. In order to experiment with SDN or SDN enabled technologies and ideas, the researcher must have either SDN enabled devices (such as those manufactured by Brocade, HP, Nicera or BigSwitch) or have access to SDN virtual machines or VMs. For example we might use Linux VMs and install controller/switch packages such as POX and OVS. When working with VMs, compute resources can be obtained from either a local or cloud facility. Cloud resources include Amazon Web Services or even private clouds. Locally we might use clusters or build a smaller setting such as one based on VMWare ESX.

The problem with general compute resources is that they may not be appropriate for the task at hand. For example, clusters may be well suited for providing significant processing capability but they are not very efficient when the need is for a large number of VMs. GENI (Global Environment for Network Innovation) is an architecture built for the purpose of experimentation with SDN type topologies. However, the resources are time restricted and storage is extremely limited. This project sought to explore the issues associated with construction of an SDN infrastructure. An examination of remote facilities will also be completed and the two approaches will be compared. The original project goals included:

— Develop a list of necessary hardware and software components.
— Deploy both ESX and KVM hypervisors and interconnect them with the wired network.
— Deploy SDN controllers and the associated openvswitch.
— Deploy network hosts
— Provide an explanation of system operation.
— Develop a testing methodology and collect a suite of tools for researchers and educators.

Time permitting, one of the supplemental tasks will be to emulate the Internet of Things or IoT. The IoT can have several meanings including machine to machine communication, very often with a collection of sensors. This emulation will be attempted using Mininet. Mininet is a virtualized network environment that can be extended to a physical network.

4.2.1. Testbed Construction and Testing - Phase 1. Since we did not possess switches or routers capable of communicating via OpenFlow, the devices had to be virtualized or installed as a virtual switch. Thus phase 1 was the construction of the virtualized infrastructure. The amounted to installation of VMWare ESX, the base virtual machines, software and some portion of the physical network in order to allow management nodes to communicate with the virtual machines. An early goal was to build a similar box based on Kernel Virtual Machine (KVM). However, there was not enough hardware and so this task was dropped.

ESX is a bare metal hypervisor that is managed remotely via VMWare vSphere. Ubuntu Linux was chosen as the base distribution for ease of use and the apt package manager. Four virtual machines were created; POX controller, OVS (Open vSwitch) and two network nodes. POX and OVS are packages that had to be installed and configured on top of Ubuntu. The virtual machines had to be connected from the virtual switch inside of ESX to the hardware switch of the testbed. The internal virtualized topology can be seen in Figure 6. This image shows not only the integration with virtualization but that network elements themselves are virtualized. Thus, there is the need to build VMs, configure them for operation and integrate them with a physical infrastructure.

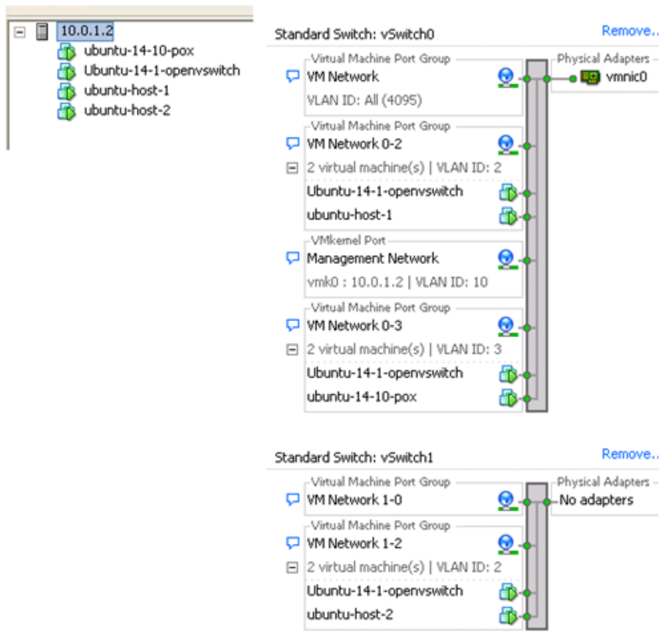


Fig. 6: ESXi Topology

Lastly, the VMs must be allowed to connect through the SDN topology within ESX to the outside world. This network configuration is non-trivial as it requires understanding of several advanced networking topics including VLANs, trunks, port forwarding and network address translation. At the conclusion of this phase, the first series of tasks for the project were completed. Once the topology was constructed, a collection of SDN experiments was selected for comparison against running them on GENI. These include layer 2 and layer 3 connectivity experiments.

4.2.2. Testbed Construction and Testing - Phase 2. The physical topology that was deployed for the project is shown in Figure 7. While the topology and tests were completed, there were still a couple of secondary activities that would increase the value of the project. One such task was to develop a methodology that might make the overall architecture less onerous to work with and therefore more practical for experimentation.

To this end, the router and switch configurations were modified for external access and the security of the system was increased. Security improvements include installation and configuration of

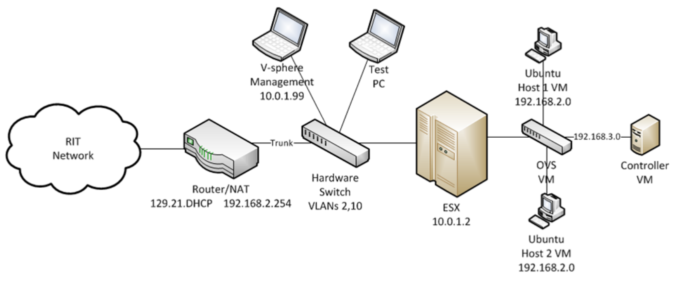

V-sphere Management 10.0.1.99
Test PC
Ubuntu Host 1 VM 192.168.2.0
RIT Network
Trunk
192.168.3.0
Controller VM
OVS VM
Router/NAT 129.21.DHCP 192.168.2.254
Hardware Switch VLANs 2,10
ESX 10.0.1.2
Ubuntu Host 2 VM 192.168.2.0

Fig. 7: Additional Physical Topology

Secure Shell servers (sshd) on virtual machines and hardening of the router.
Another modification was to the flow table tools. Flow tables contain all of the information necessary for the processing of packets but they can be difficult to read and interpret. Accessing them is straight-forward (ovs-ofctl dump-flows br0) but the information is not tailored for what the user requires. To alleviate some of this a small script was written that serves as a front end for the switch. The idea is that the user can either obtain the entire table or query the switch for specific information.

4.2.3. Testbed Construction and Testing - Phase 3. The last component of the project was to examine the testing methodologies that might be used on an SDN topology and add an aspect that might emulate the IoT. As mentioned previously, Mininet would be deployed for this purpose. Mininet is running in a separate VM with a switch and series of hosts. The challenge is in integrating Mininet into the topology because two switches will seek to connect to the same controller. Mininet commands can be run interactively but it is more efficient to use the Python API and so a script was written to handle the tasks of the Mininet topology. The entire integrated topology is shown in Figure 8.

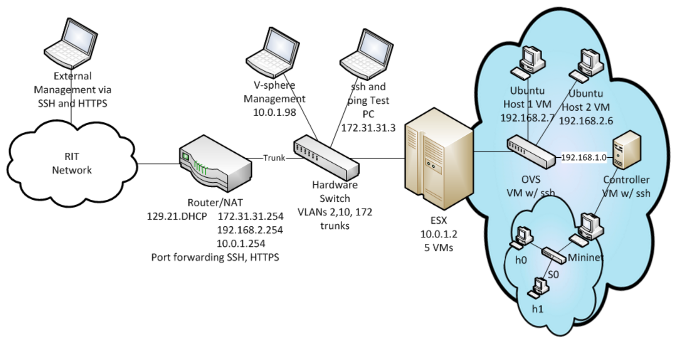

External Management via SSH and HTTPS
V-sphere Management 10.0.1.98
ssh and ping Test PC 172.31.31.3
Ubuntu Host 1 VM 192.168.2.7
Ubuntu Host 2 VM 192.168.2.6
RIT Network
Trunk
192.168.1.0
OVS VM w/ ssh
Controller VM w/ ssh
Router/NAT 129.21.DHCP 172.31.31.254 192.168.2.254 10.0.1.254 Port forwarding SSH, HTTPS
Hardware Switch VLANs 2,10, 172 trunks
ESX 10.0.1.2 5 VMs
h0
S0
Mininet
h1

Fig. 8: Complete Topology

In testing and acquiring data for this project, there are a couple of aspects of the architecture to be examined, most notably SDN on GENI vs. SDN on ESX and SDN vs. traditional networking. In the first case the results were inconclusive but this is simply because the architectures are so similar. Even though one is local and the other remote, the systems are still running in virtual machines and the virtual machines are resident on the same hardware.

However, there can be minor performance differences because of the resources allocated for each VM and the hardware configuration of the hypervisor chassis. The term minor is used because there are only 5 VMs running on a chassis with 8GB of RAM. While this does have an impact, a larger number of VMs would have been more telling. Attempting to manage and maintain remote resources can be a pain and there are resource limitations, but in this case no clear winner emerged.

When compared to traditional networking, SDN has the potential for greater management traffic because of the controller and the need for OpenFlow messages to be passed between network elements. Unless statically configured (which defeats the purpose), OpenFlow must be involved in every instantiation or modification of flows. This adds overhead, albeit only on the commencement of the flow. The impact becomes less as the flow life or packet train grows. One of the tools used to test the topology was iperf [**?**] which can be configured to send TCP or UDP packet flows. The average results from a base 30 second test are shown in the following tables.

The first set of values shown in Table II is from the virtualized SDN topology.

Table II: iperf SDN test Result Example

Window Size	Transfer	Bandwidth	Window Size	Transfer	Bandwidth
2.19KB	101.67MB	28.4Mbps	2.14KB	79.83MB	22.67Mbps
7.81KB	462.5MB	129Mbps	7.63KB	239.67MB	66.93Mbps
15.6KB	771.67MB	215.67Mbps	15.2KB	313MB	87.4Mbps

This second set of values shown in Figure III is from a comparable physical network.

Table III: Baseline Transfer Values (MB) vs. Windows Size

Window	Cisco	Juniper	D Link
4K	772.6 (16.85)	788.5 (3.41)	814.9 (33.64)
8K	1798 (6)	1820 (30.66)	1824 (23.75)
16K	1835(5)	1820 (10)	1831 (3)
64K	2793(9)	2725 (33.54)	2800 (16.73)

Unfortunately these results only lead to more questions. Due to limited resources, tests had to be run on available hardware and software. These results have far more variables than desired as they include Windows and Linux operating systems. The same is true of the iperf client software. Scaling the window sizes (a problem with different clients) does solve the apparent disparity in output. However, this may serve to underscore the changing impact of OpenFlow over flows of longer duration.

5. DISCUSSION

As systems and network administration programs seeking to provide the understanding and tools for success in either industry or graduate programs, there is the need to match suitable lab experiences with program goals. Early in the education process, these experiences can be found in software installation and management of small systems. As the student grows or a research agenda is established, finding a suitable match can be more challenging. Integrating the latest industry trends, keeping current and developing a comprehensive environment within which to work can add further challenges. In our own program we have wrestled with these issues and have discovered that the combination of software defined networks and the related trends in industry have provided a wide variety of opportunities for students and faculty. Graduate students have also discovered a new and rich vein of capstone topics to mine as SDN has grown to include areas such as SD-WAN, SD-storage and SD-data centers. We believe that having access to an SDN architecture for testing and research is critical for those working in the communication and networking fields. We have argued that SDN is an architecture that, along with related ideas of orchestration, dynamic programming and automation can add to any program. The creation of courses like Configuration Management and the adoption of tools such as Puppet, Chef and Kupernetes support this point of view. The virtualization course adopts the ”full-stack” approach which requires a comprehensive understanding of the systems rather than just the hypervisor and thus takes a more ”Devops-ish” direction.

But obtaining access to resources is another question. For many, it is not easy to obtain the physical resources required. While this might mean actual computers, it can also mean space or administrative personnel. Specifically, the construction of even a small SDN topology requires four nodes. While the controller and the network element (OVS) can reside in the same box, they still require resources. As this base topology, seen in Figure 2, is the one we used with our undergraduate students. Virtualizing helps because given enough memory, two teams might share the same chassis by running VMs and not interfere with each other. This topology also requires the students to perform a myriad of sys admin activities including accounts, software control, installation and authentication. But these projects have also shown that a local testbed requires significant resources and some level of expertise to configure and manage. So, one of the things that makes SDN and the related activities so attractive as a basis for many systems and network administration students may be lacking in either the students or the faculty member. But, if th steep learning curve is embraced with a local testbed, the students and faculty members gain considerable knowledge and experience as they are exposed to a broader domain of technologies and design requirements. As a starting point, GENI might be considered and the GENI conferences offer travel grants in order to bring educators and researchers up to speed. Again, the base topology is emulated on GENI in Figure 3.

As mentioned previously, many of the experiments resulted in either additional capstone projects for students or more in-depth research projects. The fist project outlined in section 4 is an excellent example of a successful graduate capstone. For example, the creation of the testbed led directly to intensive testing and a comparison between metrics on virtual and physical topologies [Hartpence and Kwasinski 2015]. Some of these tests indicate that for many experiments, a cloud environment like GENI is an effective and appropriate solution. For this reason, and because graduate students are less familiar with our lab infrastructure, we adopted on line or remote resources for graduate coursework. But in cases where increased storage or local control over resources are desired, a local chassis and testbed is still a superior solution.

6. CONCLUSION

Emerging technologies, such as SDN and network virtualization, are gaining increased attention in the industrial sector. The same is true for strategies such as orchestration and automation. Academic programs should be aligned to cope up with these new developing areas, to ensure the student body is well prepared for future endeavors. In this paper we have discussed what might be called traditional network and systems administration curricula and how these courses can be furthered or augmented by SDN topics and experiences. SDN provides an integrated environment which required many if not all of the background and skills obtained in a systems or network administration program, particularly if the goal is to work with a comprehensive SDN infrastructure. Our projects, the successful modification to coursework and industry examples such as the Facebook Auto Remediation System all lead us to believe that SDN and the related ideas a an important part of networking and systems administration programs.

Building, maintaining and operating an SDN environment for testing can be a considerable undertaking depending on the scope of the projects, the desired experimentation and the number of users. Some of the challenges associated with bringing an SDN architecture into fruition were outlined here including space, physical machines and heat. Since these challenges are not insignificant, this paper also describes the facilities used to mitigate the biggest issues. GENI is an on-line, well-supported infrastructure used primarily for the support of SDN projects and research.

Courses, advanced projects or simple tests can all make use of virtual machines and connections built within GENI.

Software Defined Networking provides an ample domain for research and investigation, as it is still a fairly new and diversified technology. Projects as the ones discussed in this paper, showcase various ways universities can introduce these new concepts to the scholars. By transitioning from the theory to practice, students gain important knowledge which could be then translated to improved ideas and novel designs for existing technologies.

REFERENCES

David Bernstein. 2014. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing* 1, 3 (2014), 81–84.

Chip Elliott. 2008. GENI-global environment for network innovations.. In *LCN*. 8.

Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. 2008. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review* 38, 3 (2008), 105–110.

Bruce Hartpence and Andres Kwasinski. 2015. Performance evaluation of networks with physical and virtual links. In *Global Information Infrastructure and Networking Symposium GIIS, 2015*. IEEE, 1–6.

Peter Hoose. 2011. Facebook Auto Remediation. (2011).

Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2015. Software-defined networking: A comprehensive survey. *Proc. IEEE* 103, 1 (2015), 14–76.

Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 19.

Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, and others. 2015. The design and implementation of open vswitch. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 117–130.

Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, and others. 2010. Carving research slices out of your production networks with OpenFlow. *ACM SIGCOMM Computer Communication Review* 40, 1 (2010), 129–130.

OpenFlow Switch Specification. 2011. Version 1.2 Wire Protocol 0x03. *Open Network Foundation* (2011).