Using Mininet for Emulation and Prototyping Software-Defined Networks

Rogério Leão Santos de Oliveira
Jales Technology College
State Center of Technology Education "Paula Souza"
Jales, Brazil
rogerio.leao@fatec.sp.gov.br

Ailton Akira Shinoda
Electrical Engineering Department
São Paulo State University "Julio de Mesquita Filho"
Ilha Solteira, Brazil
shinoda@dee.feis.unesp.br r

Christiane Marie Schweitzer
Mathematics Department
São Paulo State University "Julio de Mesquita Filho"
Ilha Solteira, Brazil
chris@mat.feis.unesp.br

Ligia Rodrigues Prete
Jales Technology College
State Center of Technology Education "Paula Souza"
Jales, Brazil
ligia.prete@fatec.sp.gov.br

***Abstract*— Software-Defined Networks (SDNs) represents an innovative approach in the area of computer networks, since they propose a new model to control forwarding and routing data packets that navigate the World Wide Web. Since research on this topic is still in progress, there are not many devices such as routers and switches that implement SDN functionalities; moreover, the existing ones are very expensive. Thus, in order to make researchers able to do experiments and to test novel features of this new paradigm in practice at a low financial cost, one solution is to use virtual network emulators. As a result, this paper focuses on study and evaluation of SDN emulation tool called Mininet. Initial tests suggested that the capacity of rapid and simplified prototyping, the ensuring applicability, the possibility of sharing results and tools at zero cost are positive factors that help scientists boost their researches despite the limitations of the tool in relation to the performance fidelity between the simulated and the real environment. After presenting some concepts of this paradigm, the purpose of its appearance, its elements and how it works, some net prototypes are created to better understand the Mininet tool and an evaluation is done to demonstrate its advantages and disadvantages.**

Keywords — Software Defined Networking, emulation, OpenFlow, virtualization, Floodlight, Mininet.

I. INTRODUCTION

It is evident the immense success of the computer networks today. Majority of the people's activities, directly or indirectly, use resources or services offered by communication networks.

The Internet is a computer network that interconnects thousands of computing devices around the world. A little time ago, these devices were basically desktop computers, workstations, and so-called servers that store and transmit information, such as web pages and e-mail messages. However, more and more devices such as TVs, laptops, game consoles, cell phones, webcams, cars and electronic security systems are being connected to the network. Indeed, the concept of computer networks is beginning to seem somewhat outdated, considering the large number of non-traditional equipment being connected to the Internet [1].

In the meantime this worldwide network of computers known as the Internet provides communication around the world, even though promoting sometimes dependence in the users, it faces a problem. The level of maturity in its structure also reduced its flexibility. Implementations of new technologies often require replacement of hardware devices (routers and switches), what can be very expensive and very hard-working for network administrators.

The research community of computer networks has been searching for solutions that enable the use of networks with more programming resources and less need for replacement of hardware elements, so that new technologies designed to solve new problems can be inserted gradually into the network and without significant costs.

The section 2 of this paper discusses the SDN paradigm reporting its motivation, the network elements that are part of this new structure as well as the operation of these components by comparing them to the traditional structure. The content of section 3 presents the simulation tool called Mininet and also describes some SDN controllers. The section 4 proposes an environment simulation using the presented SDN tools where tests are performed. These tests provide us results to discuss the topic of this work in section 5. Finally, section 6 describes other network simulators and section 7 presents the conclusion and suggestions for future works.

II. SOFTWARE-DEFINED NETWORKS

To mitigate flexibility problems, the complex administration of the current networks, the researchers of computer networks have invested in initiatives that implements networks with greater programming capabilities and reduce the

978-1-4799-4340-1/14/$31.00 ©2014 IEEE

need to replace switching equipments. Examples of such initiatives are the proposals for active networks [2], testbeds like PlanetLab [3] and, more recently, GENI [4]. Active networks, although they represent a potential item, had little acceptance because of their need to replace the network elements in order to allow them to become programmable.

It is exactly in this context which arose the new paradigm called Software-Defined Networking (SDN). It's a structure that aims to preserve the current performance on routing and forwarding data packets, since it maintains the actual structure with dedicated routers doing this job, but at the same time it delegates the method as it will be done to a new component, the controller.

A. The SDN elements

The SDN structure consists of three components: controllers, programmable switching elements and the standard protocol for communication between them.

The controllers or Network Operating Systems provides a programming interface where the developer or administrator can access the events generated by a network interface and can also generate commands to control the forwarding and routing of packets in the programmable switches.

This element centralizes all communication with the programmable switching elements that compose the network, so providing a unified view of the network status. This also simplifies the activity of network administrators, which did not need 'to know in depth' the details of the programming of the switching elements.

This new network control structure allows us to have a more effective and independent management. The controllers can implement more sophisticated traffic monitoring rules, for example, a solution that provides new abstractions for users, giving for each one the view that their machines are connected to a single and private switch, independent on others.

The switches and network routers, previously independent and autonomous, are now configured by the network controllers, that can implement rules for switching and policies for security, based on higher abstraction levels than the current model, because were previously defined by network administrators in the controller.

From a historical standpoint, SDNs have their origin in the definition of network structure Ethane [6], which defined to implement access control policies in a distributed way from a centralized monitoring mechanism [7]. In this structure, each network element should consult the supervisor element to identify a new flow. The supervisor consult a global policies group to decide, based on the characteristics of each flow, as the routing member should treat it. This decision would be communicated to the switch with a rule in its routing table, until sometimes a discarding rule. This model was later formalized by some of the authors in to OpenFlow protocol [5]. This is the third and no less important element of the SDN structure.

III. THE MININET

The paradigm SDN is still recent therefore the network researches have focused their on studies of the topic. When those researchers want to test the new SDN features in the controllers, switches or even in the OpenFlow protocol, they have some difficulties. Those difficulties happen specially because there are so few cheap devices available that are able to implement in SDN standard. Moreover, in more specific cases, when it is necessary to simulate large networks with large numbers of hosts, switches and SDN controllers, using the Internet may not be a good idea, because improper configurations can cause unwanted problems.

One of the solutions for this problem is making prototypes and simulating them in virtual mode. To do this, some tools have been created and one of them is the Mininet software [8].

The Mininet is a system that allows rapidly prototyping large networks on a single computer. It creates scalable Software-defined networks using lightweight virtualization mechanisms, such as processes and network namespaces. These features permit the Mininet create, interact with, customize and share the prototypes quickly.

Some characteristics guided the creation of Mininet are 1) flexibility, that is, new topologies and new features can be set in software, using programming languages and common operating systems; 2) applicability, correctly implementations done in prototypes should be also usable in real networks based on hardware without any changes in source codes; 3) interactivity, management and running the simulated network must occur in real time as if it happens in real networks; 4) scalability, the prototyping environment must be scaling to large networks with hundreds or thousands of switches on only a computer; 5) realistic, the prototype behavior should represent real time behavior with a high degree of confidence, so applications and protocols stacks should be usable without any code modification; and finally 6) share-able, the created prototypes should be easily shared with other collaborators, which can then run and modify the experiments.

A. SDN elements on Mininet

The Mininet can create SDN elements, customize them, share them with other networks and perform interactions. These elements include Hosts, Switches, Controllers and Links. A host on Mininet is a simple process with its own network environment running on the operating system. Each one provides processes with exclusive ownership virtual network interface, ports, addresses and routing tables (such as ARP and IP). The OpenFlow switches created by Mininet provide the same packet delivery semantic that would be provided by a hardware switch. Both user-space and kernel-space switches are available. In Mininet simulation, the controllers can be run on the real or simulated network as long as the machine on which the switches are running has connectivity to the controller. If desired, the Mininet creates a standard controller inside the local simulation environment, and virtual connections can also be created among the elements through their virtual interfaces.

For example, the command line

mn –topo single,3 –mac –switch ovsk –controller remote

creates three virtual hosts, creates a single Software OpenFlow switch in kernel with 3 ports, connects the virtual switch using virtual links, sets the MAC and IP addresses for each host and then finally configures the switch to connect to a remote controller. In this case, the controller is running locally on the same hardware that the simulator Mininet is running.

B. Interacting with a network and using control tools.

After creating the elements and all the connections between them, it is crucial to be able to run commands on hosts to test network functionality, and verify switch operation.

For this purpose, the Mininet includes a command line interface to allow developers to control and manage an entire network. Typed commands are interpreted by the emulator and executed in the simulated network environment.

For example:

mininet > nodes
Displays available node list;

mininet > help
Displays available network command list;

mininet > h2 ifconfig
Displays host IP address, h2 in this *case;*

mininet > h2 ping h1
Sends a packet (ICMP REQUEST) from host2 to host1.

Several examples and tools, such as text-based scripts and graphic applications, are in the virtual machine that contains the Mininet. Two very useful tools to monitor and measure the functionality of the Mininet networks are dpctl and wireshark.

The dpctl is a tool that can display and control the flow table of switches. It is especially useful for debugging, because it allows viewing the flow state and flowmeters. Most switches that implement OpenFlow listening to commands on port 6634 (by default), what makes it possible to interact with the switch without needing to add debugging code in the controller.

For example, the command line

dpctl dump-flows tcp:127.0.0.1:6634

connects to the switch and displays the flow table installed.

It is also possible to enter rules in the switches' flow tables manually. The command line, for example,

dpctl add-flow tcp:127.0.0.1:6634 in_port=1, actions=output:2

creates a rule in which all packets arriving at port 1 of the switch will be forwarded to port 2.

The wireshark is a tool that captures and dissects all data packets transmitted by network interfaces. Exclusively for Mininet, wireshark has a library called OpenFlow (of) that filters all OpenFlow packets, so it allows OpenFlow packets captured by a local interface of the running hardware to be isolated from other application packets. To use the tool, it is enough just to run it along a graphic server.

C. The SDN controllers and the Mininet.

The SDN principle is the capability to control the packet forwarding process through a unique interface. The controller can center all communications with the programmable network elements and provide a single view of its state by isolating the details of each element.

One of the SDN advantages is exactly this centralized view of the network that makes possible to develop a detailed analysis and to decide how the system should operate.

The Mininet emulator implements connection between switches and different controllers, such as NOX and Floodlight for instance. This possibility allows developers interested in creating and testing controller resources to be able to use Mininet to perform their simulations.

IV. Simulating a SDN on Mininet and Floodlight Controller.

Floodlight [9] is an OpenFlow controller for enterprise networks based on Java programming language and distributed under the Apache license. The first project originated from the Beacon controller [12] and now it is supported by a developers' community and also by Big Switch Networks, a start-up that produces commercial hardware controllers. The core and main modules are written in Java. Recently, the Jython programming language was included in the project allowing development in Python. In its infrastructure, all the elements are modules and these modules provide services. All the communication among modules is done through services. The ItopologyService interface allows discovering the network topology automatically; moreover, it can integrating non OpenFlow networks and is compatible with the simulation tool Mininet.

This section presents samples of Mininet utilization with the Floodlight controller software. The objectives are testing the emulation tool in different types of networks and obtain viability results.

A. Emulation environment specifications.

For the experiments we used a microcomputer HP Compaq 8200 Elite SFF PC with the following specifications: Processor Intel ® Core ™ i5-2400 3.10GHz, 4GB of RAM running the Operating System Windows 7 64bits and VirtualBox Oracle VM version 4.2.12.

In this microcomputer, under the management of VirtualBox, we installed the following guest operating systems: Mininet Emulator version 2.0 on Linux operating system Ubuntu 10.12 64bits with 1Gb of RAM; Floodlight Controller version 0.90 on Linux operating system Ubuntu 12.10 64bits with 256MB of RAM.

B. Running tests.

In this first simulation scenario we create two hosts, two switches, one controller and all nodes were connected with wired links in according to topology in Figure 1.

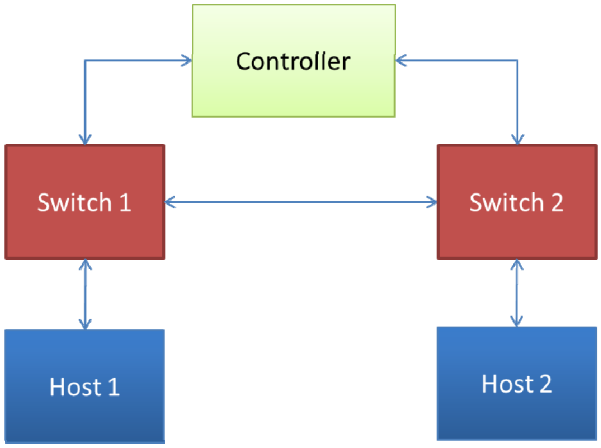

Fig. 1. Simulation topology.

The follow command line was used to create the proposed scenario.

mn –topo=linear,2

After creating the scenario, we execute a test of connection using the packet ICMP (echo-request "ping") from hosts 1 to host 2 as the following command line:

h1 ping –c1 h2

The ICMP test execution succeeded, hence, no packet loss occurs. It is important to emphasize that no additional configuration was set on the switches or controller, so the communication was only successful because the ICMP packet sent from host 1 to host 2 when arrived at switch 1 which has no forwarding rule on your flow table, so it sent the packet to the controller which forwarded it to switch 2.

It is very negative for the packet forwarding performance in the network if none of both switches has its respective flow tables, because all packets will pass through the controller before reaching its end destination. This way, there is significant dependency and likely network controller congestion.

To demonstrate this negative dependency on the controller, some other tests were also run in this scenario.

The controller was disconnected from the switch and the same communication test with the ICMP packet between host 1 and host 2 was repeated. As a result, the communication cannot be established, generating 100% packets loss. This happened because the switch 1, which has no forwarding rule upon receiving the ICMP packet, tried to send it to the controller, but this one was absent and caused the error.

After that, with the controller disconnected, the forwarding rules were directly set in flow tables of the both switches and the communication test with ICMP packet was repeated again. The result was successful. The ICMP packet could reach its destination, because of the inserted rules on the switches; these ones did not need the controller for forwarding the packets. Figure 2 displays the rules set in switches' flow tables.

It is important to remember that these rules were manually added to the flow tables of switches as the following line commands, but in the SDN paradigm, this task is performed by the controller that can add, modify and delete any rules.

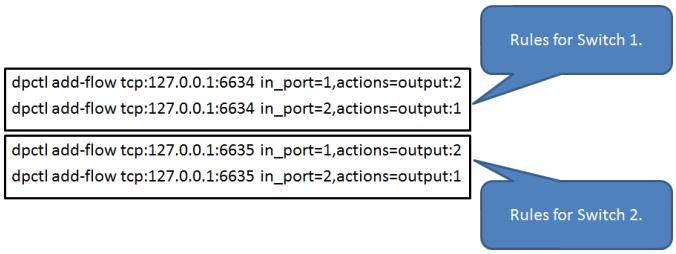

Fig. 2. Inserted rules on switches' flow tables.

C. *Scalability tests.*

Even in this environment, some scalability tests were done. They will be explained in the following section. We created a bash script that initiates different virtual networks topologies in Mininet, prints the amount of memory used up to this point, destroys the virtual network and prints the time spent on the entire procedure. The script mentioned is transcribed below.

```
#!/bin/bash
for n in `seq 1 6`; do
echo "n: $n"
echo -e "h1 free -m | grep ^Mem | awk '{print \"mem: \" \$3}'\nexit" | \
mn --topo tree,$n 2>&1 | \
grep -E "^(mem:|completado em)"
done
```

The script execution results are displayed in Figure 3.

Topology	Node numbers	Host numbers	Switch numbers	start/stop time (in seconds)	Memory usage (in MB)
Tree	3	2	1	0,190	441
Tree	7	4	3	0,525	442
Tree	15	8	7	1,175	445
Tree	31	16	15	2,795	452
Tree	63	32	31	7,088	466
Tree	127	64	63	20,210	449
Tree	255	128	127	64,818	572
Tree	511	256	255	242,842	739

Fig. 3. Scalability test results.

It is noted that for the creation of small virtual networks, the spent time is also very short and this time increases with the increase of virtual nodes number. For example, the Mininet took approximately 64 seconds to create a tree topology with 255 nodes, while with 511 nodes, the time spent substantially increased for about 242 seconds. The graphic in Figure 4 illustrates it.

In relation to the amount of memory used for creating the virtual networks, as illustrated in Figure 5, while the number of nodes was little, the Mininet showed a lower memory consumption. This memory consumption was practically the same until virtual network creation with 127 nodes where increased linearly after that.

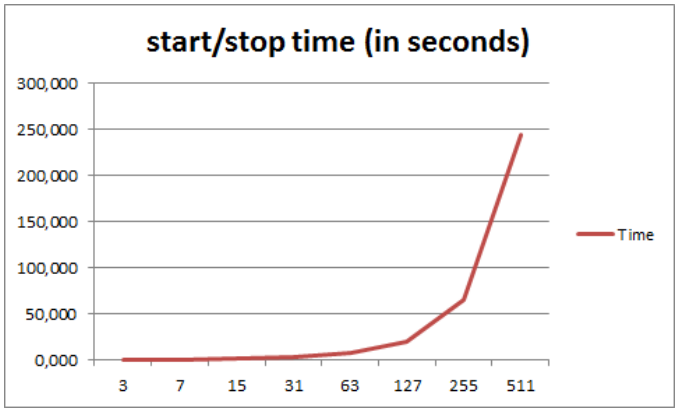

Fig. 4. Graphic displaying the time spent for creating and destroying the virtual networks.

Another paper [10] also made some scalability tests like those ones and excepting only the differences found in the processing power of hardware used in both tests, the results were similar. Mininet begins to take longer to create and destroy virtual networks and use much memory space when the number of virtual networks becomes larger.

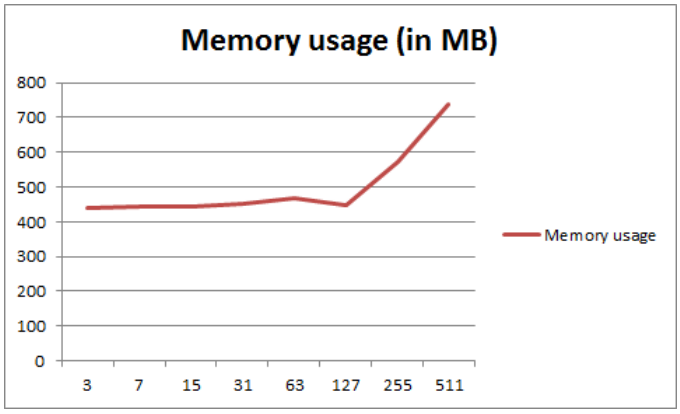

Fig. 5. Gráfico representando a memória ocupada para as redes.

V. RESULTS AND DISCUSSIONS.

This new paradigm called Software-Defined Networks (SDN) has received great attention from the scientific community and from the network management system manufacturers. Being a current topic with innovative trends and providing various application possibilities, this paradigm is being called "The Future of Internet" [11]. Therefore, many manufacturers and researchers joined the idea of a network structure defined by independent applications of hardware and no longer just based on the principle of traditional IP Routing.

The emulator software Mininet studied in this paper brings a unique contribution to the advancement of research on SDN paradigm and after performing several tests and reviewing researcher studies, here are some considerations.

A. Limitations

Currenty, the most significant limitation of Mininet is its lack of performance fidelity, especially at high loads. CPU resources are multiplexed in time by the default scheduler, which provides no guarantee that a host that is ready to send a packet will be scheduled promptly, or that all the switches will forward at the same rate [10].

The Mininet currently runs on a single machine and emulates all hosts, switches and links in a single operating system. All these elements share the same hardware resources, what is a disadvantage for experiments on a larger scale.

B. Prototyping

Although not suitable for large-scale simulations which require significant amounts of nodes, the Mininet is a great software for prototyping small and medium networks.

Students, researchers, network administrators and other stakeholders with a simple laptop can use Mininet for rapid prototyping of a SDN idea. The short startup time and low overhead enable exploring a design space and making a medium scale system that runs on a modest hardware. Several researchers can share scripts, configurations, topologies and work on prototypes simultaneously without interference.

C. Applicability

One of the most important requirements for any emulator software is the applicability. In this context, all implemented resources and ideas in a simulation environment must function properly in real environments without significant changes.

The Mininet proves this capability since it allows those tests performed in a simulated environment to be able to be easily shared to other test infrastructure, independent on hardware or even in a production environment.

D. Sharing

Thus the ability to share is perhaps one of the strongest points of the emulator Mininet. A project, a topology or a test code can very easily be distributed to the entire research community. The emulator itself, when downloaded from the official website [8], includes in the virtual machine package initially, the Mininet, a variaty of different tools to perform and analyze SDN prototypes, and a set of sample codes that create different SDN topologies and facilitate the development of research.

Moreover, it is also possible that researchers who create additional tools, sample codes and different SDN settings can share them with other researchers. To do so, it is necessary just generate a new image for the virtual machine and distribute them over the internet.

VI. ANOTHER NETWORK SIMULATORS

IMUNES [13] added virtual Ethernet interfaces and a feature similar to network namespaces into the BSD Kernel. It enables rapid prototyping, but in itself it does not provide a path to hardware deployment or a means for distribution and sharing.

EMULAB [14] took another OS-level virtualization tecnhonology, FreeBSD jails, and modified it to allow multiple virtual interfaces per process group, similar to network namespaces. EMULAB virtual nodes present a different design

point, emulating 10 or more nodes on a single PC with close fidelity. Despite implementing lightweight virtualization techniques, these tools do not support the OpenFlow protocol.

Nowadays, very few network simulators support the OpenFlow protocol; one is ns-3 [15]. The ns-3 tool simulates the operations of an Open-Flow switch by compiling and linking an Open-Flow switch C++ module with its simulation engine code. To simulate a real OpenFlow controller, ns-3 also implements it as a C++ module, and compiles and links it with its simulation engine code. There is a project of ns-3 for supporting the OpenFlow protocol, but only the version 0.89 of the OpenFlow protocol is supported, which is quite old as the latest version is already 1.3.2.

Another OpenFlow network simulator and emulation is EstiNet [16]. EstiNet uses a unique approach to testing the functions and performances of OpenFlow controllers. By using an innovative simulation methodology, which is called kernel re-entering [17], EstiNet combines the advantages of both the simulation and emulation approaches. In a network simulated by EstiNet, each simulated host can run the real Linux operating system, and any UNIX-based real application program can readily run on a simulated host without any modification. Although EstiNet also supports the OpenFlow protocol and can be used to simulate SDNs, easy sharing results and tools with researchers is a Mininet advantage.

VII. CONCLUSIONS AND FUTURE WORKS

This paper aimed to evaluate the SDN emulation tool called Mininet.

In the first sections of this paper the SDN paradigm, its purpose, its elements and its operational structure were introduced. The Mininet, the Floodlight controller and other SDN tools were presented and used for prototyping and simulation. Other researches results were also referenced in this study.

After performing several tests and simulations, we concluded in this paper that despite the limitations of the tool in relation to the fidelity of performance between the simulated and the real environment, the Mininet tool has great importance for SDN research. In practice, waiting a few seconds for full larger topologies to start is quite reasonable and faster then the boot time for hardware switches. Moreover, the capacity of rapid and simplified prototyping, the applicability safety, the possibility of sharing results and tools at zero cost are positive factors that help scientists to boost their researches.

Mininet has been used by over 100 researchers in more than 18 instituions, including Princeton, Berkeley, Purdue, ICSI, UMass, University of Alabama Huntsville, NEC, NASA, Deutsche Telekom Labs, Standford, and a startup company, as well as seven universities in Brazil [10]. This fact is a good indicator of advantages of this tool.

Future works could research about Mininet with other external controllers and also compare the performance results among it and other network simulation tools, such as PlanetLab [3], GENI [4] and EstiNet [16].

Using the Mininet tool addressed in this paper, it is also possible to conduct further researches into Software-Defined Networks. Questions such as what the most appropriate network abstraction is, how to implement clearance mechanisms and how to configure the controllers to ensure best reliability, performance and security, still represent a wide variety of topics that can be explored.

REFERENCES

[1] K. F. James and R.W. Keith, “Computer Networks: A Top-Down Approach”, 5th ed. Person Addison Wesley, 2010, pp. 1-50.

[2] T. L. David and W. J. David, “Towards an active network architecture.” in SIGCOMM Comput. Commun. Rev., vol. 37, no. 5, pp. 81-94, Oct. 2007.

[3] P. Larry and R. Timothy, “The design principles of planetlab.” in SIGOPS Oper. Syst. Rev., vol. 40, no. 1, pp. 11–16, Jan. 2006.

[4] GENI. (2010, Mar). Global Environment for Network Innovations: Spiral 2 overview. GENI. [Online].Available: http://www.geni.net/

[5] M. Nick *et al*., “OpenFlow: enabling innovation in campus networks”, ACM SIGCOMM Computer Communication, vol. 38, no. 2, pp. 69-74, Apr. 2008.

[6] C. Martin *et al.*, “Ethane: Taking control of the enterprise.” in Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, vol. 37, no. 4, pp. 1–12, Oct. 2007.

[7] C. Martin *et al.*, “Rethinking enterprise network control.” in IEEE/ACM Transactions on Networking, vol. 17, no. 4, pp. 1270-1283, Aug. 2009.

[8] Mininet. (2013, Mar). An Instant Virtual Network on your Laptop (or other PC). [Online].Available: http://mininet.org/

[9] Floodlight. (2013, Mar). Open Source Software for Building Software-Defined Networks. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[10] L. Bob *et al.*, “A network in a laptop: rapid prototyping for software-defined networks.” In Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets, vol. 10, no. 19, pp. 1-6, Oct. 2010.

[11] D. D. M. Marcelo *et al.*, “Internet do futuro: Um Novo Horizonte,” in *Simpósio Brasileiro de Redes de Computadores*, 2009, pp. 1-59.

[12] OpenFlow (2013, Jul). OpenFlow 1.3.2 Specification. [Online]. Available:https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.2.pdf

[13] M. Zec and M. Mikuc. Operating system support for integrated network emulation in imunes. In Proc. of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS), 2004.

[14] M. Hibler *et al.*, Large-scalevirtualization in the emulab network testbed. In USENIX 2008 Annual Technical Conference. USENIX, 2008, pp. 113-128.

[15] T. R. Henderson *et al.*, “Network Simulations with the ns-3 Simulator,” ACM SIGCOMM ’08, Seattle, WA, Aug. 17–22, 2008.

[16] EstiNet 8.0 OpenFlow Network Simulator and Emulator, EstiNet Technologies Inc., http://www.estinet.com.

[17] S. Y. Wang and H. T. Kung, “A Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulators,” IEEE INFOCOM ’99, New York, Mar.21–25, 1999.