Emulation of Software Defined Networks Using Mininet in Different Simulation Environments

Faris Keti
Electrical and Computer Engineering Department
Faculty of Engineering
Duhok-Kurdistan Region,Iraq
faris.alyas@uod.ac

Shavan Askar
Electrical and Computer Engineering Department
Faculty of Engineering
Duhok - Kurdistan Region, Iraq
Shavan.askar@uod.ac

Abstract- **In this paper an evaluation of an SDN emulation tool called Mininet is conducted. Tests were conducted to study Mininnet limitations related to the simulation environment, resource capabilities. To evaluate the later, the scalability of Mininet in term of creating many topologies is tested with varying number of nodes and two different environment scenarios. Results show that the simulation environment has a remarkable effect on the required time for building a topology, for instance, the powerful resources scenario needed only 0.19 sec, whereas, 5.611 sec were needed when the resources were less. However, the required time were increased in both scenarios when the number of nodes was increased into 242.842 and 3718.117 sec for the powerful and less capabilities resources respectively.**

Keywords-Software Defined Networking, Emulation, OpenFlow, Pox controller, Minint.

I. Introduction

Because of the astronomic number of users each with different applications or services which also increase continuously, the Internet has already become a kind of global communication infrastructure.

But, the Internet with its current capabilities is still suffering from variety of great challenges in network control, configuration, and management issues. Research on Internet application, service, network control and management is one of thehot and interesting topics in recent years. Many of research projects have been initiated in this subject's area [1].

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 [2, 3]. Software-Defined Networking (SDN) is a new network paradigm that virtualizes network infrastructure by decoupling the control and data plane logic of traditional network devices, creating a dynamic, flexible, automated and manageable architecture.

SDN is implemented through a protocol known as OpenFlow that lets administrators select the path through which data will flow through a network [4].

Section 2 of this paper discusses the SDN paradigm describing its motivation, network elements that are part of this new structure, in addition, the operation of these components. Section 3 describes the utilized simulation tool which is called Mininet, in addition, it presents MININET characteristics. In section 4, the role of controller in SDN networks was proposed utilizing SDN tools and also one of MININET performance related to the simulation's environment was tested.Section 5conclude the paper and give suggestions for future work.

II. SOFTWARE-DEFINED NETWORKS

Experienced IT personnel are inevitable when it comes to configuring and installing network's elements. Accurate and complicated simulators are needed when the simulating networks in which interactions among its elements are needed such as switches, routers, etc. Supporting this is highly difficult to achieve. Therefore; a new network model is required to support these agility requirements.

There are four factors that SDN focuses on which are namely;

A. *Control plane and data plane Separation*
B. *A centralized controller and view of the network*
C. *Open interfaces between the devices in the control plane and the data plane*
D. *Use of external application to support the programmability feature of the network*

A. Control plane and data plane separation:

One of the tenets expressed early in the introduction of SDN is the potential advantage of the separation of a network device's control and data planes. This separation affords a network operator certain advantages in terms of centralized or semi-centralized programmatic control. It also has a potential economic advantage based on the ability to consolidate in one or a few places what is often a considerably complex piece of software to configure and control onto less expensive, so-called commodity hardware [5,6,7,8].

B. A centralized controller and view of the network

The most important characteristic of SDN is the use of a centralized control plane, which transfer the decision-making logic from the network devices into external controllers [4]. Because of this, an awareness of network elements and characteristics will be attained.

C. Open interfaces between the devices in the control plane and the data plane (OpenFlow):

In the OpenFlow architecture, an OpenFlow switch consists of many flow tables in addition to an abstraction layer which communicates with a controller in a secure manner via OpenFlow protocol. Flow tables contain forwarding flow entries, each one match packets to its correspondent flow, then processed and forwarded. Forwarding flow entries consists of the following parameters; match fields which are used for the purpose of matching the incoming packets; match fields use information of packet header, ingress port, and metadata; counters, which are used to build up statistic data for each flow such as the number of received packets, bytes and duration of a particular flow; and a set of instructions, which applies when there is a match; they dictate how to handle matched packets[9].

D. Use of external application to support the programmability feature of the network:

In SDN environment, it is possible to control and program network devices, there is a capability of utilizing software applications that are developed from hardware for the purpose of controlling the policies of packet forwarding [7]. With SDN, it is possible to deploy network applications that haveprecise traffic monitoring and processing capabilities. This will consequently support; a dynamic QoS provisioning, load balancing, and access control.

SDN has many advantages [10]; it supports a faster response to changing traffic conditions. In addition, it allows for more options for dynamic provisioning, better load balancing, more preferred traffic engineering, improved network resource utilization, and enhanced opportunities to implement many new types of services.

III. MININET

There are many reasons that trigger the use of a new emulator called Mininet; first, there are only few network devices available for the purpose of implementing SDN standard as it is yet not widespread technology from the industrial perspective. In addition, implementing network with large number of network devices is very difficult and costly. Therefore, to overcome these problems, virtual mode strategy has been conducted for the purpose of prototyping and emulating there kind of network technologies and the most important one is MININET [1].
Mininet has the capability to emulate different kinds of network elements such as; host, layer-2 switches, layer-3 routers, and links. It works on a single Linux kernel and it utilizes virtualization for the purpose of emulating a complete network utilizing only a single system. However, the created host, switched, routers, and links are real-world elements although they are created by means of software [11].

A. Some characteristics guided the creation of Mininet are:

- Flexibility, that is, new topologies and new features can be set in software, using programming languages and common operating systems.
- Applicability, correctly implementations conducted in prototypes should also be usable in real networks based on hardware without any changes in source codes.
- Interactivity, management and running the simulated network must occur in real time as if it happens in real networks.
- Scalability, the prototyping environment must be scaling to large networks with the hundreds or thousands of switches on only a computer.
- 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.
- Share-able, the created prototypes should be easily shared with other collaborators, which can then run and modify the experiments [12].

B. MININET Workflow

Mininet has the capabilities that enable researchers or network programmers to create software defined network prototype in a simple manner. With the capability to interact, customize and share it, and provides a smooth path to running it on hardware.

. Creating a Network

A network can be created in MININET with a single command;
```
$ sudo mn --topolinear,3
```

Create a virtual network of three switches connected linearly and three virtual hosts, eachhost configured to the corresponding switch.

. Interacting with a Network

In Mininet's the entire virtual network can be controlled, and managed from a single console. For example, the CLI command
Mininet >h1 ping –c3h2
Is used to pingh2 from h1
Mininet > nodes
To see the list of nodes available
Mininet > help
To see a list of available commands
Dpctl: control and edit flow tables.
Iperf: TCP speed test.

. Customizing a Network

Custom networks with a few lines of Python can be created by Mininet's API. For example,
~/mininet/custom/topo-2sw-2host.py
This few lines of python create a virtual network of two hosts connected via virtual links to two switches.

. Sharing a Network

Mininetsupports the capability of sharing the created VM image to other researchers for the purpose of running, evaluating, or modifying it.

. Running on Hardware

The created prototype could be implemented into hardware for testing and validation [13].

IV. SDN SIMULATION USING MININET AND POX CONTROLLER

POX is a software platform developed in Python [14][15].POX began early as a controller for an OpenFlow protocol. However, it can, nowadays, act as an OpenFlow switch, and can be used for developing networking software.

POX work with Python 2.7 (it can also work fine with Python 2.6), and can run under Linux OS, Mac OS, and Windows. The core and main modules are developed in python.

In this section many tests are presented which explain some features and capabilities of MININET emulator along with POX controller software in order to test the MININET tool in different networks topologies and environment.

A. MININET environment specifications.

In all the tests or experiments a computer DELL Inc. Inspiron1525 PC with the following specifications was used; processor Intel ® Core ™ 2 Duo CPU 2.00 GHZ,3072 MB of RAM running the operating system Windows 7 32bits and VirtualBox Oracle VM version 4.3.18.

The guest operating system: MININET Emulator version 2.1 on Linux operating system Ubuntu14.0432bits with 1 GB of RAM and POX 0.2.0(carp) Controller are installed on this computer, under the management of VirtualBox.

B. First Experiment.

In this test, the role of the controller in setting forwarding rules in the flow tables is explained. It also shows how it can control the behavior of forwarding hardware in the way of dealing with packets and also what happens in the absence of controller and what should we do alternatively. So, from MININET console:
$ cd pox

Now the controller will be programmed as a hub:
$./pox.py log.level –DEBUG misc.of_tutorial

This tells POX to enable verbose logging and to start the of_tutorial component (which runs as a hub).
In a new SSH terminal, a topology with four hosts, one switch, one remote controller was created. The connections between nodes were wired links as shown in Figure 1 below.

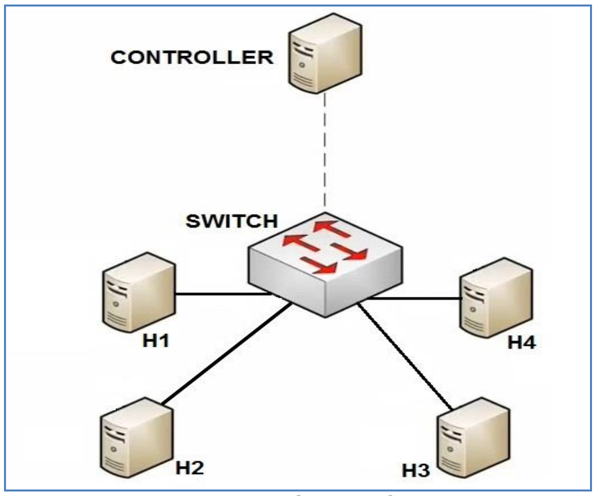


Figure 1. Simulation topology

The proposed topology created with the following command line:
$ sudo mn –topo single,4 – controller remote

Now to verify the behavior of a hub, that all hosts see the exact same traffic when hosts ping each other, xterms for each host was created, the traffic in each host was viewable. In the Mininet console:
Mininet> xterm h1 h2 h3 h4

In the xterms for h2, h3, and h4, we run tcpdump , a utility that print the packets seen by a host:

```
# tcpdump –xx –n –I h2-eth0
# tcpdump –xx –n –I h3-eth0
```

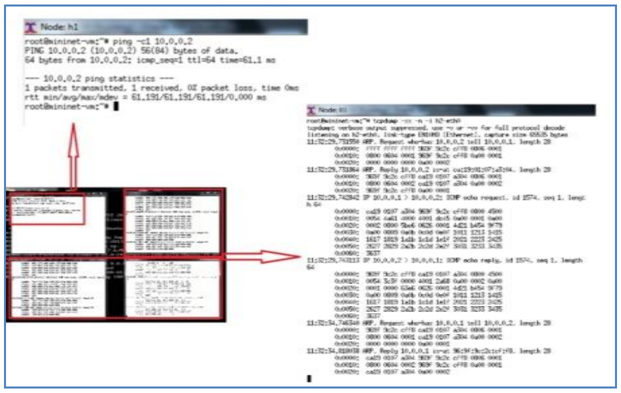
And respectively

```
# tcpdump –xx –n –I h4-eth0
```

In the xterm for h1, we sent a ping:

```
# ping -c1 10.0.0.2
```

The ping packets are now going up to the controller, which floods them out into all interfaces except the sending one. There should be identical ARP and ICMP packets corresponding to the ping in all the xterms running tcpdump. This is the behavior of a hub; it sends all packets to every port on the network, Figure 2.



```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=61.1 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 61.191/61.191/61.191/0.000 ms
root@mininet-vm:~# 
```

```
root@mininet-vm:~# tcpdump -xx -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:32:29.731550 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        [illegible]
11:32:29.731864 ARP, Reply 10.0.0.2 is-at [illegible], length 28
        [illegible]
11:32:29.742842 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1574, seq 1, lengt
h 64
        [illegible]
11:32:29.743113 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1574, seq 1, length
64
        [illegible]
```

Figure 2. The controller hub behavior.

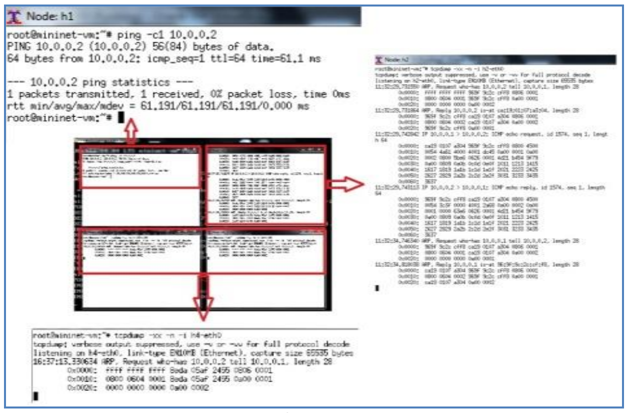
Now the previous scenario will be repeated but this time with programming the controller as a learning switch by using the following component:

```
$ cd pox
$ python ./pox.py forwarding.l2_learning
```

Again in the xterm for host1a ping request is executed

```
# ping –c1 10.0.0.2
```

Then as shown in Figure 3 only the destination host will receive the ARP and ICMP packets corresponding to the ping. Hosts that are not the destination for a ping should display no tcpdump traffic after the initial broadcast ARP request.



```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=61.1 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 61.191/61.191/61.191/0.000 ms
root@mininet-vm:~# 
```

```
root@mininet-vm:~# tcpdump -xx -n -i h4-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h4-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:37:13.330634 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 8eda 05af 2455 0806 0001
        0x0010:  0800 0604 0001 8eda 05af 2455 0a00 0001
        0x0020:  0000 0000 0000 0a00 0002
```

Figure 3. The behavior of controller as a learning switch.

To further have an insight into the role of controller in inserting fields into the forwarding rules, the controller was disconnected and another ping command was executed. Because switches are only forwarding hardware and there are no flow entries in its flow tables, all the ping packets were lost because of the absence of the controller, therefore; forwarding rules were entered manually as follow:

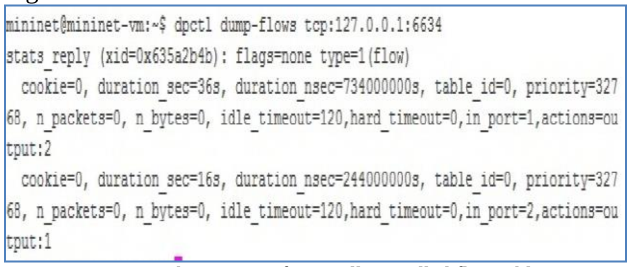
```
$ dpctl add-flow
tcp:127.0.0.1:6634in_port=1,idle_timeout=120,actions=ou
tput:2
$ dpctl add-flow
tcp:127.0.0.1:6634in_port=2,idle_timeout=120,actions=ou
tput:1
```

This will forward packets coming at port 1 to port 2 and vice-verca.

To display the flow table installed, this command line is used:

```
$ dpctl dump-flows tcp:127.0.0.1:6634
```

The content of manually entered flow table is shown in Figure 4.



```
mininet@mininet-vm:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x635a2b4b): flags=none type=1(flow)
  cookie=0, duration_sec=36s, duration_nsec=734000000s, table_id=0, priority=327
68, n_packets=0, n_bytes=0, idle_timeout=120,hard_timeout=0,in_port=1,actions=ou
tput:2
  cookie=0, duration_sec=16s, duration_nsec=244000000s, table_id=0, priority=327
68, n_packets=0, n_bytes=0, idle_timeout=120,hard_timeout=0,in_port=2,actions=ou
tput:1
```

Figure 4. the content of manually installed flow table

After installing a flow table with few last steps, a ping command could be executed and the destination host expected to receive the ARP and ICMP packets corresponding to the ping request. It is important to remember that the above task, the manual case, was performed by the controller automatically in SDN paradigm.

C. Second Experiment.

Although MININET is considered as a great and convenient tool, it still has some limitations. In this experiment the effect of one of its limitations on its scalability was studied. Because MININET runs on a single system, it imposes resource limitation. Those resources need to be normalized and shared among virtual hosts and switches.

A comparison between two sets of scalability tests were conducted, where, two different environments were tested. Each test environment differs from the other by the resource characteristics. The first set of scalability test was conducted by using the following environment: a microcomputer HP Compaq8200 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, the following guest operating systems were installed: 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.
The obtained scalability results are shown in Table I.

TABLE I. FIRST SCALABILITY TEST RESULTS

Topology	Node Numbers	Host Numbers	Switch numbers	Start/ stop Time (sec)
Tree	3	2	1	0,190
Tree	7	4	3	0,525
Tree	15	8	7	1,175
Tree	31	16	15	2,795
Tree	63	32	31	7,088
Tree	127	64	63	20,210
Tree	255	128	127	64,818
Tree	511	256	255	242,842

From the second set of scalability test, which was conducted with the characteristics specified in part (A) of this section, the scalability results displayed in Table 6 were obtained.

TABLE II. SECOND SCALABILITY TEST RESULTS

Topology	Node numbers	Host number s	Switch numbers	Start/ stop time (sec)
Tree	3	2	1	5,611
Tree	7	4	3	16,162
Tree	15	8	7	35,480
Tree	31	16	15	77,304
Tree	63	32	31	172,535
Tree	127	64	63	410,153
Tree	255	128	127	1054,714
Tree	511	256	255	3718,117

Comparing results of the two different scenarios mentioned above gives an indication that the amount of time required to create a virtual network increases with the increase of virtual nodes number in both scenarios, however, there is a remarkable difference in the required time between the first and second scenario which means that the resource limitation has a huge impact on the results as depicted in Figure 7.

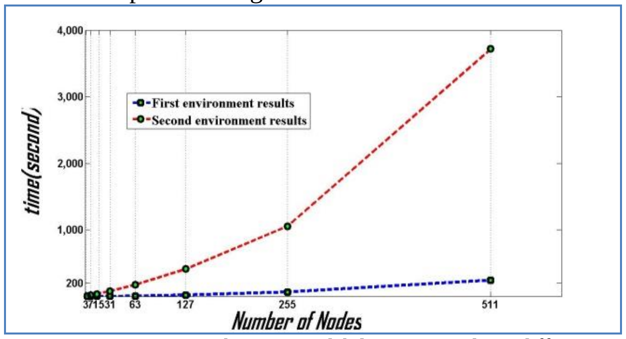


Figure 7. Comparison between scalability tests results in different environment.

V. CONCLUSIONS

In this paper the performance of Mininet tool for emulating SDN networks was evaluated.
During this study many capabilities of Mininet emulator in the SDN paradigm evaluation was covered, from the creation of basic topologies with reference controller to the ability of connection with remote controllers (in this case POX controller). In addition, this paper took into consideration the following scenarios; changing the topologies, increasing the number of nodes, controlling the behavior of forwarding hardware (switches).

The effect of simulation environment limited resources was studied and a comparison between results for two different environments was described, as a result it is noted that for small number of nodes the time required for creation (start/stop) increases from 0.190 sec in the environment of better resources to 5.611 sec in the environment of worse resources, the time required for creation (start/stop) in both environment increases with the increase in the nodes number but with different rates so that for the largest topology in our test which is 511 nodes the time increases from 242.842 sec in first environment to 3718.117sec in second environment which is 15.31 times the time required in the first environment.

It was concluded that the characteristics and qualifications of simulation environment must be taken into consideration before starting to use mininet, if the above characteristics were carefully selected then Mininet can be utilized as one of the powerful tools in emulating the SDN and virtual networks.

REFERENCES

[1] WANG Wendong, Yannan HU, Xirong QUE, GONG Xiangyang, “Autonomicity Design in OpenFlow Based Software Defined Networking”, GC'12 Workshop: The 4th IEEE International

Workshop on Management of Emerging Networks and Services©2012 IEEE.

[2] Guohui Wang, T. S. Eugene Ng, Anees Shaikh, “Programming your network at run-time for big dataapplications”,HotSDN’12, Helsinki, Finland, August 13, 2012.

[3] Rogério Leão Santos de Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, Ligia Rodrigues Prete, “Using Mininet for Emulation and Prototyping Software-Defined Networks”, Brazil, ©2014 IEEE.

[4] Kannan Govindarajan, Kong Chee Meng, Hong Ong , Wong Ming Tat,Sridhar Sivanand, Low Swee Leong, “Realizing the Quality of Service (QoS) inSoftware-Defined Networking (SDN) Based Cloud Infrastructure, Kualalumpur, Malaysia, 2014 second international conference on information and communication technology(ICoICT),©2014IEEE.

[5] Cristian Cleder Machado, Lisandro Zambenedetti Granville, Alberto Schaeffer-Filho,Juliano Araujo Wickboldt, “Towards SLA Policy Refinement for QoSManagement in Software-Defined Networking”, 2014 IEEE 28th International Conference on Advanced Information Networking and Applications.

[6] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan,N. Viljoen, M. Miller, and N. Rao, “Are we ready for sdn? Implementationchallenges for software-defined networks,” CommunicationsMagazine, IEEE, vol. 51, no. 7, pp. 36–43, 2013.

[7] O. W. Paper, “Software-Defined Networking: The New Norm for Networks,” Open Networking Foundation, Tech. Rep., April 2012.

[8] Thomas D. Nadeau, Ken Gray, “SDN: Software Defined Networks”, Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, First Edition, August 2013.

[9] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti, “A Survey of Software-Defined Networking: Past,Present, and Future of Programmable Networks”, hal-00825087,version 5- 19 Jan 2014.

[10] Joe Mambretti, Jim Chen, Fei Yeh, “Software-Defined Network Exchanges (SDXs): Architecture, Services, Capabilities, and Foundation Technologies”, Proceedings of the 2014 26th International Teletraffic Congress (ITC), ©2014 ITC.

[11] Mininet. An Instant Virtual Network on your Laptop.2014, Accessed: Sept. 2014[Online]Available: http://mininet.org.

[12] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapidprototyping for software-defined networks,” in Proceedings of the 9thACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010.

[13] The Openflow Switch, openflowswitch.org.

[14] POX, “Pox openflow controller,” 2014, Accessed: Sept.2014. [Online].Available: http://www.noxrepo.org/pox/about-pox.

[15] Python Software Foundation, “Python language reference, version 2.7,” 2014, Accessed: Sept. 2014. [Online].Available: http://www.python.org.