Mininet Network Emulator: A Review

Omran M. A. Alssaheli[1*], Z. Zainal Abidin[2], N. A. Zakaria[3]

[†]Faculty of Manufacturing Engineering, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

[††]Carbon Research Technology, Advanced Manufacturing Centre, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

Summary
Mininet is a network emulation software that allows launching a virtual network with switches, hosts, and a software-defined networking (SDN) controller on the limited resources of a single computer or virtual machine. This software was created to enable research and learning about SDN and OpenFlow, and to test SDN controllers and applications. Mininet provides convenience and practicality at a low cost because it is a free open-source software that emulates OpenFlow devices and SDN controllers. This paper presents a comprehensive and critical review of Mininet by introducing its basic concepts and implementation. Challenges in using the Mininet emulator tool are also discussed to provide insights into future research directions.

Key words:
Mininet; Network Emulator; SDN testing tool.

1. Introduction

Computer networks today are constantly evolving. Next-generation networks must be scalable, programmable, flexible and adaptable to innovative ideas. Researchers continue to work on providing innovative solutions or modifications to the current network infrastructure. However, such experiments cannot be conducted on existing operational networks as these experiments may interrupt operational traffic on those networks. Thus, customised test-beds for experimentation and testing of new innovations and new protocols, which can be solved by virtual network infrastructure, are necessary [1].

Virtualisation is the task of creating an electronic version of a physical structure and can be conducted on hardware, platforms, servers or networks [2]. Researchers' new ideas and protocols cannot be tested on real networks because doing so may affect the functions of these operational networks [1]. Only a few network devices are available for implementing software-defined networking (SDN) standards. Furthermore, implementing a network with a large number of devices is difficult because inaccurate configurations are costly and may cause unwanted problems [3]. A virtual network infrastructure is the solution to these problems. Virtual mode strategy has been conducted for prototyping and emulating network technologies; the most important of these is Mininet Emulator. The proposed study aims to enhance such a virtual network test-bed to assist researchers in their experimentation and enable students to understand the functioning of computer networks [1].

Numerous hosts, switches, SDN/OpenFlow controllers and wires have to be connected in a network. The Mininet Emulator is a simple tool that facilitates the creation of small topologies as well as large ones with hundreds or thousands of nodes and switches. Mininet is a free and open-source Python-based emulator tool that allows the creation of realistic virtual SDN networks [4]. This tool provides application programming interface (API) for automation and command line interface for interactive commanding.

Mininet is a system that allows rapidly prototyping large networks on a single computer [5]. This emulator can create a network of virtual hosts, switches, controllers, and links. Mininet hosts run on standard Linux software and the switches support the OpenFlow protocol, which is highly attractive and handy for SDN research. OpenFlow is the most prominent SDN component supported by several vendors. The OpenFlow controller is used to design a mobile cloud management system. Such an OpenFlow SDN network can be easily created by Mininet. The Python-based code developed on Mininet can be applied to real-world networks with minimal changes. In the existing Mininet, creating a custom topology other than the pre-defined ones requires Python programming skills, which could be a challenging task particularly in creating a large custom network topology. The proposed work ensures user-friendly functionality of the custom topology without the need for any programming code. All the required details of the network connectivity and characteristics are specified in a simple configuration file [1].

2. Mininet

A group of professors at Stanford University created Mininet for use as a tool in research and teaching network technologies [6]. At present, Mininet is designed to easily create virtual-software-defined networks consisting of an OpenFlow controller, a flat Ethernet network of multiple OpenFlow-enabled Ethernet switches and multiple hosts

Manuscript received September 5, 2019
Manuscript revised September 20, 2019

connected to those switches. Mininet has built-in functions that support using various types of controllers and switches. Complex custom scenarios can also be created using the Mininet Python API [7].
Several characteristics that guided the creation of Mininet are the following [8] [9]:

- Flexibility: new topologies and features can be set in software using programming languages and common operating systems.
- Applicability: correct implementations conducted on 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 scaled to large networks with hundreds or thousands of switches on a single computer.
- Realistic design: the prototype behaviour should represent real-time behaviour with a high degree of confidence, so applications and protocol stacks should be usable without any code modification.
- Shareability: the created prototypes can be easily shared with other collaborators, who can then conduct and modify the experiments.

3. Advantages of Mininet

The advantages of using Mininet are the following [3]:
1) Custom topologies: a single switch, large Internet-like topologies such as that of a data centre or anything else can be created.
2) Operation of real programs: Mininet hosts are able to run any software that can be operated on a Linux system. These tools could be tcpdump, curl, elinks, wireshark and others.
3) Customised packet forwarding: Mininet switches are programmable using the OpenFlow protocol.
4) Code sharing and result replication: anyone with a computer can run any other code once it has been packaged.
5) Ease of use: Mininet experiments can be conducted by writing simple Python scripts.

4. System Requirement for Working with Mininet

Mininet can be used on various platforms such as Windows, Linux and MAC. We ran the Mininet virtual machine (VM) on VirtualBox on a Windows 7 host computer and used XMing as X-server and Putty as SSH client. Table 1 shows the requirements for working with Mininet on a different operating system (OS) type [10].

- Laptop/Computer with at least 2 GB RAM and at least 6-8 GB of free hard disk space
- Mininet Controller (POX, NOX, Beacon, FloodLight, Maestro etc.).
- Java/Python language support.
- Mininet.

5. Mininet VM Installation

VM installation is the easiest and most frequently recommended method to install Mininet. The installation is conducted as follows [10]:

1. Download the Mininet VM image.
2. download the files corresponding to your OS,
3. Download and install a virtualisation system such as VirtualBox or VMware Workstation, which works on OS X, Windows and Linux.
4. Import Mininet VM image on virtualization system .then, it is Important to Select “settings,” and add an additional host-only network adapter that you can use log in to the VM image. Start the VM. For Mininet VM image, the user name is 'mininet' with password 'mininet'.

We can obtain native installation Mininet from Ubuntu 14.04 using the following command [11] [12]:
$ git clone git://github.com/mininet/Mininet.
The basic Mininet install can then be completed using the command
$ mininet/util/install.sh.

Table 1: Requirement for Working with Mininet

OS Type	OS Version	Virtualization Software	X Server	Terminal
Win	7+	VirtualBox	Xming	PuTTY
Win	XP	VirtualBox	Xming	PuTTY
Mac	OS X 10.7-10.9 Lion/ Mountain Lion/ Mavericks	VirtualBox	download and install XQuartz	Terminal. app (built in)
Mac	OS X 10.5-10.6 Leopard/Snow Leopard	VirtualBox	X11 (install from OS X main system DVD, preferred), or download XQuartz	Terminal. app (built in)
Linux	Ubuntu 10.04+	VirtualBox	X server already installed	gnome terminal +SSH built in

6. Mininet Topologies

Mininet topologies are basically classified into two types [7] [13].

A. Default Topologies

Mininet consists of a number of default topologies such as minimal, single, reversed, linear and tree [10]. This section explains each of these topologies. Understanding the naming method for interfaces, hosts and switches is essential for the successful use of Mininet. Switches are named s1–sN and hosts are named h1–hN. Host interfaces are prefixed with the host name followed by the Ethernet name starting with 0. First interface of host 'h1' is called 'h1-eth0' and third interface of host 'h2' is called 'h2-eth2'. First port of switch 's1' is called 's1-eth1'. On switches, numbering begins with 1.

1) Minimal Topology

Minimal is a simple topology that consists of a controller, an Open-Flow switch and two hosts by creating a link between switches and hosts, as shown in Fig. 1. Topology creation through command line is
 # mn--topo minimal as shown in Fig 2.

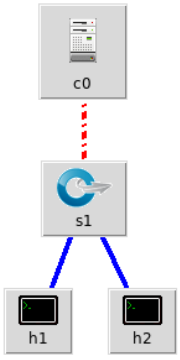

Fig. 1 Minimal Topology

```
mininet@mininet-vm:~$ sudo mn --topo minimal
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Fig. 2 Creating A Minimal Topology Using Mininet

2) Single Topology

The single topology consists of a switch and *n* number of hosts. This simple structure creates a link between the switch and hosts, as shown in Fig. 3. Topology creation through command line is
 # mn--topo single, 4 as shown in Fig 4.

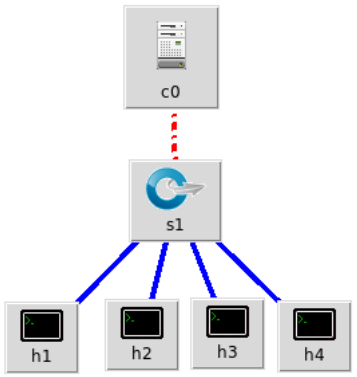

Fig. 3 Single Topology

```
mininet@mininet-vm:~$ sudo mn --topo reversed,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Fig. 4 Creating A Single Topology Using Mininet

3) Reversed

Reversed topology is similar to the single topology but has a connection in reverse order. Topology creation through command line is
 # mn--topo Reversed, 4 as shown in Fig 5.

```
mininet@mininet-vm:~$ sudo mn --topo linear,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

Fig. 5 Creating A Reversed Topology Using Mininet

4) Linear

A linear topology contains *n* number of switches and *n* number of hosts. It creates a link between each switch and each host and among the switches in linear order, as shown in Fig. 6. Topology creation through command line is
 # mn--topo Linear, 4 as shown in Fig 7.

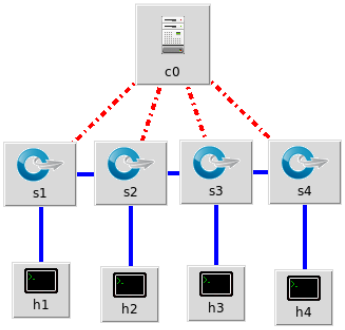

Fig. 6 Linear Topology

```
mininet@mininet-vm:~$ sudo mn --topo linear,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

Fig. 7 Creating A Linear Topology Using Mininet

5) Tree

This topology consists of *n*-level of switches. Hosts are attached to lower-level switches as shown in Fig. 8. Topology creation through command line is
 # mn--topo Tree, 3 as shown in Fig 9.

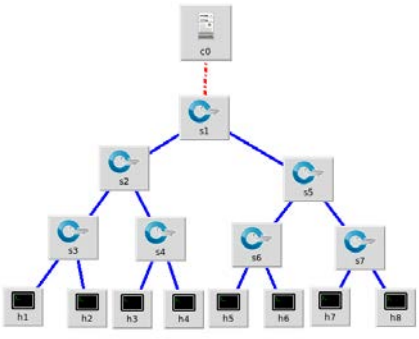

Fig. 8 Tree Topology

```
mininet@mininet-vm:~$ sudo mn --topo tree,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6)
 (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet>
```

Fig. 9 Creating A Tree Topology Using Mininet

B. Custom topologies

Mininet can be created by using Python code [14]. These topologies are run in Mininet through specific commands. Python API [5] uses its own classes, methods, functions and variables to create these topologies. For example, creating a custom topology with 2 switches and 5 hosts, as shown in Fig. 11, only requires writing a few lines of Python code as shown in Fig. 12. A highly complex, flexible and robust topology can also be created. This topology can be configured based on relevant parameters and reused for multiple experiments.

```
mininet@mininet-vm:~$ sudo mn --custom ./mininet/custom/omran.py --topo=mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

Fig. 10 Creating A Custom Topology Using Mininet

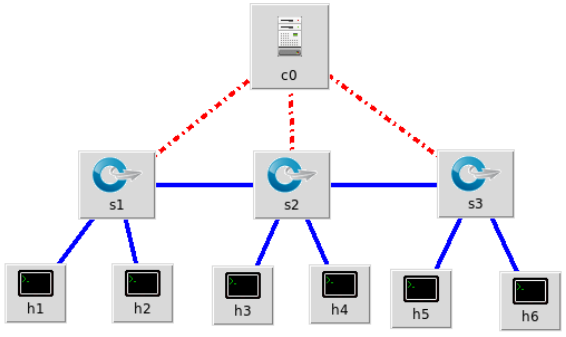

Fig. 11 Custom Topology

```
""" Custom topology example:

3  directly connected switches plus a 2 host for each switch:

        s1--------s2--------- s3
       / \       / \         / \
      /   \     /   \       /   \
     h1   h2   h3   h4     h5   h6
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):

        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        Host1 = self.addHost( 'h1' )
        Host2 = self.addHost( 'h2' )
        Host3 = self.addHost( 'h3' )
        Host4 = self.addHost( 'h4' )
        Host5 = self.addHost( 'h5' )
        Host6 = self.addHost( 'h6' )
        Switch1 = self.addSwitch('s1')
        Switch2 = self.addSwitch('s2')
        Switch3 = self.addSwitch('s3')
        # Add links
        self.addLink( Host1, Switch1 )
        self.addLink( Host2, Switch1 )
        self.addLink( Host3, Switch2 )
        self.addLink( Host4, Switch2 )
        self.addLink( Host5, Switch3 )
        self.addLink( Host6, Switch3 )
        self.addLink( Switch1, Switch2 )
        self.addLink( Switch2, Switch3 )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Fig. 12 Custom Topology using Python code

7. Mininet Controller

SDN controllers direct traffic according to forwarding policies set by a network operator, thereby minimising manual configurations for individual network devices. By taking the control plane off the network hardware and running it instead as software, the centralised controller facilitates automated network management and enables easy integration and implementation of business applications. In effect, the SDN controller serves as an OS for the network [15]. The five most important open-source controllers that can be used by Mininet remotely are POX, Ryu, Trema, FloodLight and OpenDaylight [16]. Table 2 presents the specifications of various SDN controllers [17, 18].

A. POX

NOX is the first OpenFlow controller but is no longer being developed. POX is a new version of NOX [19] and is a software platform developed in Python. POX began early as a controller for an OpenFlow protocol. However, it can act as an OpenFlow switch and can be used to develop networking software. POX works with Python 2.7 (and even with Python 2.6), and can run under Linux OS, Mac OS and Windows. The core and main modules are developed in Python [8].

B. Ryu

Ryu is an open-source Python implementation of an SDN controller with a component-based architecture [19]. It has a set of well-defined APIs for network application development and has been widely used by the research community. Ryu also supports multiple protocols in a single bilateral integration for hardware integration and configuration. The event handler of a Ryu controller dispatches the events to all of the subscribed applications [20].

C. Trema

Trema [15] is an OpenFlow controller framework developed on Ruby and C. It is basically a framework that includes libraries and functional modules, which work as an interface to OpenFlow switches. Several sample applications are provided to enable the execution of various controllers, thereby facilitating the extension to new features.

D. Open DayLight

Open Daylight is an open-source project with a modular, pluggable and flexible controller platform at its core. It is implemented in Java, and therefore can be deployed on any hardware and operating system platform that supports Java [22].

E. Floodlight

The Floodlight open SDN controller is an Apache-licensed Java-based OpenFlow controller that provides an extension to ensure the security of the OpenFlow protocol. This OpenFlow-based application provides an interface on northbound and southbound sides [23]. It is supported by a group of developers, including engineers from Big Switch Networks. OpenFlow protocols are an open standard managed by the Open Networking Foundation. This standard specifies a protocol by switching a remote controller that can modify the behaviour of networking devices through a well-defined ‘forwarding instruction set’. Floodlight is designed to work with an increasing number of switches, routers, virtual switches and access points that support the OpenFlow standard. The features of Floodlight are the following [24].

- a module-loading system that facilitates extensions and improvements;
- a simple set-up with minimum accreditation;

- support for a wide range of virtual and physical OpenFlow switches and non-OpenFlow networks that can manage multiple OpenFlow switches;
- high-performance design; and
- Support for OpenStack (link) cloud synchronisation platform.

Table 2: SDN controllers

	POX	Ryu	Trema	FL	ODL
Language Support	Python	Python	C Ruby	Java	Java
OpenFlow Support	v1.0	v1.0	v1.0	v1.0	v1.0
Open Source	Yes	Yes	Yes	Yes	Yes
GUI	Yes	Yes	no	Web GUI	Yes
REST API	no	Yes	no	Yes	Yes
Platform Support	Linux Mac Windows	Linux	Linux	Linux	Linux Mac Windows
Created by	Niciria Networks	NTT	NEC	Big Switch Netwo-rks	Cisco and ODL

8. Results and Findings

A study and evaluation of SDN emulation tool termed Mininet, was recommended by Scheweitzer, Prete, De Oliveira & Shinoda in [25]. Preliminary investigations propose that the ability of quick and simple prototyping, the ensuring applicability, the chances of sharing tools and results at zero cost are encouraging factors that enable scientists to carry their investigations forward despite the tool's disadvantages in terms of performance fidelity between the real and simulated environment. Following presentation of some of this paradigm's concepts, its appearance's purpose, its parts and mechanism, several net prototypes were generated to provide better understanding of the Mininet tool. An evaluation was also carried out to show its pros and cons. Extensible Service ChAin Prototyping Environment (ESCAPE) was proposed by Csikor, Sahhaf, Sonkoly and Csoma in [26] via Mininet, Click, NETCONF and POX. They ran a similar prototyping system termed ESCAPE, which is able to generate and test different parts of the service chaining architecture. Click was incorporated by this framework for running Virtual Network Functions (VNF), NETCONF for running Click-based VNFs and POX for managing the traffic steering. They also incorporated their extendable Orchestrator module, which is able to hold mapping algorithms from abstract service descriptions that were to be ran service chains.

This framework incorporates Click for implementing Virtual Network Functions (VNF), NETCONF for managing Click-based VNFs and POX for taking care of traffic steering. They also add their extensible Orchestrator module, which can accommodate mapping algorithms from abstract service descriptions to deployed and running service chains. Kuzmin, Fomichev, Petrov & Zabrovskiy suggested an investigation of the delivery of media content through the Internet via MPEG-DASH technology within the network emulation environment [27]. A real hardware client and a server from an actual IP-network was linked to Mininet. A Mininet setting that permitted embedding of this virtual environment into the current network infrastructure was presented by them. The investigation demonstrated that the communication channel's bandwidth variation emulated in Mininet produces the same outcome on streaming video in comparison to experimental results that are collected from a specialized hardware-software network emulator that has similar channel property configurations. They made a conclusion that the Mininet may be regarded as a practical instrument for the emulation of video stream transmissions that are using Dynamic Adaptive Streaming over HTTP, and also to be utilized for the generation of novel adaptive control algorithms.

Prete, Gerola, Salvadori, Salsano, Siracusano and Ventre recommended the architecture and services of a hybrid IP/SDN networking scenario in [28]. They also provided an elaboration of the architecture and mechanism of an Open Source Hybrid IP/SDN (OSHI) node. It was a combination of Quagga for OSPF routing and Open vSwitch for OpenFlow based switching on Linux. The SDN's evolution is heavily reliant on the availability of tools for experimental validation and performance assessment of SDN solutions. A set of open source instruments was provided by them to enable the facilitation of the design of hybrid IP/SDN experimental networks, their function on Mininet or on distributed SDN research testbeds and their test. Lastly, based on provided tools, they assess key performance parameters of the proposed solutions. The test environment and the OSHI development is available in a VirtualBox VM image that is downloadable.

A characterization of the scalability and performance of Mininet through an experimental analysis was presented by Novillo, Ortiz and Londoño in [29]. They utilized a typical topology for large data centers for this purpose. The results show that under certain restrictions, Mininet provided results that are in accordance with the theoretical expectation. In other results, the emulation platform was shown to be heavily reliant on the underlying software/hardware system's performance, as well as on the mode of operation of the SDN controller utilized in the network.

Mininet-WiFi was suggested by Afzal, Santos, Rothenberg, Fontes and Brito in [30], as an instrument to provide an emulation of wireless OpenFlow/SDN scenarios that permit high-fidelity experiments that mirrors authentic networking environments. Mininet-WiFi is able to augment the Mininet emulator with virtual wireless

stations and access points while retaining the original SDN abilities and lightweight virtualization software architecture. They also provide elaborations on the potential uses of Mininet-WiFi, as well as the pros and current cons.

Seçinti, Özçevik, Canberk, Teoman, & Erel provided a Mininet-based Software-Defined Network (SDN) simulation environment in [31], that enhanced the overall network's total flow throughput and scalability. Seçinti, Özçevik, Canberk, Teoman, & Erel provided a Mininet-based Software-Defined Network (SDN) simulation environment in [31], that enhanced the overall network's total flow throughput and scalability. Mininet serves as an appropriate and manageable tool to run the suggested SDN based flow admission control module, which would then allow the configuration of the whole topology since it already contains in-built Open-Flow switches and virtual controllers. This open-source platform can also be configured easily through its drag and drop abilities. In this Demo for the Control Plane, an OpenDaylight controller is utilized to provide a simulation of the flow admission control module that fairly allows flow into the OpenFlow switches. OpenFlow version 1.3 for relaying information between Control plane and separated Data, and operating systems based on Linux to generate Mininet 2.1.0, are carried out in the simulator environment.

Veena, Murthy, Rustagi and Pal gave a custom topology framework in [2], which allowed the generation of any custom network topology of user's choice, which includes several IP networks. This way, users are able to run research on distinct networks that possess distinct broadcast domains. The framework has been made in such a way that it does not severely impede the current Mininet's performance.

9. Discussion

Considering the increasing popularity of SDN within cloud computing field and its associated applications, there has been simultaneous increase in interests over the SDN network configuration's behaviour modelling. Since SDN is an open source network emulation software, it can be run easily and investigated with Mininet. There was a presentation of SDN tools which were then utilized for prototyping and simulation. This study also included references of other investigators' result. The Mininet tool is imperative for SDN investigations. In reality, several seconds of waiting for complete larger topologies to commence is rather reasonable and quicker than the hardware switches' boot time. In addition, several positive variables such as the applicability safety, ability of quick and simplified prototyping, opportunities to share results and tools at minimum cost may boost their investigations. A critical disadvantage of Mininet is that it can emulate networks by utilizing slower links and it is not particularly adept for high speed links. This is due to the fact that packets are carried forward by a collection of software switches that co-share memory and CPU resources, and are more accustomed to having lower performance than dedicated switching hardware. It is admittedly challenging when using Mininet to emulate networks that spread over a hug scale.

10. Conclusion

A review had been conducted to identify the benefits of using the Mininet emulator. This software was proven to be a handy tool for networking researchers to emulate real operational networks and test innovative ideas and new protocols. Mininet is also suitable in investigating the behavioural characteristics of SDN because it is OpenFlow-enabled. Furthermore, experiments on topologies were conducted and SDN controllers that can be utilized by this emulator tool were enumerated. Finally, challenges posed by using the Mininet emulator tool were discussed, such as its inefficiency in high-speed links and the difficulty encountered in its application to large-scale networks.

References

[1] Veena, S., Pal, C., Rustagi, R. P., & Murthy, K. N. B. (2014). A Framework for Implementing Realistic Custom Network Topology in Mininet. International Journal of Science and Research (IJSR), (7), 1316-1323.

[2] Pal, C., Veena, S., Rustagi, R. P., & Murthy, K. N. B. (2014, February). Implementation of simplified custom topology framework in Mininet. In 2014 Asia-Pacific Conference on Computer Aided System Engineering (APCASE) (pp. 48-53). IEEE.

[3] Gupta, V., Kaur, K., & Kaur, S. (2016, March). Network programmability using software defined networking. In 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1170-1173). IEEE.

[4] Sharma, K. K., & Sood, M. (2014). Mininet as a container based emulator for software defined networks. International Journal of Advanced Research in Computer Science and Software Engineering, 4(12).

[5] Lantz, B., Heller, B., & McKeown, N. (2010, October). A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (p. 19). ACM.

[6] Sood, M. (2015). SDN and Mininet: Some Basic Concepts. International Journal of Advanced Networking and Applications, 7(2), 2690.

[7] Kaur, K., Singh, J., & Ghumman, N. S. (2014, February). Mininet as software defined networking testing platform. In International Conference on Communication, Computing & Systems (ICCCS) (pp. 139-42).

[8] Keti, F., & Askar, S. (2015, February). Emulation of software defined networks using mininet in different

simulation environments. In 2015 6th International Conference on Intelligent Systems, Modelling and Simulation (pp. 205-210). IEEE.
[9] De Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., & Prete, L. R. (2014, June). Using mininet for emulation and prototyping software-defined networks. In 2014 IEEE Colombian Conference on Communications and Computing (COLCOM) (pp. 1-6). IEEE.
[10] Kumar, D., & Sood, M. (2016). Software defined networks (SDN): experimentation with mininet topologies. Indian Journal of Science and Technology, 9(32).
[11] DeCusatis, C., Carranza, A., & Delgado-Caceres, J. (2016). Modeling Software Defined Networks using Mininet. In Proc. 2nd Int. Conf. Comput. Inf. Sci. Technol. Ottawa, Canada (No. 133, pp. 1-6).
[12] Topoloi, S. G., & Borcoci, E. (2018, June). Software Defined Networking and Network Function Virtualisation Cooperation-Experiments. In 2018 International Conference on Communications (COMM) (pp. 281-286). IEEE.
[13] Bholebawa, I. Z., & Dalal, U. D. (2016). Design and performance analysis of OpenFlow-enabled network topologies using Mininet. International Journal of Computer and Communication Engineering, 5(6), 419.
[14] Jiang, J. R., Huang, H. W., Liao, J. H., & Chen, S. Y. (2014, September). Extending Dijkstra's shortest path algorithm for software defined networking. In The 16th Asia-Pacific Network Operations and Management Symposium (pp. 1-4). IEEE.
[15] Fernandez, M. P. (2013, March). Comparing openflow controller paradigms scalability: Reactive and proactive. In 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA) (pp. 1009-1016). IEEE.
[16] Izquierdo-Zaragoza, J. L., Fernandez-Gambin, A., Pedreno-Manresa, J. J., & Pavon-Marino, P. (2014, June). Leveraging Net2Plan planning tool for network orchestration in OpenDaylight. In 2014 International Conference on Smart Communications in Network Technologies (SaCoNeT) (pp. 1-6). IEEE.
[17] Kaur, S., Singh, J., & Ghumman, N. S. (2014, February). Network programmability using POX controller. In ICCCS International Conference on Communication, Computing & Systems, IEEE (Vol. 138).
[18] Khattak, Z. K., Awais, M., & Iqbal, A. (2014, December). Performance evaluation of OpenDaylight SDN controller. In 2014 20th IEEE international conference on parallel and distributed systems (ICPADS) (pp. 671-676). IEEE.
[19] Stancu, A. L., Halunga, S., Vulpe, A., Suciu, G., Fratu, O., & Popovici, E. C. (2015, October). A comparison between several software defined networking controllers. In 2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS) (pp. 223-226). IEEE.
[20] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., & Smeliansky, R. (2013, October). Advanced study of SDN/OpenFlow controllers. In Proceedings of the 9th central & eastern european software engineering conference in russia (p. 1). ACM.
[21] Arbettu, R. K., Khondoker, R., Bayarou, K., & Weber, F. (2016, September). Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers. In 2016 17th International telecommunications network strategy and planning symposium (Networks) (pp. 37-44). IEEE.
[22] Izquierdo-Zaragoza, J. L., Fernandez-Gambin, A., Pedreno-Manresa, J. J., & Pavon-Marino, P. (2014, June). Leveraging Net2Plan planning tool for network orchestration in OpenDaylight. In 2014 International Conference on Smart Communications in Network Technologies (SaCoNeT) (pp. 1-6). IEEE.
[23] Morales, L. V., Murillo, A. F., & Rueda, S. J. (2015, September). Extending the floodlight controller. In 2015 IEEE 14th International Symposium on Network Computing and Applications (pp. 126-133). IEEE.
[24] Taher, A. (2014). Testing of floodlight controller with mininet in sdn topology. ScienceRise, (5 (2)), 68-73.
[25] De Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., & Prete, L. R. (2014, June). Using mininet for emulation and prototyping software-defined networks. In 2014 IEEE Colombian Conference on Communications and Computing (COLCOM) (pp. 1-6). IEEE.
[26] Csoma, A., Sonkoly, B., Csikor, L., Németh, F., Gulyas, A., Tavernier, W., & Sahhaf, S. (2014, August). ESCAPE: Extensible service chain prototyping environment using mininet, click, netconf and pox. In ACM SIGCOMM Computer Communication Review (Vol. 44, No. 4, pp. 125-126). ACM.
[27] Zabrovskiy, A., Kuzmin, E., Petrov, E., & Fomichev, M. (2016, April). Emulation of dynamic adaptive streaming over HTTP with Mininet. In 2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT) (pp. 391-396). IEEE.
[28] Salsano, S., Ventre, P. L., Prete, L., Siracusano, G., Gerola, M., & Salvadori, E. (2014). OSHI-Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds). arXiv preprint arXiv:1404.4806.
[29] Ortiz, J., Londoño, J., & Novillo, F. (2016, October). Evaluation of performance and scalability of Mininet in scenarios with large data centers. In 2016 IEEE Ecuador Technical Chapters Meeting (ETCM) (pp. 1-6). IEEE.
[30] Fontes, R. R., Afzal, S., Brito, S. H., Santos, M. A., & Rothenberg, C. E. (2015, November). Mininet-WiFi: Emulating software-defined wireless networks. In 2015 11th International Conference on Network and Service Management (CNSM) (pp. 384-389). IEEE.
[31] Erel, M., Teoman, E., Özçevik, Y., Seçinti, G., & Canberk, B. (2015, November). Scalability analysis and flow admission control in mininet-based SDN environment. In 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN) (pp. 18-19). IEEE.

Omran Maki Abdelsalam Alssaheli holds Bachelor of Computer Science at Higher Institute for Comprehensive Professions - Sebha, Libya in 2010. He received Master of Computer Science (Software Engineering) from Universiti Tun Hussein Onn Malaysia (UTHM), Malaysia in 2016. He is currently pursuing his PhD in Information and communications technology (Networking) from Universiti

Teknikal Malaysia Melaka, Malaysia. Research interest in computer science, computing and digital making, networking, and Internet-of-Things (IoT). Contact: imran20032006@yahoo.com

Zaheera Zainal Abidin received Bachelor of Information Technology from University of Canberra, Australia in 2002. She joined ExxonMobil Kuala Lumpur Regional Center as a Project Analyst in 2000-2001. She completed her MSc. in Quantitative Sciences (2004), MSc. in Computer Networking (2008) and PhD in I.T. and Quantitative Sciences (2016) from Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam, Selangor. She served as a lecturer at Universiti Kuala Lumpur (2005-2009) and senior lecturer & researcher in Universiti Teknikal Malaysia Melaka (2009 – present). She is a member of Information Security, Forensics and Networking (INSFORNET) research group. She is one of the certified CISCO Academy (CCNA) in computer networking field and certified Internet-of-Things specialists. Research interest in Internet-of-Things (IoT), biometrics, network security and image processing. Contact: zaheera@utem.edu.my

Nurul Azma Zakaria holds a B.Eng degree in electronic computer system from University of Salford, UK. She received the MSc in Information System Engineering and PhD in Information and Mathematical Sciences from UMIST, UK and Saitama University, Japan, respectively. She is currently a senior lecturer at Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka (UTeM) and also a member of Information Security, Forensics and Networking (INSFORNET) research group. Her area of research interests include computer system and networking, embedded system design, IoT devices and application, and IPv6 Migration. Contact: azma@utem.edu.my