American Journal of Software Engineering and Applications
2014; 3(6): 74-82
Published online December 16, 2014 (http://www.sciencepublishinggroup.com/j/ajsea)
doi: 10.11648/j.ajsea.20140306.12
ISSN: 2327-2473 (Print); ISSN: 2327-249X (Online)

Survey of software components to emulate OpenFlow protocol as an SDN implementation

Mohammed Basheer Al-Somaidai, Estabrak Bassam Yahya

Dept. of Electrical Engineering, Mosul University, Mosul, Iraq

Email address:
mohammedbasheerabdullah@uomosul.edu.iq (M. B. Al-Somaidai), eng_est_1990@yahoo.com (E. B. Yahya)

To cite this article:
Mohammed Basheer Al-Somaidai, Estabrak Bassam Yahya. Survey of Software Components to Emulate OpenFlow Protocol as an SDN Implementation. *American Journal of Software Engineering and Applications*. Vol. 3, No. 6, 2014, pp. 74-82.
doi: 10.11648/j.ajsea.20140306.12

Abstract: Software Defined Networks (SDN) is the next wave in networking evolution. It may be considered as a revolution rather than an evolution since; many concepts of conventional network protocols are reshaped. OpenFlow protocol is the most widely deployed protocol in SDN. Emulation of OpenFlow based network projects facilitates the implementation of new ideas and driving the development of the protocol. In this paper, a summary of many software components related to OpenFlow is presented. Most of these software components were tested by the researchers in order to simplify the choice for other researchers considering the implementation of OpenFlow projects. These tests showed that there are differences in performance for the controllers that support OpenFlow 1.0 and OpenFlow 1.3. Furthermore, the tested controllers differs in the applications they support.

Keywords: Software Defined Network, OpenFlow, Emulation, Mininet

1. Introduction

A new paradigm in the field of networking is the software defined networks a promising architecture, which is gaining rapid attention of researchers and vendors as well [1-3]. This is so because; the unlimited development of network applications and the extensive demands of an explosive growth in network users are driving conventional network devices to their limits. SDN introduces a new way to handle the vast amount of packets traversing the network. Many packets belong to a single flow; thus, handling that flow and distributing the actions to be taken to all its packets would numerously speed up their forwarding. This is only one of many other benefits of a centralized control of the network. The most widely deployed SDN architecture is the OpenFlow protocol. Many gigantic Internet vendors including Google are considering the application of OpenFlow protocol in their data centers [4]. A gradual implementation of SDN and OpenFlow suggests the co-existence of OpenFlow networks with conventional networks. This requires extensive studies and projects to investigate the limitations and possibilities of these protocols.

Simulation and emulation of network projects provide a solid base to determine their pros and cons. Emulation is more realistic than simulation since, it must be carried out in real time and could provide a way to some real devices running real operating systems to interact with some simulated devices [5].

B. Lantz, et. al. [6] analyzed the performance of Mininet emulator to develop, interact with, and customize the SDN concept with OpenFlow protocol. This study showed Mininet ease of use, scalability, and limitations.

S. Wang, et. al. [5] introduced the EstiNet OpenFlow network simulator and emulator, and studied its performance to design SDN networks. They compared EstiNet behavior, capabilities and scalability with Mininet and ns-3 platforms.

A. Shalimov, et. al. [7] proposed a method to test and compare popular open source SDN/OpenFlow controller. They analyzed throughput, latency, scalability and security by developing new framework called Hcprobe based on Cbench framework.

B. Nunes, et.al. [8] provided historic review about programmable network idea from its beginning time down to the SDN revolution. The study presented the architecture of SDN and discussed OpenFlow features, application and related software to deploy and develop SDN networks based on OpenFlow.

A. Lara, et. al. [9] discussed the architecture of OpenFlow

network to understand SDN, and centralized control concept by using different controllers' platform. In addition, studies have measured the performance of OpenFlow networks through modeling and experimentation. The researchers clarify the challenges facing the large-scale OpenFlow networks and applications.

The rest of this paper is structured as follows: in section 2 we briefly discuss software defined network architecture. Section 3 introduces an overview of OpenFlow protocol, its fundamental concepts and messages. Some SDN and OpenFlow platforms are presented and compared in section 4; while, sections 5 and 6 give a survey of OpenFlow software controllers and switches respectively. Any OpenFlow project could make use of some tools that are presented in section 7. Finally, section 8 contains some concluding remarks and future work suggestions.

2. Software Defined Networks (SDN)

The adaptation of packet switching in networking made each network device such as gateway, router, or switch a standalone device. These devices manage themselves independently even if this management was according to a certain routing protocol or administration policy. Each data packet undergoes the same parsing and processing efforts at each network node even if it belongs to the same flow. This conventional architecture of networks may fail to support the dramatically increase in users' requirements and the fast deployments of new network applications.

Segregating the control plane and the management plane from the data forwarding plane in network devices is what software defined network (SDN) about [10,11]. In such a paradigm, a central controller is responsible for managing many forwarding devices that lay under its supervision. Such configuration would results in efficient, faster innovative, and more scalable networks that meet users' demands. Software defined network is managed through a network operating system implemented at the controller to make all the subsequent switches work in harmony and more flexibility.

These switches need not be in the same geographical area; the management of many planet wise distributed data centers that belong to a cloud service provider is an example of this diverse distribution of forwarding devices [2]. Fig. 1, shows the architecture of a software defined network. It is worth to mention that SDN is not a protocol; but it is an operational and programming architecture. Albeit, SDN uses certain protocols for making the network programmable. These could be OpenFlow [12], I2RS, PCE-P, BGP-LS, NetConf/Yong, and OMI [11]. In this paper, we are focusing on the widely deployed OpenFlow protocol.

3. Open Flow

OpenFlow started at Stanford University in 2008 [13]. The aim of the project was to give researchers a tool to implement their experimental protocols in networks. OpenFlow network consists of three major components: a controller, an OpenFlow switch, and the OpenFlow protocol.

The Open Networking Foundation (ONF) a non-profit

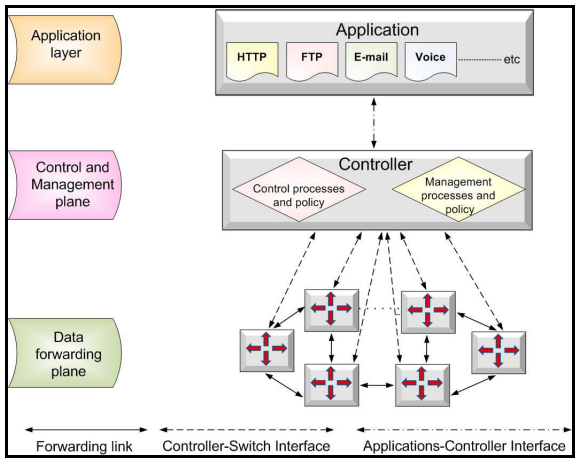

Fig 1. *Architecture of a software defined network.*

organization was created in 2011 by a group of vendors [14]. It is dedicated to coordinating the development of SDN standards and solutions in order to accelerate the delivery of SDN products, services, and applications. Since then ONF had published each new version of OpenFlow standard. Up to the date of writing this paper (March 2014) the last version of OpenFlow switch specification is 1.4 and it was published in October 2013 [12]. According to this specification, the architecture of an OpenFlow switch should contain the blocks shown in Fig. 2, each OpenFlow switch contains one or more flow tables processed in pipeline, a single group table, a single meter table; and a various types of ports. Each table and port in the OpenFlow switch is associated with many counters that could gather various statistics describing the events that the switch is subjected to. The controller creates all the tables and their entries; the data packets that traverse the OpenFlow switch update the counters.

The corner stone in the OpenFlow protocol is the flow table, which has 256 entries. Each entry in the flow table contains six sections as shown in Fig. 3.

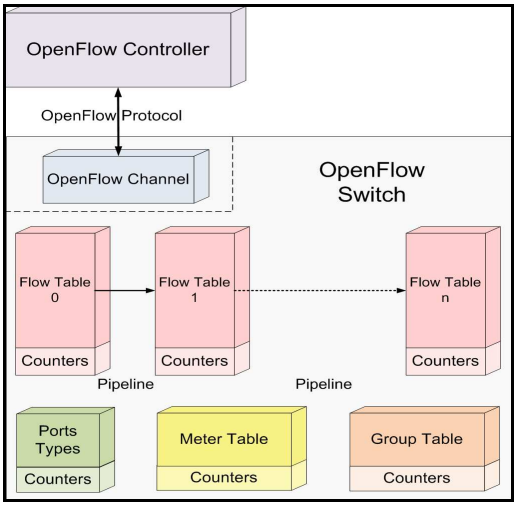

Fig 2. *Architecture of an OpenFlow switch*

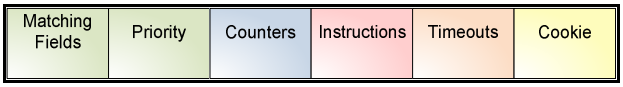

Fig 3. *OpenFlow switch flow table entry fields.*

The matching fields section is used to match the packet with the entry according to various packet header fields. When more than one entry match a packet the priority field determines the flow table entry that will be executed and the per flow table entry counters are updated.

The instructions section contains among other things the actions that will be acted upon the matched packet. The timeouts field specifies the maximum amount of hard time and idle time before the flow table entry expires. A zero value in any of them disable the corresponding timer. The hard timeout determines the maximum amount of time in seconds before the flow table entry expires; while the idle time out causes the expiration of the entry if it has matched no packet in the given number of seconds. The cookie field is used by the controller to filter flow statistics, flow modification, and flow deletion. Each flow table must support a table-miss flow entry clarifying the action that should be taken upon the unmatched packet either sending it to the controller, dropping it, or directing it to the subsequent flow table in the pipeline [12].

OpenFlow protocol has three types of messages to communicate between the controller and the OpenFlow switch over a secure channel or over a TCP channel as shown in Fig.4. They are classified according to the initiator of the message into controller to switch messages, asynchronous (switch to controller) messages, and symmetric messages. The controller to switch messages are used to assert its control upon the switch, reading the switch status, and modifying the switch states which includes editing the switch flow tables.

The switch to controller messages are used to inform the controller about a new incoming flow, a change in a switch state; or a request for modifying a flow table entry. Either the controller or the switch could initiate the symmetric messages. They include hello messages, echo messages, error messages, and experimenter message that identify the vendor of the controller or the switch [12]. Table 1. shows a summary of OpenFlow switch standards specification properties. It can be observed that almost every year there is a new version in the 1.x numbering of the standard, and although OpenFlow protocol is still in its 1.x version, there is huge development every year.

4. SDN Development Platforms

There are many platforms that could be used by researchers to emulate and/or simulate their SDN projects. Researchers use these tools to perform experiments, study the behavior of the network, and develop new methods to support different applications. In this section, a description of these currently available SDN platforms is presented emphasizing on the rapidly developed and deployed Mininet platform. Table 2. gives an overview of some properties of these platforms.

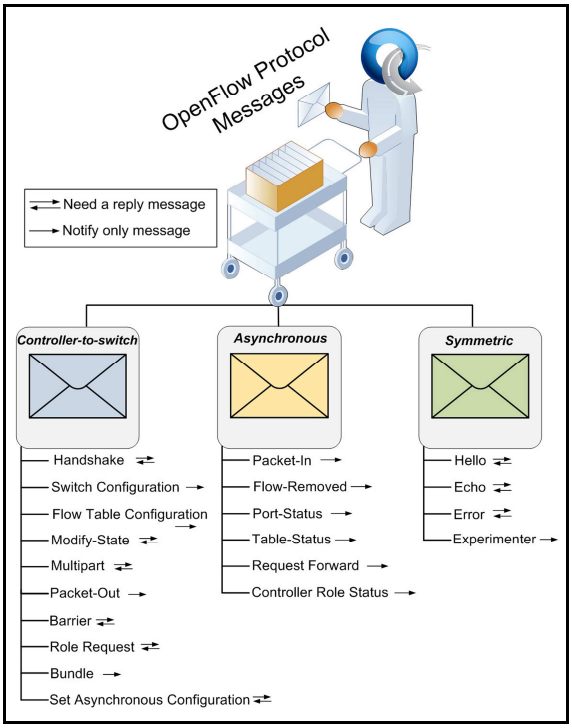

Fig 4. *OpenFlow protocol messages.*

Table 1. *OpenFlow switch standards properties.*

Version / Property	1.0	1.1	1.2	1.3	1.4
Publication date	Dec. 31, 2009	Feb. 28, 2011	Dec. 5, 2011	Jun. 25, 2012	Oct. 15, 2013
Widely deployed	Yes	No	No	No	No
Flow table	single	multiple	multiple	multiple	multiple
Group table	No	Yes	Yes	Yes	Yes
Meter table	No	No	No	Yes	Yes
VLAN and MPLS Tag	No	Yes	Yes	Yes	Yes
Controller connection failer	Emerg-ency	Stand alone /	Stand alone / secure	Stand alone /	Stand alone /
IPv6 support	No	No	Yes	Yes	Yes
Multiple controller	No	No	Yes	Yes	Yes
Eviction /Vacancy/Synchronization	No	No	No	No	Yes
Optical ports	No	No	No	No	Yes

Table 2. *Properties of SDN platforms*

Platform	Mininet	EstiNet	ns-3	Trema
Last version	2.1.0+	8.0	3.19	0.4.6
Vendor	Stanford University, ON. Lab	EstiNet Technologies Inc.	ns-3 Project	NEC Corporation
Web site	www.mininet. org	www.estinet. com	www.nsnam. org	io/trema/ Trema.github.
Operating system	Ubuntu, Fedora	Linux Fedora (14,17)	GNU/Linux, Windows, FreeBSD, Mac, OSX	GNU/ Debian, Ubuntu, Fedora
OpenFlow versions	1.0 – 1.3	1.0, 1.1, 1.3, 1.3.2	0.89	1.0, 1.3, 1.3.1
GUI	VND*, Miniedit	EstiNet GUI	VND*	VND*
Emulation mode	Yes	Yes	No	Yes
Simulation mode	No	Yes	Yes	No
Free / Proprietary	Free	Proprietary	Free	Free

*VND: Visual Network Description, to be mentioned in section 7

4.1. Mininet

Mininet is a network emulation platform that supports rapid development in SDN using OpenFlow protocol. It is the most popular SDN platform used by SDN researchers due to its simplicity, availability, and flexibility. Furthermore, Mininet is entirely devoted to OpenFlow architecture [6].

Mininet uses Linux kernels along with Python language scripts to construct a virtual network of large number of hosts network, OpenFlow switches, and controllers in any network topology the researcher employs over a single desktop or laptop station.

Mininet could use its built-in software tools to develop such networks through Command Line Interface (CLI), or adapts to a third-party software tools that implement other controllers or Graphic User Interface (GUI) engines [15, 16]. It has the flexibility of adding many controller types that will be mentioned in section 5.

4.2. EstiNet

EstiNet is an emulation and simulation platform of many network protocols; one of them is OpenFlow protocol. It also supports some of the controllers of section V. EstiNet is a proprietary software tool and it uses the company servers to run the simulation or the emulation projects. This cloud service is referred to as Simulation as a Service [17].

EstiNet has good simulation properties among them are accurate and repeatable result with a graphical user interface and packet animation along with good presentation of the simulation statistics as a graph for each node in the network [5].

4.3. ns-3

ns-3 is a well established network simulator usually compared to OPNET for providing simulation environment to a wide range of network protocols. ns-3 supports OpenFlow protocol and its switches in simulator environment but it cannot readily run a real OpenFlow controller such as NOX, POX, or Floodlight without modifications. This is why ns-3 has implemented its own OpenFlow controller as a C++ module with a different performance from the above real controllers.

Another drawback of using ns-3 is that it until now supports version 0.89 of OpenFlow protocol only, this limits the researchers' ability to test and develop projects that are compatible with the new versions of OpenFlow protocol [5]. It could be used to introduce the concepts of SDN and OpenFlow to beginners who are used to ns-3.

4.4. Trema

Trema is an OpenFlow framework that includes everything the researcher needs to conduct an OpenFlow project. The source tree includes basic libraries and functional modules that work as an interface to OpenFlow switches. Several examples of sample applications are also provided.

It has an integrated testing and debugging environment that manage, monitor, and diagnose the entire system with a network emulator and a diagnostic tool chain (Trema shark, Wireshark plug-in) [18]. The lack of a graphical user interface and the use of the programming languages C and Ruby may limit the popularity of this platform.

5. Controller Software

A block diagram of the controller; which is the brain of any software defined network is shown in Fig. 5. The controller communicates with the forwarding devices through an SDN protocol such as OpenFlow. This link is also called the southbound Application Programming Interface (API). From the other side the controller uses a northbound API to deal with various applications. If we made an analogy for the network as an orchestra then the controller plays the role of the maestro. In fact, some SDN implementations use these designations to refer to SDN and the controller [11, 19].

As the basic concept of SDN is to decouple the control

plane and the management plane from the data-forwarding plane then the controller has to bear all the burden of controlling and managing all the data forwarding devices. It should maintain and update through the rule-placement algorithm information about all the forwarding devices that are under direct responsibility of the controller including their flow tables, links, and states. The routing policy is another task of the controller any change in any forwarding device state causes the controller to reshape the routing path of all flows traverse that device resulting in updates to a large number of switches' flow tables. Security strategies along with end devices policy are also, placed in the controller.

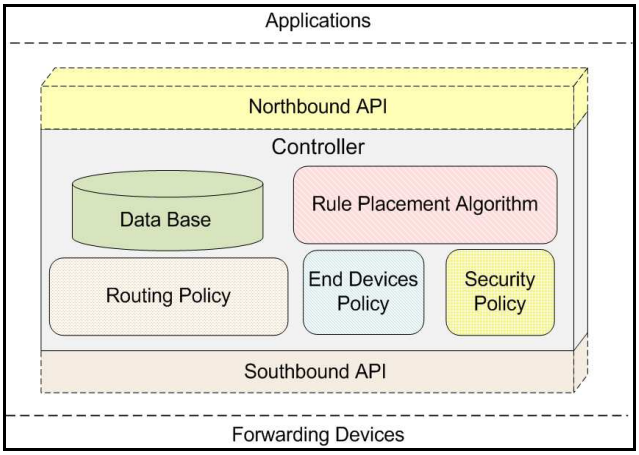

Fig 5. *Block diagram of the controller.*

As mentioned above the controller plays a vital role in the OpenFlow network; therefore multi controllers could establish communication with a forwarding device (switch) provided that only one of them has the master role upon the switch and the others should be in the slave role. Having multiple controllers improves reliability, as the switch can continue to operate in OpenFlow mode if one controller or controller connection fails. The hand-over between controllers is entirely managed by the controllers themselves, which enable fast recovery from failure and controllers load balancing [12]. Many software implementations of the controller are summarized in Table 3.

5.1. NOX

NOX controller was the original controller of OpenFlow. It is written in C++ language and its first version provided an API for Python scripts, but last version of NOX has dropped this API and supported C++ only. NOX provides a high-level programmable interface upon forwarding devices and applications. It is designed to support both small networks of a few hosts and large enterprise networks of hundreds of switches and hosts.

NOX's core has features of fast, asynchronous I/O, topology discovery, host tracking possibility, and learning switch feature [20]. NOX combined with Mininet provides a platform for academic research in networking [21]. It supports now many features of OpenFlow protocol specification 1.3, but the researchers when implement this version discovered the Iperf command which determine the bandwidth utilization does not work properly.

5.2. POX

POX controller is another SDN control platform and it is considered an active development tool. POX was derived from NOX controller platform with the main difference is using Python programming language instead of C++ platform. POX uses Python API (version 2.7) to support network virtualization, SDN debugging, and different application such as layer-2 switch, bridge, hub, etc [22]. NOX and POX controllers support the same GUI and visualization tools to setup, configure controllers, and flow tables. POX still does not support OpenFlow 1.3, which many other controllers support now.

5.3. Floodlight

Floodlight is a very popular SDN controller. It is a contribution from Big Switch Networks and it uses Java based platform (API) thus it runs within a Java Virtual Machine (JVM) and it is considered suitable with continuous increase in number of network devices (switches) that deal with OpenFlow concept [11,23].

Floodlight controller realizes a set of common functionalities to control and inquire an OpenFlow network. The controller has features of simple to extend and enhance, easy to setup with minimal dependencies, support for Open Stack Quantum cloud, topology management, and it deals with mixed OpenFlow and non-OpenFlow network. Floodlight supports applications that include a learning switch, hub application, firewall, and static flow push applications [21]. Floodlight as POX does not support OpenFlow 1.3.

5.4. OpenDaylight

OpenDaylight is an OpenFlow controller. It has open and reference framework for programmability and control through open source SDN, it uses JVM so it can be used with any platform or operating system that supports Java 1.7+. It is a modular, extensible, scalable and multi-protocol controller infrastructure built for SDN deployment on modern heterogeneous multi-vendor networks [21, 24].

OpenDaylight enables users to reduce operational complexity, extend the lifetime of their testing network infrastructure, and enable new services and capabilities. In our test of OpenDaylight, it proved to have an excellent GUI, but Iperf command undergo the same problems that we faced with NOX when dealing with controller.

Table 3. *Controller software implementations.*

Name	Vendor	Progra-mming language	OpenFlow versions	GUI	Operating system
NOX	Nicira	C ++	1.0, 1.3	NOX GUI	Linux
POX	Nicira	Python	1.0	NOX GUI	Linux, Windows, Mac
Floodlight	Big Switch Networks	Java	1.0	Flood-light web UI, Avior	Linux, Mac
OpenDaylight	Linux Foundation Collaborative Project	Java	1.0, 1.3	Open-Daylight web UI	Linux, Windows
Ryu	Nippon Telegraph and Telephone Corporation	Python	1.0, 1.2, 1.3 and Nicira extension	VND	Linux
Mul	kulcloud	C	1.0, 1.3.1	VND	Linux
Beacon	Stanford University	Java	1.0	VND	Windows, Linux, OSX

5.5. Ryu

Ryu is a component-based, open source framework implemented entirely in Python. Nevertheless, the Ryu messaging service does support components developed in other languages [25].

The goal of Ryu is to develop an operating system for SDN that has high quality enough for use in large networks.

Ryu controller includes event management, in-memory state management, application management, and series of reusable libraries (e.g NetCOONF library, sFlow/NetFlow library and OF-Config library). Additionally, it supports applications such as OpenStack Quantum, layer-2 switch, Generic Routing Encapsulation tunnel interface (GRE), and tunnel abstractions. As well, as services about topology and statistics [11].

5.6. Mul

Mul is an OpenFlow SDN controller and it uses C based multi-threaded infrastructure at its core and it is designed to provide good services and ensure reliability through the network [26]. Mul supports OpenFlow 1.3.1and did not work in our test with OpenFlow 1.3 switches such as Open vSwitch.

5.7. Beacon

Beacon is an OpenFlow SDN controller and it uses Java based API. Beacon has features of rapid development, fast and dynamic performance in order to code bundle features [27].

5.8. Special Purpose Controllers

There is a type of controllers; that operates with general purpose controllers such as FlowVisor, and RouteFlow [21]. FlowVisor acts as a proxy between an OpenFlow switch and multi controllers. So that it directs the first packet of a new flow to the appropriate controller according to application, port, MAC, or IP address. This would results in the separation of the network or applications into slices where each slice is controlled by a different controller [28]. It does not support OpenFlow 1.3 yet.

RouteFlow can be considered as a network application on top of general OpenFlow controllers. The major objective of RouteFlow is to build up an open source framework for virtual IP routing solution over product hardware implementing the OpenFlow API [29].

6. Switch Software

OpenFlow switch is an important component of software defined network, switch connects with controller and when a packet arrives to the switch; the switch performs a number of processes, compares the packet header with flow entries, and identifies the actions to be implemented as illustrated in prior sections. Mininet can support different type of switches such as:

6.1. Open vSwitch (OVS)

Open vSwitch is a production quality open source software switch designed to be used as a virtual switch in large scale virtualized environments. Open vSwitch supports many flavors of Linux operating systems such as Debian, Ubuntu, and Fedora. Furthermore, it supports Windows and FreeBSD operating systems [30].

Open vSwitch uses OpenFlow protocol to support the efficient management, virtual switch configuration, and QoS policies need to be applied across a large number of hosts. Open vSwitch supports OpenFlow versions 1.0, 1.1, 1.2, 1.3. As well, it supports other standard management protocols such as SNMP or NETCONF. Additionally, Open vSwitch provides interfaces to monitoring protocols such as sFlow and NetFlow [31]. Open vSwitch is commonly used with Mininet emulator for testing networks that use OpenFlow protocol [21].

6.2. OFSoftSwitch13

OFSoftSwitch13 is an OpenFlow 1.3 compatible user-space software switch implementation. This project is

supported by Evicsson Innovation center/Brazil [21].

Mininet users can install the switch software, NOX controller that supports OpenFlow version 1.3, and download useful documentation to run and configure OFSoftSwitch13 from public Github web site [32].

6.3. LINC

LINC is an open source project that supports OpenFlow protocol versions 1.2, and 1.3. LINC is architected to use generally available commodity, x86 hardware and runs in various operating systems such as Linux, Windows, Mac, etc [21]. Mininet user can install this switch software from Github web site [33].

6.4. Indigo Virtual Switch (IVS)

Indigo project is an open source project, which supports OpenFlow protocol on physical and hypervisor switches. It is designed for high performance and minimal administration and it uses the hardware feature of Application Specific Integrated Circuit (ASICs) of Ethernet switch to run OpenFlow at line speed [21].

Indigo Virtual Switch is a lightweight high performance virtual switch support OpenFlow version 1.0 only. It is designed to enable virtualization in big networks applications as it is used with floodlight controller [34].

7. Tools

Mininet emulator can be integrated with a number of open source tools to meet and implement the different needs of Mininet users, such as: editors, GUI, and benchmarks, ..etc.

7.1. Editors

Mininet user can use one of the Integrated Development Environment (IDE) supported by Mininet environment such as Python IDLE version 2.7, Python IDLE version 3.2, GNU Emacs editor, and Nano editor as a text editor for writing code to build and configure the network topology.

7.2. Graphic User Interface (GUI)

There are a number of GUIs that are used to configure network elements (controller, switches, and hosts) and display network topology. They include many component such as:

7.2.1. Miniedit

Miniedit is a simple Python script presented with Mininet examples. It is used as GUI to construct network topology and emulate it.

Miniedit was developed to add new features and capabilities for the purpose of forming a networks, such as the use of the remote controller and multi controllers, select properties of the links, controller, switches, and hosts, provide command line interface terminals for each node, use monitoring protocols (sFlow, NetFlow), and export python script for network topology [15].

7.2.2. Visual Network Description (VND)

Visual Network Description-SDN version is an online GUI used to form network topology and configure node properties, link type and properties, setup switches flow table entries, and export network topology and its configuration as a Python script to Mininet emulator and OpenFlow controllers or as a C++ script to ns-3 simulator [16].

7.2.3. Avior

Avior is a GUI used with floodlight controller. It provides features of eliminating dependency on using Python script and API in order to manipulate network and monitoring its behavior [35].

Avior has flow manager tools and could give a summery about controllers, switches, and hosts. Controllers summery provides information about host names, JVM memory bloat and other controllers information. Switches summery provides information about port, counters, match header fields, and switch flow entries (add/ delete). On the other hand, host summery provides information about the attached switch Data Path ID (DPID) and the switch port connect to it [21].

7.2.4. Web User Interface (UI)

It is one of the GUI used with some controllers such as Floodlight, and OpenDaylight. It is an online GUI; where user can access it after installing and running the controller using the URL address (http://localhost:8080). Web UI displays topology of network run in Mininet, network node (switches, hosts) information such as IP, MAC, and DPID, flows outline and add/remove switches flow tables entries,..etc.

7.3. Benchmarks

In order to test network performance, many benchmarks could be used. The following are examples for benchmarks.

7.3.1. OFtest

It is a framework and collection of testes for validating OpenFlow switches. OFtest provides a connection as a controller to the OpenFlow switch and send messages to test OpenFlow basic functionalities. It supports OpenFlow specification versions (1.0-1.3) [21].

OFtest uses Python and Scapy as a pre-requisites, where Scapy is a powerful interactive packet manipulation program; used to decode packets, match requests with replies. It also can handle tasks like scanning, and trace routing [36].

7.3.2. OFlops

It is an OpenFlow testing platform used to focus on OpenFlow protocol behavior by implementing basic measurement tests that allow developers to specify and study the capabilities of OpenFlow devices [37].

OFlops tests are used to assess performance of OpenFlow switches in network by utilizing multi-threading parallelism [21]. OFlops has features of modularity, low overhead with minimum delay in processing to support parallelism, and heterogeneity by being compatible with a number of packet

generation and capturing tools such as Cbench and Wireshark.

7.3.3. Cbench

It is a program for testing OpenFlow controllers by generating packet-in messages and waits for flow-mods messages to receive. Cbench has two emulated modes: latency mode and throughput mode. Cbench can be used to measure controller performance by changing its arguments such as number of switches, number of MACs per switch (hosts), number of tests and time of test [7]. Cbench supports OpenFlow 1.0 only, but the Mul controller vender Kulcloud introduced a modified version of Cbench that supports OpenFlow 1.3 and is called Kcbench, albeit, it worked with Mul controller only in our test.

7.4. Linux Kernel Programs

Because Mininet emulator uses Linux kernels, it supports a number of Linux programs and commands such as Dump, Ping, Pingall, Iperf, and plot programs like Gnuplot program; which supports many types of plots in 2D and 3D. The Dump command illustrates network nodes with their interfaces connections. Ping and Pingall test network connectivity and latency. Iperf determines bandwidth utilization and retransmission of packets in TCP applications. It also measures loss and jitter for UDP applications.

7.5. Frenetic

Frenetic is a domain-specific language used to program software defined networks [38]. It has features of high-level abstractions. Therefore, it is useful to replace the low-level interfaces available today. Frenetic offers a suite of information about network state, identity, forwarding policies, and updating policies [39].

7.6. Wireshark

Mininet supports Wireshark packet analyzer and uses it to capture packets traverse the network nodes and analyze these packets to study performance of the network and obtain statistical measurements about its behavior [40]. OpenFlow messages could be displayed and studied using Wireshark. Wireshark version 1.11 and above supports a new filter for OpenFlow 1.3 packets.

8. Conclusions and Future Works

Many SDN protocols are available now, but employing the OpenFlow protocol is highly recommended due to its open source nature, rapid development, and wide deployment.

The proper use of emulation software components in developing OpenFlow and SDN projects would save a lot of time and money compared to practical testbeds since real hardware devices are still expensive and support primitive versions of OpenFlow standard only.

In this paper, we examined many software components related to OpenFlow protocol. Most of them were downloaded, installed, and operated successfully. OpenDaylight, Floodlight, and OFSoftSwitch13 proved to have good properties like good documentation and flexibility.

Observing the rapid development of OpenFlow standards predicts that a major breakthrough is expected in version 2.0, but for the time being the use of software components that supports version 1.3 like NOX, OpenDaylight, and Mul is recommended since no software component supports the new 1.4 version yet.

Most of the tested software components are standalone components. The need for a frame that gather the installation and operation of switches and controllers into a single platform with a certain GUI and benchmark would facilitate the development of OpenFlow projects. EstiNet is a good example of such a platform. An emulation projects to test the compatibility of OpenFlow protocol with WLAN and IPv6 deployment is under consideration by the researchers.

References

[1] M. Mendonca, B. Nunes, K. Obraczka, and T. Turletti, "Software defined networking for heterogeneous networks," IEEE COMSOC MMTC E-Letter United States, vol. 8, no. 3, pp. 36-39, May 2013.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart and A. Vahdat, "B4: experience with a globally-deployed software defined WAN," SIGCOMM'13 Hong Kong, China, pp. 3-14, August 2013.

[3] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," IEEE Internet Computing, vol.17, pp. 20 – 27, March-April 2013.

[4] Migration Working Group, "Migration use cases and methods," Open Networking Foundation (ONF), February 2014.

[5] S. Wang, C. Chou, and C.Yang, " EstiNet OpenFlow network simulator and emulator," IEEE Communications Magazine, vol. 51, pp. 110-117, September 2013.

[6] B. Lantz, B. Heller, and N. McKeown, "A network in a lap-top: rapid prototyping for software-defined networks," In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks New York,2010.

[7] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers,"CEE-SECR'13 Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, November 2013.

[8] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," IEEE Communications Surveys & Tutorials, in press, January 2014.

[9] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: a survey," IEEE Communications Surveys & Tutorials, vol. 16, pp. 493 – 512, February 2014.

[10] Open Network Foundation (ONF), "Software-defined networking: the new norm for networks," April 2012. Available at: https://www.opennetworking.org/images/stories/downloads/openflow/wp-sdn-newnorm.pdf, accessed on 23/3/2014.

[11] T. Nadeau and K. Gray, SDN: Software Defined Networks, 1st ed., O'Reilly Media, Inc., August 2013.

[12] Open Network Foundation (ONF), "OpenFlow switch specification, version 1.4.0 (wire protocol 0x05)," October 2013. Available at:

[13] https://www.opennetworking.org/images/stories/downloads/sdnrsources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf, accessed on 23/3/2014.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, March 2008.

[15] Open Networking Foundation (ONF),"Open networking foundation," Available at https://www.opennetworking.org/images/stories/downloads/about/onf-what-why.pdf.

[16] Tech and Trains, available at: http://gregorygee.wordpress.com/category/miniedit/.

[17] Visual Network Description (VND), available at: http://www.ramonfontes.com/visual-network-description/.

[18] EstiNet Technologies Inc., "The GUI user Manual for the EstiNet 8.0 Network Simulator and Emulator", January 2013.

[19] T. Dietz, "Trema tutorial," NEC Corporation, March 2012. Available at: http://www.fp7-ofelia.eu/assets/Uploads/201203xx-TremaTutorial.pdf, accessed on 23/3/2014.

[20] Z. Cai, A. Cox, T. Eugene Ng, "Maestro: balancing fairness, latency and throughput in the OpenFlow control plane," Rice University Technical Report TR11-07, 2011.

[21] NOX, http://www.noxrepo.org/nox/about-nox/.

[22] S. Azodolmolky, Software Defined Networking with OpenFlow, Packt Publishing, 1st ed., October 2013.

[23] POX, http://www.noxrepo.org/pox/about-pox/.

[24] Floodlight, http://www.projectfloodlight.org/floodlight/.

[25] OpenDaylight, http://www.opendaylight.org/software/.

[26] Ryu, http://osrg.github.io/ryu/.

[27] Mul, http://sourceforge.net/projects/mul/.

[28] Beacon, https://openflow.stanford.edu/display/Beacon/Home.

[29] FlowvisorExecise, https://github.com/onstutorial/onstutorial/wiki/Flowvisor-Exercise.

[30] RouteFlow, https://sites.google.com/site/routeflow/.

[31] Open vSwitch, http://openvswitch.org/.

[32] A. Clemm, and R. Wolter, "network-embedded management and applications understanding programmable networking infrastructure," Springer New York, 2013.

[33] OpenFlow 1.3 Tutorial, available at: https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3Tutorial.

[34] LINC, https://github.com/FlowForwarding/LINC-Switch.

[35] Indigo, https://github.com/floodlight/indigo.

[36] Avior, https://github.com/Sovietaced/Avior.

[37] Scapy, http://www.secdev.org/projects/scapy/.

[38] OFLops: user manual, available at: http://archive.openflow.org/wk/images/3/3e/Manual.pdf.

[39] Frenetic, http://www.frenetic-lang.org/overview.php.

[40] N. Foster, M. Freedman, A. Guha, R. Harrison, N. Katta, C. Monsanto, J. Reich, M. Reitblatt, J. Rexford, C. Schlesinger, A. Story, D. Walker, "Languages for software-defined networks," IEEE Communications Magazine, vol. 51, pp. 128 – 134, February 2013.

[41] R. Shimonski, Analyzing and Troubleshooting Network Traffic, Elsevier Inc., 1st ed., 2013.