Emulating Software Defined Network Using Mininet and OpenDaylight Controller Hosted on Amazon Web Services Cloud Platform to Demonstrate a Realistic Programmable Network.

Lindinkosi L. Zulu
College of Science, Engineering and Technology
University of South Africa
Florida, Johannesburg, South Africa
61366935@mylife.unisa.ac.za

Kingsley A. Ogudo
Department of Electrical and Electronic Engineering Technology
University of Johannesburg
Johannesburg, South Africa
kingsleyo@uj.ac.za

Patrice O. Umenne
Department of Electrical and Mining Engineering
University of South Africa
Florida, Johannesburg, South Africa
umennpo@unisa.ac.za

***Abstract*— In this paper, a Software Defined Network was created in Mininet using python script. An external interface was added in the form of an OpenDaylight controller to enable communication with the network outside of Mininet. The OpenDaylight controller was hosted on the Amazon Web Services elastic computing node. This controller is used as a control plane device for the switch within Mininet. The OpenDaylight controller was able to create the flows to facilitate communication between the hosts in Mininet and the webserver in the real-life network. In order to test the network, a real life network in the form of a webserver hosted on the Emulated Virtual Environment – Next Generation (EVE-NG) software was connected to Mininet.**

Keywords—SDN; Mininet; OpenDaylight; Amazon AWS, Cloud networking and Cloud computing

I. INTRODUCTION

Traditional networking is currently very decentralized with each network device having its own control plane. It requires individual manual configuration of each device on the network if there are changes to be implemented. The hardware and the software of the traditional networking architecture are proprietary and specifically designed to work together. This current setup makes it difficult for the network to be flexible and scalable to meet the high demand of modern applications and requirements. Software Defined networking (SDN) was developed to address these challenges the current network model is failing to address and OpenFlow was developed as the first standard communications interface defined between the control and forwarding planes of an SDN architecture [1].

Open Networking Foundation (ONF) defines Software Defined Networking as the physical separation of the network control plane from the forwarding plane, whereby the control plane controls several devices externally. It is an architecture that decouples the network control and forwarding functions. This allows the network to be dynamic, adaptable, cost-effective, software programable and easily manageable [2]. The advent of cloud computing allows the control plane to be logically centralized and distributed in cloud platforms.

This paper begins by looking at the literature covering Software Defined Networking, Mininet, OpenDaylight controller and cloud networking. The methodology section follows which describes the network used and its components. On the results section, the test results such as throughput are documented. The conclusion summarizes key points and is followed by the references.

II. BACKGROUND STUDY

The control plane is the centrally located control unit called SDN controller acting as the Network Operating Systems (NOS). The data plane resides inside the network core devices and is only responsible for forwarding data packets controlled by the central SDN controller. These separated planes use protocols and an Application Programmable Interface (API) to communicate [3].

OpenFlow is one of the protocols used by Software defined networks and was started by Stanford University in 2008 [4]. Different companies came together in 2011 and formed Open Networking Foundation (ONF) to further develop OpenFlow and Software Defined Networking [5]. With the separation of the control and data planes, the data plane only performs the data packet forwarding action and it resides in the network device. The control plane is logically positioned on top of the data plane and acts as the brains of the network [6]

Software Defined networks makes it possible to consolidate in one place complex software used to configure and control several devices making the process less expensive. A centralized controller gives a benefit of having a view of the

978-1-5386-6477-3/18/$31.00 ©2018 IEEE

network, which then enables it to make decisions on how data planes must move the traffic [7-10].

SDN makes it possible to dynamically provision the network. It improves network resources utilization and simplifies traffic engineering [11]. It makes it possible to use external applications to program the network. Communication between the devices in SDN uses open interfaces making it to be vendor neutral [12]. To test Software Defined Networks, an emulator called Mininet is amongst the popularly used tools [13].

A. Mininet

Mininet is the container-based emulator [14]. It allows the running of unmodified code interactively on virtual hardware on a regular computer. It provides convenience and realism at low cost compared to running on a hardware. Programs run on emulators require none or minimum modification when applied to real live networks [15]. Mininet runs unmodified code of network applications in lightweight Linux containers to achieve its scalability and accuracy.

Mininet supports OpenFlow-based Software-defined Networking (SDN). It provides a flexible and cost-efficient experimental platform to develop, test, and evaluate OpenFlow applications. In Mininet, the processes of the virtual hosts and their application processes run inside the container. This allows them to have an independent view of system resources but still share the kernel with other containers [16].

Mininet supports five built-in network topologies. These built-in topologies are Minimal, Single, Linear, Tree and Reversed. Network topologies in Mininet can be modified using the command-line interface (CLI) [17].

The study by Keti and Askar in [18] highlights Mininet's characteristics as being flexible, applicability, interactivity, scalability, realistic and share-able. This is because in Mininet, new topologies and new features can be set in software using programming languages and common operating systems. Networks emulated in Mininet are usable with real life networks based on hardware without the need to make changes in source codes. To manage and run the simulated network in Mininet occurs in real time as it happens in a real-life network. Mininet can be scaled to large networks with hundreds or thousands of nodes. Networks implemented on Mininet can be easily shared as it is share-able [19].

Software Defined Networking switches, hosts, controllers and links can be created by typing commands through Mininet's command line interface. The command line interface (CLI) in Mininet supports most Linux commands. The most commonly used commands are-: nodes: which lists all created nodes, dump: which displays the information about the network and created nodes, net: which shows how network elements are connected to each other. The CLI also support the day-to-day troubleshooting commands used in computer networking. These commands include "pingall", which output the results of the connectivity test among all nodes. "Ifconfig" is also supported which displays the internet protocol (IP) information of the node. "Iperf" is also supported which is a tool used to test network performance. It uses a client/server model, where traffic is initiated from the client and traverses the network to the server. Iperf creates data test streams supported by the network with a time-stamp and report the amount of data transferred and the throughput measured. Iperf supports two types of transport protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Many applications like File Transfer Program (FTP), Simple Mail Transfer Protocol (SMTP) and Hypertext Transfer Protocol (HTTP) use TCP as the transport protocol. Using TCP mode, Iperf tests the maximum TCP bandwidth at the transport layer. In UDP mode, Iperf tests the jitter, packet loss and bandwidth. UDP mode is ideally for testing quality of service for applications like voice and video streaming.

To see the list of all available commands, one can use help command, which is also supported in Mininet [20].

Mininet Python API can also be used to create custom network topologies [21]. Python is an interpreted, interactive, object-oriented programming language. It provides high-level data structures. Python is modular by nature. The kernel is very small and can be extended by importing extension modules. A python program is compiled automatically by the interpreter and can be installed in any computer running any operating system [22].

B. OpenDaylight Controller

OpenDaylight (ODL) controller is the Software Defined Networking controller used in this project. It is based on Services Abstraction Layer (SAL), which allows it to support other protocols and not only OpenFlow. It is implemented in Java and can be deployed in any system supporting Java. OpenDaylight controller was developed by the OpenDaylight consortium in 2013. OpenDaylight project is supported by Cisco, Juniper, VMWare and many other vendors and companies operating in the networking environment. This support by many organizations enables OpenDaylight to be vendor neutral [23]

Fig 1 shows the OpenDaylight controller architecture. The controller uses Application Programable Interfaces (API) like Representational State Transfer (REST) technology to communicate with the Network Applications orchestrations and services layer. This can include OpenStack Neutron, Virtual Tenants Network (VTN) coordinator [24-25]. The controller layer itself run several services which includes service abstraction layer (SAL), OpenStack service, base network service and many more. To communicate with data plane elements, the controller uses southbound interfaces and protocol plugins such as OpenFlow, the Open vSwitch Database Management Protocol (OVSDB) [26], the Network Configuration Protocol (NETCONF) [27] and many more [28].

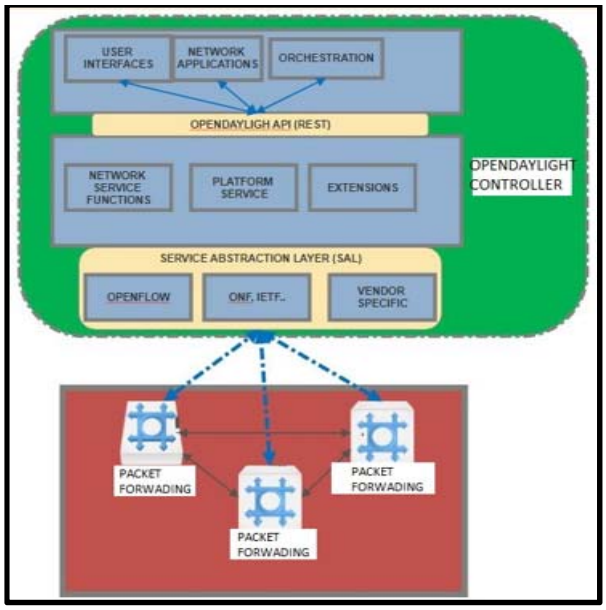

Fig. 1 OpenDaylight controller architecture

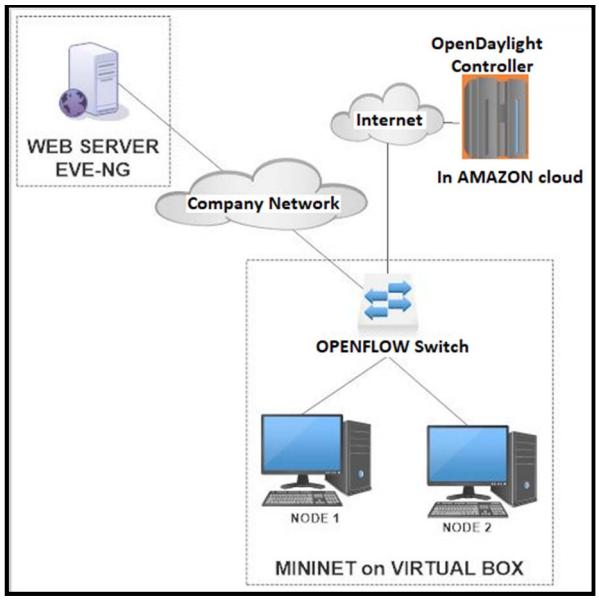

Fig. 2. Logical network used on this paper

C. *Cloud Computing*

The centralization of control plain and other SDN features has made cloud computing to be the essential part for most companies. It is one of the fastest emerging business for Internet Service Providers (ISP) [29]. A cloud can be described as a large-scale environment that can consists of many physical hosts and virtual machines (VMs). Each host in the cloud environment can serve multiple virtual machines. Services on the cloud network are provided on-demand bases. Virtual machines in a physical host can be dynamically provisioned as per the need. They can be added, removed or even migrated dynamically [30].

Amazon Web Services (AWS) is the leading cloud provider that offers computing, storage, and content delivery platforms. Amazon cloud services includes Elastic Compute Cloud (EC2) [31] and Simple Storage Service (S3), with "CloudFront", the Content Delivery Network (CDN). Amazon through AWS offers a large set of computing resources, such as storing and processing where capacities can be split, assigned dynamically as per customer's needs. Companies like Netflix and Dropbox are among the companies that uses Amazon Web Services (AWS) [32].

III. Methodology (Network Design)

The network used in this paper consist of a web server, OpenDaylight controller and Software Defined Network emulated on Mininet as seen in Fig 2.

A. *Web Server*

The web server uses Ubuntu 17 as the operating system. We have configured Apache 2, which enabled us to host a simple html page as the website. This webserver is hosted on the Emulated Virtual Environment – Next Generation (EVE-NG) software. EVE-NG is the Emulated Virtual Environment for networking. It provides tools to be able to model a real-life network as virtual devices and interconnect them with other virtual or physical devices.

B. *OpenDaylight Controller*

In this project we used the eighth release of the OpenDaylight controller, which is called Oxygen. We downloaded the software from the OpenDaylight software download page and installed the controller on the Ubuntu 17 server. To install and enable required features that the OpenDaylight controller must use, an open source application called Apache Karaf is used. Karaf as it is normally called is a modular Open Services Gateway Initiative (OSGI) that provides tools and features required to deploy an application. An Open Services Gateway Initiative (OSGI) is a set of specifications for developing and deploying modular software programs and libraries, which are packed in bundles. Karaf enables modules to be installed, started, stopped, updated, and uninstalled without requiring a reboot.

By default, the OpenDaylight controller has no features enabled. We have installed and enabled the following features in this project on the controller (but there are many features, which can be installed and enabled)-:

- odl-restconf – Representational State (REST) like protocol that provides a programmatic interface over Hyper Text Transfer Protocol (HTTP) for accessing data on port 8080 for HTTP requests.
- odl-l2switch-all – Layer2 switch functionality.
- odl-mdsal-apidocs - Model Driven Service Abstraction Layer (MD-SAL) Application Programmable Interface (API) Documentation.
- odl-dlux-all - Graphical user interface for OpenDaylight based on the AngularJS Framework.

The OpenDaylight controller used on this paper is hosted on Amazon Web Services (AWS) cloud platform. We have used the Elastic Compute Cloud (EC2), which is a secure and resizable compute node. It allowed us to obtain and configure capacity in minutes.

The Elastic Compute Cloud (EC2) can scale both up and down allowing us to increase or decrease capacity as per our need. Wireshark, the network packet analyzer was used to analyze communication between the Open Virtual Switch in Mininet and the OpenDaylight controller.

When the communication between the switch and the controller is established. The controller adds flows to enable the switch to behave like a learning switch. When the switch receives a packet, it starts by performing a table lookup in the first flow table called table 0. In pipeline, each flow table contains one or more flow entries. Matching starts with the first flow table. If a Match is found, instructions associated with flow entry are executed. Instruction may direct the packet to next flow table in pipeline. When processing stops, the associated action set is applied, and packet forwarded. Instructions describe packet forwarding, packet modification, group table processing and pipeline processing. The summary flow chart is shown in Fig 3.

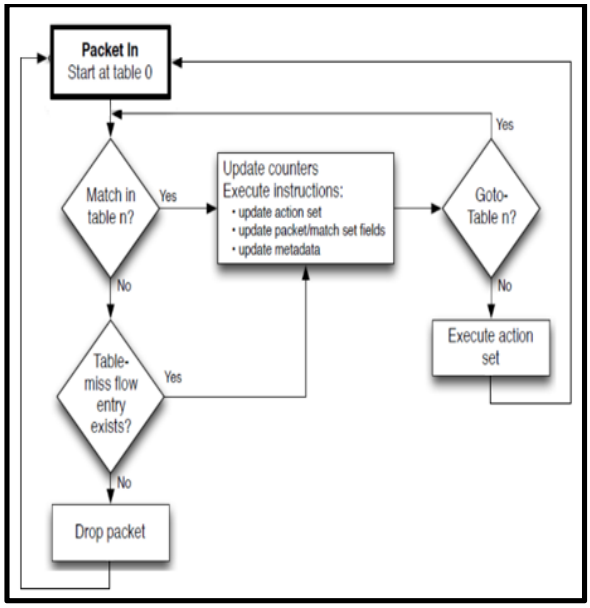

Fig. 3. Open Virtual Switch packet processing flow chart

C. Mininet

The Mininet Virtual Machine (VM) used in this paper is hosted on Oracles' Virtual Box. Mininet was installed on Ubuntu 17 operating system. The emulated Software Defined Network in Mininet was created using a Python script. The three main functions used in this script are Topo, Switch and Controller. Topo: This is the base class for Mininet topologies. It creates Data center network representation for structured multi-trees.

A function, which is used to create a custom network was created using Python. For this function to create the network, Mininet was prevented from creating the network using the default values. This was achieved by setting the topo class to none and the build class to false. Using the controller class, a remote controller was defined and given values for the name and an IP address, which in this case is the IP address of the OpenDaylight controller hosted on Amazon cloud. The connection port was set to port 6633.

Using OVSKernelSwitch sub class, an Open Virtual Switch (OVS) was created. Two (2) network hosts were also defined and given networking properties. The script defines the network subnet that the controller must use together with the links between the switch and the hosts. As part of the program, the script programs the controller to add the external interface to the switch after creating the network. This interface is used by Mininet to reach the Linux server inside the company domain.

To start the program, we loaded the saved python script from the directory that it was saved on. Mininet created the network as defined by the script. The created network consists of two (2) hosts and the Open Virtual Switch (OVS). The created switch has a control channel, which it uses to communicate with the controller. It has the pipeline, which consists of flow tables. There is also data path, which is the forwarding plane as seen in Fig 4.

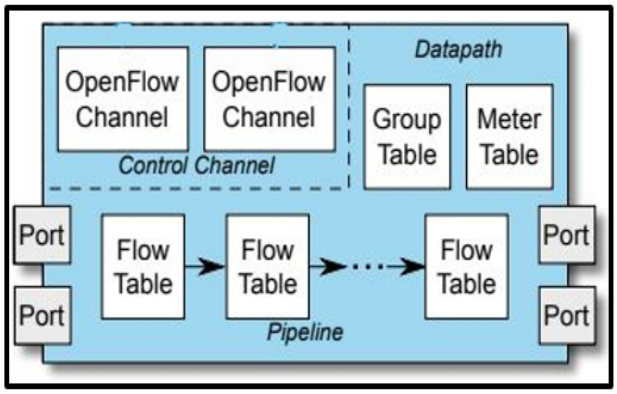

Fig. 4. Open Virtual Switch

D. Application Program Interface (API)

An application program interface (API) is a set of routines, protocols, and tools for building software applications. It specifies how software components should interact. They are used when programming graphical user interface (GUI) components. API makes it easier to develop a program by providing all the building blocks. One of the commonly used API in the field of networking is REST API.

Representational State Transfer (REST) API is an architectural style and an approach for communication used in the development of Web Services. It enables users to connect and interact with cloud services efficiently. To test the API, the program called Postman can be used. Postman is an application for testing APIs by sending request to the web server and getting the response back. Postman makes it easy to test, develop and document APIs. It allows users to set up all the headers and cookies the API expects and checks the response. "Postman" was used to send RESTCONF GET API to retrieve node inventory and topology as created by Mininet and seen by OpenDaylight controller.

RESTCONF is an Internet Engineering Task Force (IETF) draft that describes how to map a YANG specification to a RESTful interface. The REST-like API provide an additional simplified interface that follows REST-like principles and is compatible with a resource-oriented device abstraction. RESTCONF uses HTTP methods to provide CRUD (Create, read, update and delete) operations on a conceptual datastore containing YANG-defined data, which is compatible with a server that implements NETCONF datastores.

IV. RESULTS

"Iperf" is a tool used to test the maximum bandwidth that can be achieved between two (2) network devices. "Iperf" sends test data between the defined network devices and measures the throughput, bitrate, loss and other parameters. To test the functionality of the created network, an TCP "Iperf" test between the host and the Linux server was performed.

Using Wireshark, communication between the switch and the controller was captured. In the beginning of the communication, OpenFlow Channel messages between the switch and the controller are observed. The OpenDaylight requested the identity and basic capabilities of the switch. The switch responded with the requested information as seen with the OFPT_HELLO, OFPT_FEATURES_REQUEST and OFPT_FEATURES_REPLY packets as seen in Fig 5.

Source	Destination	Protocol	Lengt	Info
10.10.204.37	52.15.83.11	OpenFlow	74	Type: OFPT_HELLO
52.15.83.11	10.10.204.37	OpenFlow	90	Type: OFPT_FEATURES_REQUEST
10.10.204.37	52.15.83.11	OpenFlow	98	Type: OFPT_FEATURES_REPLY
52.15.83.11	10.10.204.37	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
10.10.204.37	52.15.83.11	OpenFlow	74	Type: OFPT_BARRIER_REPLY
52.15.83.11	10.10.204.37	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
10.10.204.37	52.15.83.11	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
52.15.83.11	10.10.204.37	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
10.10.204.37	52.15.83.11	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_FEATURES
10.10.204.37	52.15.83.11	OpenFlow	122	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP_FEATURES
10.10.204.37	52.15.83.11	OpenFlow	338	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC

Fig. 5. OpenFlow channel messages

Once the flows were added, the TCP "Iperf" test between host and the Linux server was successful. With the server using default TCP window size of 85.3 KBytes and the host using the default window size of 391 KByte, we were able to transfer 896 MBytes at a rate of 751 Mbits/sec from host to Linux server. From Linux server to host, 1.32 GBytes was transferred at a rate of 1.13 Gbits/sec as seen in the TCP "Iperf" results in Fig 6.

```
mininet-wifi> h1 iperf -c 10.1.10.2 -d
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.1.10.2, TCP port 5001
TCP window size:  391 KByte (default)
------------------------------------------------------------
[  3] local 10.1.10.31 port 53638 connected with 10.1.10.2 port 5001
[  5] local 10.1.10.31 port 5001 connected with 10.1.10.2 port 51612
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   896 MBytes   751 Mbits/sec
[  5]  0.0-10.0 sec  1.32 GBytes  1.13 Gbits/sec
mininet-wifi>
```

Fig. 6 Iperf test results

Another way of representing the Iperf test results using Wireshark is seen in Fig 7. Fig 7 shows the average throughput that will be the maximum bandwidth during the 90 ms period of the "Iperf" test for the uplink and downlink connections.

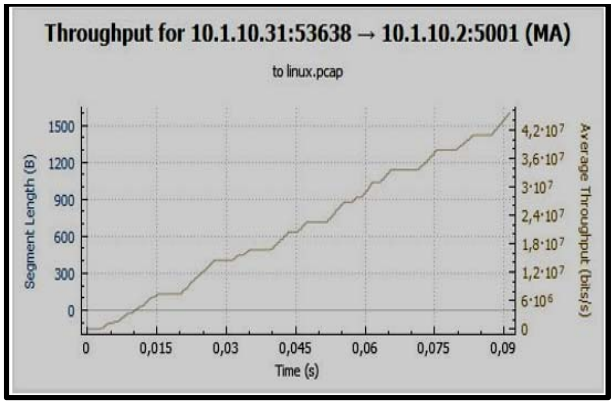

Fig. 7 Average throughput over 90ms

"Postman" was used to verify that indeed the created network in Fig. 8 is controlled by the OpenDaylight controller.

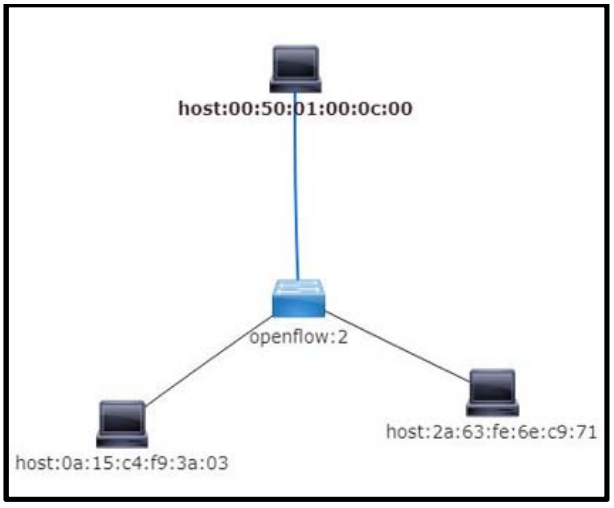

Fig. 8. ODL-DLUX network topology

To ensure reliability of the communication between the switch and the OpenDaylight controller, Transmission Control Protocol was used as seen in Fig. 9. The handshaking commands in the figure ensure reliability of the data transferred.

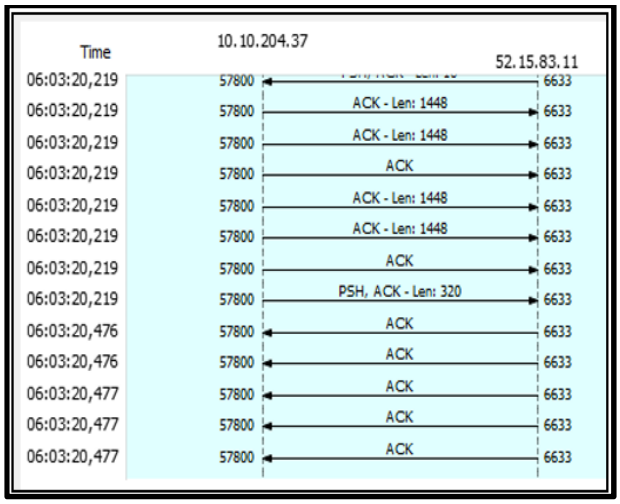

Fig. 9. TCP communication

V. CONCLUSION

In conclusion, Mininet was used to design a Software Defined Network. The SDN network was integrated to a real-life network using EVE-NG software via an OpenDaylight controller. OpenFlow protocol was modelled and used to facilitate communication between a virtual switch in Mininet and the OpenDaylight controller hosted in a cloud network. The OpenDaylight controller controls the flow of data from Mininet to the real-life network. The results showed the throughput and bandwidth measured in the communication process.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, & J. Turner. "Openflow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review, 38(2):69–74, 2008.

[2] Open Networking Foundation (ONF). Software defined networking: The new norm for networks. White Paper. From https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. 2012

[3] A. Lara, A. Kolasani, & B. Ramamurthy. "Network innovation using openflow: A survey". IEEE Communications Surveys and Tutorials, Vol 16 No 1 pp 1-20. 2013.

[4] H. Shimonishi, Y. Takamiya, Y. Chiba, K. Sugyo, Y. Hatano, K. Sonoda, K. Suzuki, D Kotani, & I. Akiyoshi. "Programmable network using OpenFlow for network researches and experiments". Proceedings of the Sixth International Conference on Mobile Computing and Ubiquitous Networking (pp.164-171). Okinawa, Japan. 2012

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, & J. Turner, J. "OpenFlow: Enabling innovation in campus networks". ACM SIGCOMM Computer Communication Review, 38(2), 69–74. 2008

[6] A. Lara, A. Kolasani, & B. Ramamurthy. "Network innovation using OpenFlow: A survey". IEEE Communications Surveys and Tutorials, 16(1), 493–512. 2014

[7] C. C. Machado, L. Z. Granville, A. Schaeffer-Filho, & J. A. Wickboldt, "Towards SLA Policy Refinement for QoSManagement in Software-Defined Networking". IEEE 28th International Conference on Advanced Information Networking and Applications. 2014

[8] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan,N. Viljoen, M. Miller, & N. Rao. "Are we ready for sdn? Implementation challenges for software-defined networks". Communications Magazine, IEEE, vol. 51, no. 7, pp. 36–43, 2013.

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

[10] D. Thomas & N. K. Gray. "SDN: Software Defined Networks", Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, First Edition, August 2013.

[11] J. Mambretti, J. Chen & F. Yeh. "Software-Defined Network Exchanges (SDXs): Architecture, Services, Capabilities, and Foundation Technologies". Proceedings of the 2014 26th International Teletraffic Congress (ITC). 2014

[12] B. Astuto, A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, & T Turletti. "A Survey of Software-Defined Networking: Past,Present, and Future of Programmable Networks". hal-00825087, version 5- 19 Jan. 2014.

[13] I. Z. Bholebawa & U. D. Dalal. "Design and Performance Analysis of OpenFlow-Enabled Network Topologies Using Mininet". International Journal of Computer and Communication Engineering. Volume 5, Number 6, November 2016.

[14] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz & N. McKeown. "Reproducible Network Experiments Using Container-Based Emulation". CoNEXT'12, December 10–13, Nice, France. 2012

[15] Y. Huang, V. Jeyakumar, B. Lantz, B. O'Connor, N. Feamster, K. Winstein & A. Siyaraman. "Teaching Computer Networking with Mininet". Wi-Fi: HHonors, SGC. 2014

[16] J. Yan & D. Jin. "VT-Mininet: Virtual-time-enabled Mininet for Scalable and Accurate Software-Define Network Emulation". SOSR2015, Santa Clara, CA, USA. June 17–18, 2015.

[17] R. Kharga, P. Bholebawa, I. Satyarthi, S. Gupta, & S. Kumari. "OpenFlow technology: Ajourney of simulation tools". International Journal of Computer Network and Information Security, 6(11), 49–55. 2014

[18] F. Keti & S. Askar. "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments". 6th International Conference on Intelligent Systems, Modelling and Simulation, Kuala Lumpur, 2015, pp. 205-210. 2016

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

[20] K. K. Sharma & M. Sood. "Mininet as a Container Based Emulator for Software Defined Networks". International Journal of Advanced Research in Computer Science and Software Engineering. Volume 4, Issue 12, December 2014

[21] Python at https://www.python.org/

[22] M. Sanner. Python: "A Programming Language for Software Integration and Development". Article in Journal of Molecular Graphics and Modelling · November 1998

[23] OpenDaylight Consortium. http://www.opendaylight.org.

[24] W. Zhou, L. Li, M. Luo & W. Chou. "REST API Design Patterns for SDN Northbound API". 28th International Conference on Advanced Information Networking and Applications Workshops, Victoria, BC, 2014, pp. 358-365. 2014

[25] L. Li, W. Chou, W. Zhou & M. Luo. "Design Patterns and Extensibility of REST API for Networking Applications". IEEE Transactions on Network and Service Management, vol. 13, no. 1, pp. 154-167, March 2016.

[26] M. Brandt, R. Khondoker, R. Marx & K. Bayarou. "Security analysis of software defined networking protocols - OpenFlow, OF-Config and OVSDB". Paper presented at Fifth IEEE International Conference on Communications and Electronics, ICCE 2014, July 30 - August 1, 2014

[27] J. Schonwalder, M. Bjorklund & P. Shafer. "Network configuration management using NETCONF and YANG". IEEE Communications Magazine, vol. 48, no. 9, pp. 166-173, Sept. 2010

[28] Z. K. Khattak, M. Awais & A. Iqbal. "Performance evaluation of OpenDaylight SDN controller". 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, 2014, pp. 671-676. 2014

[29] A. Mayoral, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez & J. Vílchez. “Integrated IT and Network Orchestration using OpenStack OpenDaylight and Active Stateful PCE for Intra and Inter Data Center Connectivity”. ECOC 2014.

[30] S. Shin & G. Gu. CloudWatcher: “Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)”. 20th IEEE International Conference on Network Protocols (ICNP), Austin, TX, 2012, pp. 1-6. 2012

[31] E. Walker. “Benchmarking Amazon EC2 for High-Performance Scientic Computing”. USENIX login: Magazine, October 2008.

[32] I. Bermudez, S. Traverso, M. Mellia & M. Munafò, “Exploring the cloud from passive measurements: The Amazon AWS case”. Proceedings IEEE INFOCOM, Turin, 2013, pp. 230-234. 2013