Software Defined Networking Automation Using OpenDaylight and Network Virtualization for security and scalability: a network enterprise case

Aws Jaff*

Pharmacy Department, Knowledge University, Erbil, Iraq

Abstract. Software Defined Network SDN is a technique that centralized the control plane in one device and the data plane (forward plane) in the other devices connected and communicated via SDN protocols, so these devices will get instructions from a central entity called a controller to manage the network efficiently. The controller generally has a global view of the network topologies and can broadcast any management policy to all devices in the network easily. SDN makes the recent network more responsive and flexibly managed by delivering interoperable and programmable network services to network organizations all around the world. In this work, two real critical scenarios were investigated against other scenarios to manifest the SDN ability to deal with them. Finally, results show that the opted implementation features are better than the others in term of testing parameters.

1 Introduction

It is all agreed traditional networks are too expensive to deploy and manage, too complicated to control, and too hard to change or update.[1] Software-Defined Networking (SDN) has emerged as a flexible programmable platform for managing and controlling networks. Thus, the main goal of SDN is to make the SDN network faster and more customized.[2]
In traditional networks, scalability and security are two of the most important obstacles when managing, configuring, maintaining the network. The huge number of appliances over big network enterprises make managing and controlling difficult since the manual configuration is needed to add, remove, update, or even maintain an actual appliance or the whole network or sub-networks [3]. Furthermore, a multi-vendor network requires network expertise to deal with network inconsistency, therefore extensive knowledge is required to achieve heterogeneous devices that complicate the network segmentation. On another hand, the time-consuming and error-prone will occur when deploying stage takes place by the network administrator which will be a highly administrative hassle to configure network standards. SDN overcomes the complexity caused by different network protocols and configuration interfaces that different vendors use for their devices. Another SDN advantage is cost-

* Aws Ahmed Jaff: aws.ahmed@knu.edu.iq

effectiveness since the use of virtualization can serve the SDN acquisition accomplishment of operational cost.[4]
SDN as a definition is the separation of the control plane and data plane of the network elements and devices. SDN is a logically centralized control unit with programable capability via Application Programming Interface (API). The separation of the two planes and their remote communication can be achieved through layered SDN architecture as Open Networking Foundation (ONF) that basically consists of three layers; 1. Device layer: enabled OpenFlow devices, 2. Control Layer: logical central controller, and 3. Application layer: the implementation applications that run on top of the controller.
Recently SDN received a lot of attention to solve the most problems of the traditional networks, the design of the SDN change the evolution of the networking by expressing network algorithms in a way that appropriate abstraction meets its specific application and without changing the underlying of the network infrastructure.[5]
SDN solve many challenges of traditional hardware networks; as simple and shouldn't be expensive, vendor-independency which easy to switch and upgrade among different vendors, and flexible; should be structured in current requirements as isolation, virtualization, and network optimization.[6] SDN also solves the legacy communication protocols in the traditional networking to be one logical centralized control unit that empowers the network automation as multiple logical control units. Finally, SDN gives a higher security level in managing network infrastructures and data centres with the ideal level of scalability for future network expansion. [5]

1.1 Literature review

The SDN implementation of networking is a new trend in today's technology industry, that's why there are few types of research works that are close to the field of our implementation work [7]. Many studies have been aimed to compare SDN architectures with their performance, but now those controllers were the foundation of nowadays controller such as Floodlight and OpenDaylight (ODL) [8]. Many aspects have been considered for building those controllers, based on a set of the requirements and these aspects were evaluated in the following areas: virtualization, Graphical User Interface GUI support, Interfaces, open sources, API, OpenFlow support, productivity, and modularity. Each controller was revealed to build those controllers each possess a critical weakness in their SDN performance [9].
Rationally, many advanced studies of performing widely used SDN-based OpenFlow controllers, doing an impractical analysis of the effectiveness of an average of workload, and maximum throughput on such controllers as Floodlight, and OpenDaylight. The variety has been witnessed in that studies regarding the analysis [10]. The ODL architecture and developing ODL model for network automation in a simple way for better understanding and intends of new controller innovation and accelerate the ODL adaptation for network programming [11]. The heavy relied on the Northbound (NB) and Southbound (SB) architectures, protocols plugins, and APIs communication that support OpenFlow 1.0 and 1.3 used for control, manage and monitoring the network including the orchestration services, Also, ODL supports clustering services where multiple controllers take place as one logical controller. The controller should not be limited with a southbound protocol such as OpenFlow and Northbound APIs by encouraging ODL adaptation ability of NB and SB of cluster redundancy, high availability, and scalability [12]. Some new methods illustrated in configuring and testing network devices by using automation for reducing configuration time, easier to maintain, and increasing the network stability [13].
Virtualization can be a single instance or combination of many such as Operating System (OS), networks, Application servers and storage devices. There are many types of virtualizations, but our concentration will be on three types of them: (i) Hardware, (ii)

Software, and (iii) network virtualization. Furthermore, virtualization provides numerous benefits such as cost-effective, efficient utilization of recourses, and fast accessibility [14].
Since the use of a Linux OS to run and build SDN devices and its infrastructure in this experiment. GNS3 is the main emulator to deal with that approach, GNS3 is an open sources multi-vendors emulator that provides virtualization environment that simulate real network devices. Moreover, the GNS3 market offers OpenvSwitch (OVS) to install in an easy way with full functionality support. OVS allows implementing policies such as permitting or denying flows between devices and users in network security, monitoring, quality of service QoS, and automated control [15].
OpenDaylight is open source SDN controller that supports multiple Southbound Protocols, as well as exposes open northbound APIs. OpenvSwitchs support both traditional and software-defined networking protocols such as OpenFlow (OF). In this topology, the commands have been used to write OpenFlow rules to OpenvSwitches that are running on GNS3.
OFM is an application that is used to visualize the OpenFlow topologies that run on OpenDaylight located on the northbound to manage the OpenFlow communication networks via RESTCONF API. RESTCONF API is an interface that used RESTCONF protocol, those interfaces are generated at the run time of YANG models. YANG models are Network Configuration Protocol (NETCONF) data modelling language used to model configuration and state data manipulated by the Network Configuration Protocol. YANG interface is dynamic generation, so not statically compiled java programming classes. an implementation has made to this interface on separate server than the same ODL server since our virtualized ODL can't handle multitasking. In another word, YANG model ran by OFM in our article as a visualizer tool to configure ODL controller [16].

2 Methodology and Implementation

In this work, an experimental test has been made on the ODL controller with topology shown in figure (1) with different case scenarios to show the SDN controller capabilities to deal with such scenarios. The SDN controller which control and manage the whole topology elements which are responsible to forward the traffic. Those switches known as Open Virtual Switches (OVS) can act as a real device. A test made with unlimited of (i) Round Trip Time (RTT) using ping command to check the connectivity among host and their linked switches via ICMP packets. (ii) Throughput and Burst Rate to check the maximum number of packets per second and per time-period. (iii) Response Time to measure how much of time needed to convey the packets and how much those packets may cost a time to travel completely from node to node. Then, Wireshark tool has been used to capture, monitor, and analyse the real time packets forwarding when the controller communicates with network devices. The subsection 2.1 to 2.4 show the implantation and scenarios. Finally, a comparison has been made with this work with other work regarding the mentioned parameters in the Results.

2.1 Network Emulator, Topologies and Networking Devices

In this experiment, the main network emulator is GNS3 version 2.2.27 to host those network devices. A star topology has been used to connect the network devises with class C network starts with IP address of 192.168.122.X/24 as depicted in figure (1). Then, a centralized Cisco switch installed to connect Windows server, ODL server, OFM server, with multiple OpenvSwitches that connect the Firefox end-user devices. The aim of Windows server in the topology to provide a DHCP IP pool to the network to prove that SDN controller can overcome the network scalability. The whole network connected to the internet with NAT

technology to install the controllers from online resources. Finally, Wireshark tool has been installed to test and monitoring the network at microscopic level by capturing the network packets.

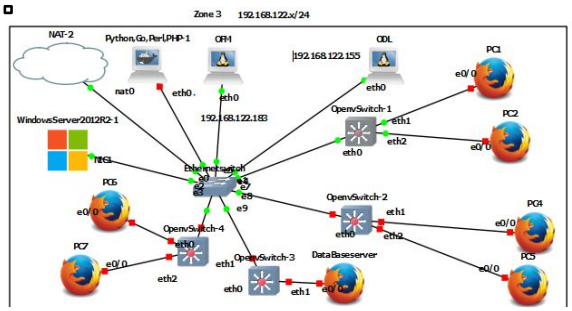

Fig. 1. The network topology

2.2 OpenDaylight controller (ODL) & OpenFlow Manager (OFM) Servers

In the topology, an OpenFlow management tool installed and used inside a separate Linux Ubuntu server (running on OFM server) to write OpenFlow rules to OpenvSwitches and visualize the OpenFlow topologies that run on OpenDaylight located on the northbound to manage the OpenFlow communication networks via RESTCONF API OpenFlow. Lastly, Open Daylight installed on the Open Virtual Appliance (OVA) running in VMware machine. ODL controller has been installed in another separate Linux Ubuntu server to provide the SDN core functionalities as shown in figure (1). In this experiment, the topology has four OpenvSwitch and two PCs are connected to each OpenvSwitch switch, and the OpenvSwitch switch is connected via Ethernet 0 to both ODL controller server, and OFM server. The first step was testing the connectivity among all devices in the topology. Then, a configuration has been made to all OpenvSwitch switches to connect ODL controller with IP address 192.168.122.155 on TCP port 6633. Once a connectivity achieved, the ODL sent a dump flow table 0 to all OpenvSwitch switches using OpenFlow 1.3 as shown in figure (3). This data link type is link layer discovery protocol (LLDP), a flow entry was written with a priority of a 100 and the traffic is being sent to the controller.

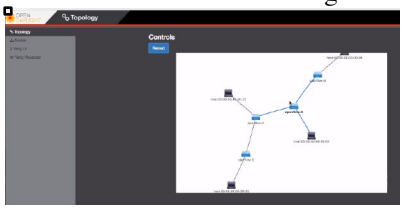

Fig. 2. OFM GUI

```
# ovs-ofctl -O OpenFlow13 dump-ports br0
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
  port LOCAL: rx pkts=8259, bytes=429669, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=45622, bytes=26933167, drop=0, errs=0, coll=0
           duration=5495.602s
  port  1: rx pkts=52992, bytes=34992482, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=61051, bytes=32741459, drop=0, errs=0, coll=0
           duration=5495.612s
  port  2: rx pkts=883, bytes=81613, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=54014, bytes=35095712, drop=0, errs=0, coll=0
           duration=5495.604s
```

Fig. 3. Traffics on Port 2.

2.3 Case studies and scenarios

Case(i): For a core security reason, any PC must not access the ODL server for several reasons such as centre-directional administrative reason, server hacking, viruses, and malware affects/attack. So, the admin can block any network traffic coming from any non-authorized device to ODL server to have the accessibility and preserve this level of security. The admin can drop any packet coming from any PCs but still the PCs have the connectivity to all other devices except ODL server. Hence, from flow management menu the admin can add new flow entry to drop a any packet at any port. One ODL flow table can be assigned to configure hundreds OpenvSwitchs in case of future topology expandability, scalability, and network's adaptations. On the other hand, the same applicable to security reason, ODL

offers the connectivity of any devices in the topology via MAC address as shown in figure 4. So, the case above can be also applied through MAC address restriction for thousands of unauthorized devices in one flow entry to reserve time and configuration effort.

Case(ii): Let's suppose that the network user in our topology heavily consuming the network traffic that effects the load of each network device with delivery latency and delivery failure or network device(s) failure (Load balancing). So, based on the network administration policy, it is better to limit the packet transfer speed to reduce the network overload among the enterprise branches as depicted in figure 4.

3 Results and Discussions

In this article, an investigation has been made to compare ODL controllers for an enterprise in a practical scenario. To compare and prove the controller capabilities, a Wireshark has been installed to analyse various packets. Since, OpenDaylight supports multiple southbound protocols, the investigation relied heavily on OpenFlow Switch Protocol's Read-State such as OFPT_STATS_REQUEST, OFPT_- STATS_REPLY, OFPT_PACKET_IN shown in figure 4. Those are to analyse the real-time packet's current configuration, statistics, and capabilities have been chosen randomly. Lastly, a comparison had been made with others ODL controller in [17] as shown in Table (1).

Table 1. ODL test parameters

Test Parameters	This article ODL controller test	Others ODL controller test
OpenFlow Response time	0.12065986 sec	0.190395-0.18951
OpenFlow Delay	0.009938 sec	0.444
OpenFlow length	20 bytes	20 bytes
OpenFlow configuration time	0.1596354 sec	0.212990-0.211245

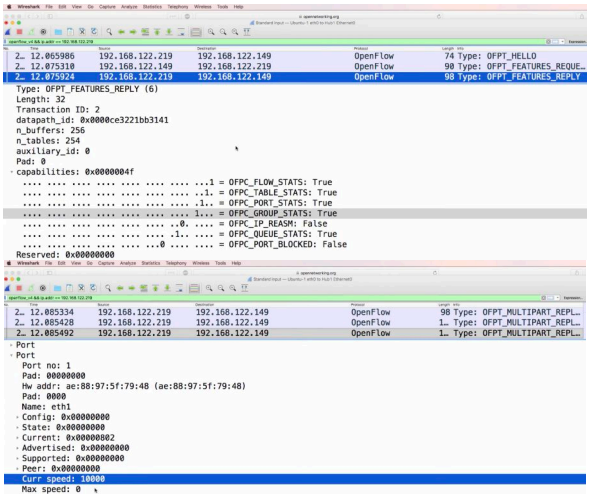

Fig. 4. Wireshark capturing test parameters

References

1. M. Mohammad & B. Eldin, A. & S. Mohamed. Software Defined Networking concepts and challenges. 79-90. 10.1109/ICCES.2016.7821979 (2016).
2. Casado, Martin & Koponen, Teemu & Shenker, Scott & Tootoonchian, Amin. Fabric: a retrospective on evolving SDN. 10.1145/2342441.2342459 (2012).
3. Zerwas, Johannes & Blenk, Andreas & Kellerer, Wolfgang. Optimization Models for Flexible and Adaptive SDN Network Virtualization Layers (2016).
4. Kreutz, Diego & Ramos, Fernando & Veríssimo, Paulo & Esteve Rothenberg, Christian & Azodolmolky, Siamak & Uhlig, Steve. Software-Defined Networking: A Comprehensive Survey. ArXiv e-prints. 103. 10.1109/JPROC.2014.2371999 (2014).
5. M. Casado, N. Foster, A. Guha 'Abstractions for software-defined networks', Magazine Communications of the ACM, Volume 57 Issue 10, October 2014 Pages 86-95. Available at http://www.cs.cornell.edu/~jnfoster/papers/sdn-abstractions.pdf (Accessed: 16 Jan 2020) (2014).
6. M. Casado, T. Koponen, S. Shenker, A. Tootoonchian 'Fabric: A Retrospective on Evolving SDN', HotSDN'12, August 13, 2012, Helsinki, Finland. Available at http://yuba.stanford.edu/~casado/fabric.pdf (Accessed: 15 Jun 2020) (2012).
7. Monaco, Matthew & Michel, Oliver & Keller, Eric. Applying Operating System Principles to SDN Controller Design. 10.1145/2535771.2535789 (2013).
8. Yeganeh, Soheil & Tootoonchian, Amin & Ganjali, Yashar. On scalability of software-defined networking. Communications Magazine, IEEE. 51. 136-141. 10.1109/MCOM.2013.6461198 (2013).
9. Zaalouk, Adel & Khondoker, Rahamatullah & Marx, Ronald & Bayarou, Kpatcha. OrchSec: An Orchestrator-Based Architecture For Enhancing Network-Security Using Network Monitoring And SDN Control Functions. Network Operations and Management Symposium (NOMS), 2014 IEEE. 10.1109/NOMS.2014.6838409 (2014).
10. Shalimov, Alexander & Zuikov, Dmitry & Zimarina, Daria & Pashkov, Vasily & Smeliansky, Ruslan. Advanced study of SDN/OpenFlow controllers. ACM Proc. CEE-SECR. 10.1145/2556610.2556621 (2013).
11. Rathi, Vipin & Singh, Karan. SDN Layer 2 Switch Simulation Using Mininet and OpenDayLight. 10.1007/978-981-10-8533-8_30 (2018).
12. Eftimie, Alexandra & Borcoci, Eugen. SDN controller implementation using OpenDaylight: experiments. 477-481. 10.1109/COMM48946.2020.9142044 (2020).
13. Wang, Lu & Sun, Meng & Tang, Shaoju. SCSCDaylight: Network Monitoring Tools for Software-Defined Networks Based on Opendaylight. 320-323. 10.1109/ICICAS48597.2019.00075 (2019).
14. Alaasam, Ameer & Radchenko, Gleb & Tchernykh, Andrei. Comparative Analysis of Virtualization Methods in Big Data Processing. 6. 48-79. 10.14529/js190107 (2019).
15. Kim, Wonho & Sharma, Puneet & Lee, Jeongkeun & Banerjee, Sujata & Tourrilhes. Automated and Scalable QoS Control for Network Convergence (2010).
16. L. Méndez, Jorge & Perdices, Daniel & Alvarez, Victor & G. Dios, Oscar & King, Daniel & Lee, Young. YANG data model for Flexi-Grid media-channels (2017).
17. Badotra, S., Panda, S.N. Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. Cluster Comput 23, 1281–1291 (2020)