Traffic Monitoring in Software Defined Networks Using Opendaylight Controller

Duc-Hung Luong(✉), Abdelkader Outtagarts, and Abdelkrim Hebbar

Nokia Bell Labs, Nozay, France
{duc_hung.luong, abdelkader.outtagart, abdelkrim.hebbar}@nokia.com

Abstract. Software Defined Network (SDN) is an emerging approach in network technology appeared in recent years. In the SDN revolution, OpenFlow is defined as the first communication standard to separates the control plane and data plane this allows the control plane to be centralized by an OpenFlow controller. The rise of SDN and OpenFlow has changed the point of view of the conventional model for network devices. In this paper, we focus on QoS monitoring in SDN controller using OpenFlow protocol. We propose a monitoring method for collecting the statistic and calculating the throughput of link traffics. We also design a new forwarding algorithm for control plane that avoids bottleneck and provides load-balancing. To evaluate our algorithm in real time traffic, we setup a test-bed in Mininet where we design the applications using the devised algorithm in Opendaylight controller, a fully functional open-source project with a rich set of API.

1 Introduction

Nowadays, Software Defined Networking (SDN) [1] is becoming an emerging paradigm in network technology. SDN enables an easy way to control for the future innovation of network and cloud [2]. Enterprises deploy SDN because it promises speedy service provisioning flexibility and reduction of operating expenses. In SDN approach, the control planes are centralized to manage all the network devices. The controller also computes the routes and installs the corresponding rules to the remote switches. The installed rules are also changed to quickly adapt to workload or network infrastructure modifications. Hence, while the traffic engineering (TE) becomes more and more important in the network, the controller needs to continually monitor the network devices status and link states.

OpenFlow [3] is the dominant southbound protocol used for SDN context that connects the control plane and data plane. The OpenFlow controller can poll the switches by using statistic request messages and collect per-switch and per-flow statistics from these switches. However, the counter values from statistics messages are not sufficient for monitoring other QoS parameters such as network throughput and packet latency.

In this paper, we propose a solution to monitoring throughput performance, an important parameter of network QoS. Our contribution is to develop a traffic monitoring solution using OpenFlow features in a SDN network. Studying and choosing a

S. Boumerdassi et al. (Eds.): MSPN 2016, LNCS 10026, pp. 38–48, 2016.
DOI: 10.1007/978-3-319-50463-6_4

suitable OpenFlow controller is an important task in our work. OpenDaylight [4], a Java-based open source controller has been chosen for implementing the monitoring application. This application focuses on the throughput monitoring and flow forwarding. The prototype demonstrates that this solution is effectively suitable in the SDN context.

The rest of this paper is organized as following. Section 2 provides a background and related works regarding the traffic monitoring solution in SDN networks. Section 3 presents our framework running on the controller OpenDaylight. We evaluate the performance of throughput monitoring through simulation using Mininet [5] in Sect. 4. Finally, we conclude the paper in Sect. 5 and point out some perspectives for our future works.

2 Background and Related Works

SDN is considered as the programmable networks with the ability to design and operate flexibly, thus, it has a lot of attentions for networking innovation. SDN's architecture has three-layer model shown in Fig. 1. The application layer includes all applications that are handled in the network and the infrastructure layer refers to the data plane, which has been decoupled from the controller. The controller can be considered as the "brain" of the networks. Its role is the control point to relay information to the hardware below via *Southbound APIs* and to the applications above via *Northbound APIs*.

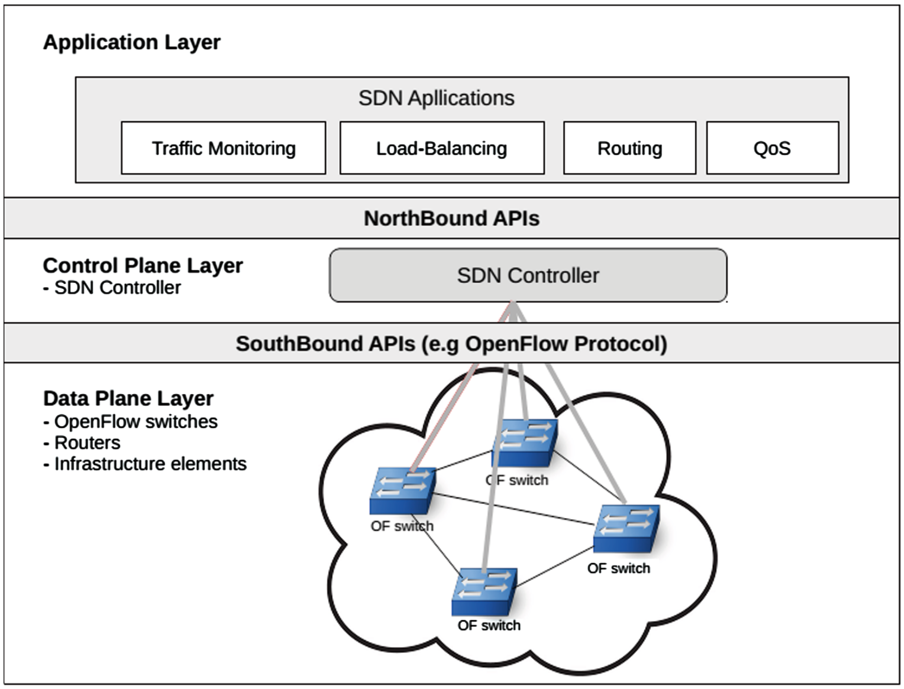


Fig. 1. Software defined network architecture.

Currently, traffic engineering (TE) is an important paradigm that provides the mechanism to manage, analyze and predict network behavior. As a result, network administrator can optimize performance and improve reliability of the network. Learning from the past, there are some existed solutions of traffic engineering for conventional network such as Asynchronous Transfer Mode (ATM) [6] network or Multi-protocol Label Switching (MPLS) [7]. However, the emergence of SDN paradigm that centralizes the control plane of distributed network devices requires new intelligent and adaptive TE mechanism. In [8], the authors consider that the scope of traffic engineering consists in four parts: monitoring for flow management, fault tolerance of network, topology update and traffic analysis. In this paper, traffic monitoring for flow management approach will be considered and focused.

In traffic monitoring paradigm, the monitoring application requires accurate and timely link statistics and device states. The traditional traffic monitoring using two approaches for measurement methods: *active measurement* and passive *measurement*. The *active measurement* method injects additional packets in to networks and monitors its behaviors. The simple example is the use of "*ping*" to determine the end-to-end connection status, compute the packets loss and discover the topology of networks. However, this measurement method produces interferences in to networks as well as being a cause of overhead. As opposed to *active measurement, passive measurement* is another approach for network monitoring while it does not inject traffic in to networks. This method measures the network traffic by observation and without generated overhead thus it does not influence network performances. However, passive method does not achieve results with good accuracy and is not always feasible for all networks.

Currently, there are some measurement protocols for the conventional network such as SNMP (RFC 1157) [9], NetFlow [10] from Cisco, sFlow [11] from InMon… Simple Network Management Protocol (SNMP) is an application layer protocol that uses passive sensors to help administrators in monitoring the network status. SMNP regularly polls the switches and requires scheduling carefully to monitor the entire network. However, SMNP is not suitable for flow-based monitoring and will not be considered in the present paper. NetFlow periodically collects the traffic information to NetFlow collector so that, the overall flow statistics are estimated based on these achieved samples. SFlow works similarly and uses time-based to collect packet sampling.

In SDN context, OpenFlow [3] protocol is chosen as the communication standard between the controller and network devices. OpenFlow enables controller to per-flow and per-switch monitoring that picked up in the several recent researches. OpenNetMon [12] is an active measurement tool designed as a Python-based module for POX controller [13]. This application monitors per-flow quality of service (QoS) metrics by adaptively polling switches at adaptive rates. OpenNetMon uses an adaptive method for changing the polling interval of statistic requests. Presented as a passive method, FlowSense [14] uses the controller messages such as *PacketIn* and *FlowRemoved* to manage the network and predict per flow link utilization. This method enables monitoring the SDN networks with zero measurement cost. Instead of actively querying the switches, the controller only uses the information from *PacketIn* messages and *FlowRemoved* messages. The traffic parse module in the controller captures flow traffic hence controller estimates the traffic utilization. FlowSense proposes a zero cost approach to monitor networks; however the estimation of FlowSense obtains inaccurate results due to long idle-time out before receiving *FlowRemoved* message. Payless [15] is another query-based monitoring framework for flow statistic collection of different levels. Payless monitoring provides a set of well-defined API which is very useful for different network applications to monitor and collect data based on it. OpenTM [16] is a monitoring scheme that estimates the traffic matrix (TM) of OpenFlow networks. This module is an application implemented for NOX, the first open-source OpenFlow controller. OpenSample [17] proposes the mechanism to monitor in real-time, low latency and flexible. It takes the advantage of sFlow to monitor high-speed networks.

In order to satisfy our context, we also study for choosing adaptable SDN controller. There are variety of OpenFlow controllers, with different languages and supported environments [18, 19]. NOX [20] is the first OpenFlow controller, but it is not the heavily implemented or used. NOX is a multi-threaded and C++ based program. Although NOX supports graphical interface and visualization tool, however its weakness is lack of documentation and performance. POX [13] is a Python-based OpenFlow controller inherited from NOX controller. In comparing to NOX, POX is more familiar to implement and use because of providing a web based interface. Beacon [21] is another Java-based OpenFlow controller that supports both event-driven and multi-thread. Same as Beacon, Floodlight [22] is Java-based framework under Apache licensed. Its core architecture is modular and provides many components, including topology management, device management, path computation and web access. Ryu [23] is a component-based frame work which has a set of predefined components. Ryu is implemented in Python and is supported by NTT labs. The components of Ryu can be modified, extended and customized for specific application. More recently, ONOS [24] offer a distributed SDN controller with similar features. However, ONOS focuses on specific tasks of service providers and will be not considered. In this paper, we investigate Opendaylight [4] for implementing our monitoring algorithm. Opendaylight is a Java-based open-source controller supported by many network vendors. Furthermore, Opendaylight has highly modular architecture with a rich set of APIs and features for SDN network.

3 System Design

3.1 Methodology

In an OpenFlow controller, the controller regularly queries *StatisticsRequest* message to retrieve the statistics of switches, ports, flows and flow tables of network switches. In the present work, we focus on the statistics of switch ports and flows via *StatisticReply* messages.

However, as specified in OpenFlow standard, we cannot retrieve directly the values of throughput of ports and flows. The OpenFlow controller only polls the statistics of switches by the number of transmitted packets or transmitted bytes. Hence, we propose a solution which regularly retrieves the traffic amount and consider that the throughput is equal to the average transmission rate by the time unit. In this work, our module receives the amount of S(bytes) in duration of time t(s) hence, the throughput is calculated as:

$$R = \frac{S}{t}(byte/s) \tag{1}$$

Polling of flows and ports is made regularly by the controller. The value of the interval duration is the same for each path in the network. In Opennetmon [12], the authors use the same solution in the monitoring module of POX controller. However, they consider the random or round robin policies that are not suitable for large scale network. In the present work, we only use the round robin policy to facilitate the solution because of the small network environment.

3.2 Architecture Design

In this section, we describe two modules for Opendaylight controller for throughput monitoring and packet forwarding in OpenFlow-based network.

Throughput monitor. As presented in the previous section, OpenFlow is a vendor-neutral standard communication that defines the interaction between controllers and switches. OpenFlow protocol-plugin is the southbound module that connects directly with the switches (real or virtual switch). In this work, the monitoring module regularly requests the OpenFlow protocol plugin to send *StatisticsRequest* messages to the switches. There are some types of statistics message such as *vendor, flows, table, port* in which we are interested to request the port state statistics. The switch that receives this request responds by a *StatisticsReply* message. More specifically in Opendaylight, OpenFlow protocol-plugin module implements services to capture and parse this message from switches. In our case, the information on switch ports and flows carried by the statistic message are collected and processed by the statistics module. We mainly focus on the number of packets and bytes passed by all ports to calculate the throughput. By storing the packets count and bytes from previous port state, the delta of these counters is obtained to determine the current transmission rate for each link. These values of throughput statistics are locally stored in the OpenFlow plug-in module. Another database is created in the controller to store all the statistics of port states and flow states. When OpenFlow protocol plug-in detects a change of a statistics counter, it automatically notifies the upper layer to update the database. Inside of the controller core, the AD-SAL module provides another service to read the statistics of all components in the network. Additionally, we modified the *StatisticsManager* module to retrieve the throughput data. This module also provides the APIs to the monitoring application. The user also retrieves the data by using REST - client via northbound interfaces (Fig. 2).

Packet Forwarding. Actually, when a conventional switch receives a packet, it learns the mapping between MAC address and port to match. If the switch already knows the specific destination, the packet is sent directly to the correct target switch port. Otherwise, the packet is flooded out all ports like in a hub. This functionality is called Layer 2 learning switch. In case of a flow-based switch managed by a controller, when a switch receives a new packet that is not matched with any installed flow table, it encapsulates the payload into a *PacketIn* message and sends this message to the

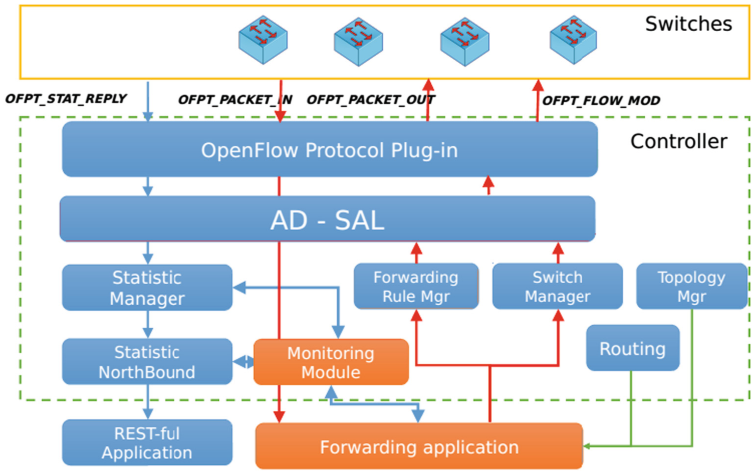


Fig. 2. Diagram of monitoring module and forwarding in Opendaylight

controller. The controller responds for installing a path using one or more *FlowMod* message and resends another *PacketOut* to the destination switch. In our application, we consider the routing in Layer 3 to find the shortest path. It is also looking for a new path while link utilization value of the shortest path is high. The decision to modify the packet path is based on the throughput result computed by the monitoring module. This method allows avoiding the collision and packet loss because the full link capacity is used at main path. It also provides the network load balancing because based on changing the path to the destination.

The algorithm used for monitoring and forwarding works as following. Firstly, the controller creates a list of rules that will be processed to install the flow. When the controller receives a *PacketIn* message, the application extracts the IP address of the destination host. Then, it looks up the destination host (*dest_host)* which corresponds to this IP address from the IP host table. The current switch (*curr_switch)* that sent *PacketIn* and the destination switch (*dest_switch)* that connects to this host is also tracked. By using Dijkstra algorithm [25], we can find the shortest path between *curr_switch* and *dest_switch*. This solution determines a unique path to destination and loop-avoiding in the network. From this path, the *next_switch* is defined as the next hop that connects directly with the *curr_switch*. The new flow is created with the IP address of the destination and the action is set to the egress port of *curr_switch*. We create a new rule that indicates that the flow will be processed by *curr_switch*. This rule is also added to the list of rules in order to the controller fabricates a new *FlowMod* message for *curr_switch*.

The results from monitoring module allow us to calculate the '*curr_switch – next_switch'* link throughput value and to compare this value to a fixed threshold. In case the link throughput exceeds the threshold, the link between *curr_switch* and *next_switch* is temporary disabled and another shortest path is computed. From this new path, we find the *new_next_switch* and the link respectively. We continuously compute the link throughput of the new link and compare it to the old link between *curr_switch* and *next_switch*. If the capacity rate of the new link is higher than the older one, we install the new flow on the temporary path.

4 Evaluation and Results

In this section, we present the performance of a demo for monitoring the throughput and forwarding the packet flows. For this application, two modules have been developed as described in previous sections. We will consider a topology with 6 switches and two scenarios for the demo (Fig. 3).

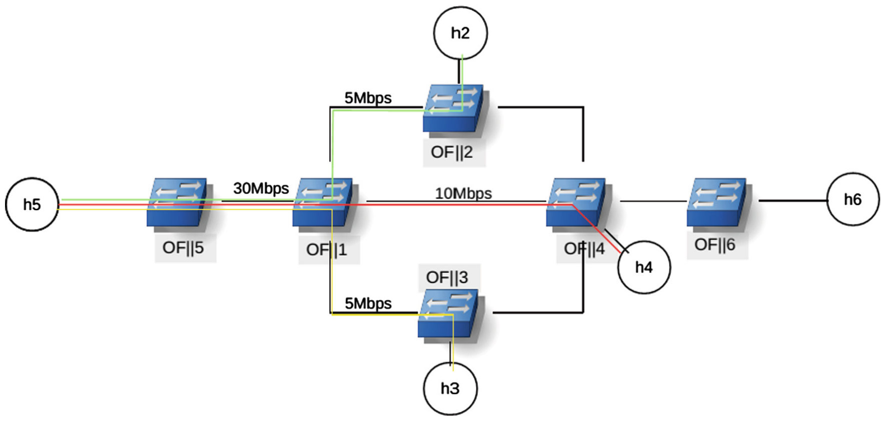


Fig. 3. Forwarding scenario for monitoring module

4.1 Evaluation

Monitoring module. Firstly, we consider the monitoring scenario while we try to capture the real-time traffic of link *OF1 – OF5*. In this scenario, the network topology is shown in the Fig. 3. We set the bandwidth of links between OF1 – OF2, OF1 – OF3, OF1 – OF4, OF1 – OF5 at 5 Mbps, 10 Mbps and 30 Mbps respectively. We use *Iperf* [26] packet generator to send and receive TCP packets between a pair of hosts. By running in client-server mode, *Iperf* will send TCP packets traffic to specific hosts at specific bandwidths. For example, H5 host runs *Iperf* at server mode and other hosts run *Iperf* in client mode. Traffic is generated from H2, H3, H4, to H5 with transmission rates of traffic equal to the max capacity value allowed on each link.

Figure 4 shows the results of the simulation during 1000 s. We firstly send 5 Mbps of traffic from H2 to H5 at time t = 0 s using *Iperf*. After 300 s, we continuously transmit the traffic from H3 to H5. This traffic has a transmission rate equal to the max traffic between H3–H5. Hence, from t = 300 s, H5 receives the traffic at 10 Mbps that

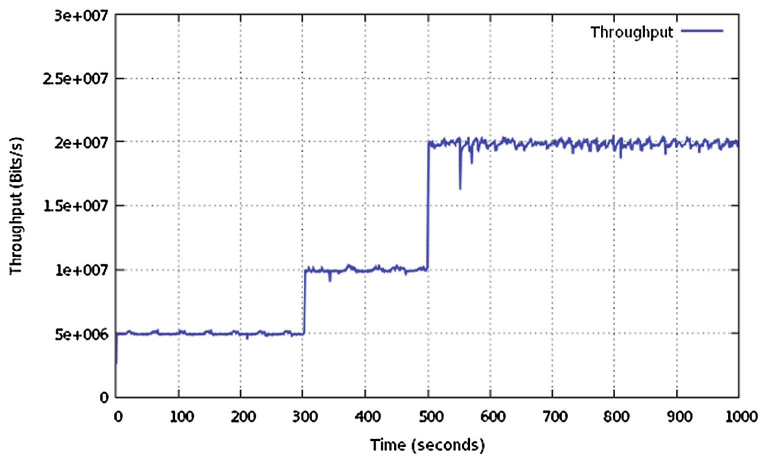


Fig. 4. Results of throughput monitoring

equals to the cumulative traffic sent from H2 and H3. Similarly, at t = 500 s we generate the traffic from H4 to H5 and the total traffic of link H1 – H5 consequently increases until 20 Mbps.

Forwarding scenario. In this scenario, the link bandwidths are set equally to 10 Mbps for every link in the network and the demonstration considers the connection path between H5 and H6 to verify the algorithm (Fig. 5).

Firstly, we consider the normal scenario to obtain the default behavior of network. We send the ICMP packet via the *ping* command in Mininet to confirm the paths. The shortest path found from H5 to H6, using Dijkstra [25] routing algorithm, is via OF1, OF4 and OF6. The flows installed in the switches are listed in Table 1. The Web GUI of Opendaylight can be used to check that the flows have been successfully installed.

Now we consider the load balancing scenario. We clear all the existed flows in the network via RestClient application and northbound API (Fig. 5). We also set the threshold value for changing the path at 80%. Then we generate some traffic between OF1 and OF4 such as the threshold is exceeded.

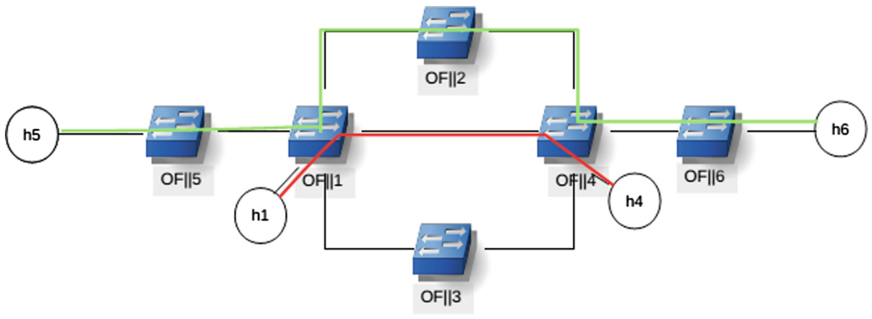
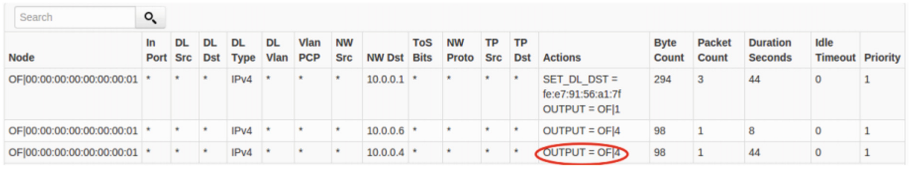


Fig. 5. Forwarding scenario for load balancing

Table 1. Flows created in normal scenario

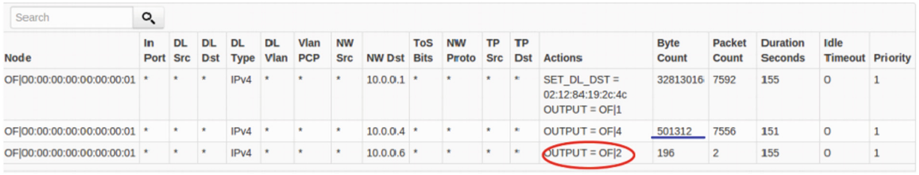
Switch	IP of destination	Action
OF5	10.0.0.6	Output = Port 2
OF1	10.0.0.6	Output = Port 4
OF4	10.0.0.6	Output = Port 5
OF6	10.0.0.6	Output = Port 6

When the traffic between these switches passes the threshold value, the link between OF1 and OF4 is temporary set unavailable and the controller looks up other paths for load balancing. The result can be verified when we ping again *h5 ping h6*, the path between H5 and H6 is changed by the load balancer. As a result, we obtain automatically the new shortest path H5 – OF5 – OF1 – OF2 – OF4 – OF6 – H6 (Fig. 5) and the flow table of these switches has been also changed to complete the path installation (Figs. 6 and 7).


Search

Node	In Port	DL Src	DL Dst	DL Type	DL Vlan	Vlan PCP	NW Src	NW Dst	ToS Bits	NW Proto	TP Src	TP Dst	Actions	Byte Count	Packet Count	Duration Seconds	Idle Timeout	Priority
OF|00:00:00:00:00:00:00:01	*	*	*	IPv4	*	*	*	10.0.0.1	*	*	*	*	SET_DL_DST = fe:e7:91:56:a1:7f OUTPUT = OF|1	294	3	44	0	1
OF|00:00:00:00:00:00:00:01	*	*	*	IPv4	*	*	*	10.0.0.6	*	*	*	*	OUTPUT = OF|4	98	1	8	0	1
OF|00:00:00:00:00:00:00:01	*	*	*	IPv4	*	*	*	10.0.0.4	*	*	*	*	OUTPUT = OF|4	98	1	44	0	1

Fig. 6. Flow table of switch OF1 in default scenario


Search

Node	In Port	DL Src	DL Dst	DL Type	DL Vlan	Vlan PCP	NW Src	NW Dst	ToS Bits	NW Proto	TP Src	TP Dst	Actions	Byte Count	Packet Count	Duration Seconds	Idle Timeout	Priority
OF|00:00:00:00:00:00:00:01	*	*	*	IPv4	*	*	*	10.0.0.1	*	*	*	*	SET_DL_DST = 02:12:84:19:2c:4c OUTPUT = OF|1	32813016	7592	155	0	1
OF|00:00:00:00:00:00:00:01	*	*	*	IPv4	*	*	*	10.0.0.4	*	*	*	*	OUTPUT = OF|4	501312	7556	151	0	1
OF|00:00:00:00:00:00:00:01	*	*	*	IPv4	*	*	*	10.0.0.6	*	*	*	*	OUTPUT = OF|2	196	2	155	0	1

Fig. 7. Flow table of switch OF1 in load balancing scenario

4.2 Discussion

In this implementation, we design the traffic monitoring application using OpenDaylight controller. This application automatically monitors the bandwidth utilization of the links in the network and reactively chooses the best alternative path to destination. The forwarding component tracks the host location to install per-flow forwarding rules on the switches. This module also learns from the real-time traffic obtained by monitoring module to find other paths. In comparing with existing forwarding algorithms, we brought some new specific differences detailed below.

As the switches process too many flows, a bottle neck may occur. When the amount of data passing through one link in a short time is too large, the latency and packet loss will be clearly considered. In case of bottle neck, we chose a temporary path for load balancing and reduction of packet lost rate. We also analyzed the traffic based-on bandwidth utilization rate. This is an important indication to evaluate the Quality of Service (QoS) of a network. The installation of per-flow rule is done by all switches between *curr_switch* and *dest_switch* and iterates until all switches are configured. This method allows avoiding the repeated request of switches to the controller. We find a temporary path at a given moment but without changing the topology to adapt the dynamic of network. The link with high utilization will be considered at other moment to look up the best path. The forwarding module considers the switch acting as Layer 3 switch, that uses the routing module to find the shortest path based on Dijkstra algorithm. This is a simple case of routing algorithm but this mechanism allows choosing only a single path to host switch and avoiding loop in topology.

5 Conclusion and Perspectives

As the Software Defined Networking becomes an important part of the future network architecture, we must consider traffic monitoring as one of the important goals to adapt quickly to the network traffic changing. In this paper, we offer a solution for traffic monitoring using OpenFlow in Software Defined Networking. While there are many management protocols in SDN such as SMNP, NetFlow, sFlow…, OpenFlow is considered as a suitable standard to monitor the traffic in SDN networks. We also study about OpenFlow framework and we choose OpenDaylight as the best OpenFlow controller to deploy our implementation. The application consists of two main components: monitoring module and forwarding module. In the monitoring module, we regularly query the switch and calculate the flow bandwidth. The forwarding module tries to use the results of monitoring module to install per-flow forwarding rules. The shortest path is found by using Dijkstra algorithm to obtain the best path and avoid loops in the topology. In the last section, we have shown experimental results using our test bed.

The application of traffic monitoring in SDN networks presents many challenges. In this paper, we only use throughput as the QoS parameter for monitoring the traffic. For the future works, we will investigate other parameters such as latency and packet loss. The forwarding module in this paper changes the path at the initialization of flow. We presented a simple scenario for changing the path in the topology. The more complex scenarios such as automatically deleting a flow, deleting a link are also in the perspectives for the future of our works.

References

1. Open network foundation: software-defined networking: the new norm for networks. https://www.opennetworking.org/
2. Weldon, M.K.: The Future X Network: A Bell Labs Perspective. Nokia BellLabs, Murray Hill (2016)

3. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. **38**(2), 6974 (2008)
4. OpenDaylight controller. https://www.opendaylight.org/
5. Mininet. https://www.mininet.org/
6. de Prycker, M.: Asynchronous Transfer Mode: Solution for BroadBand ISDN, Ellis Horwood, Ltd., (1993)
7. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Lable Switching Architecture. Internet draft, draft-ietf-mpls-arch-01.txt (1998)
8. Akyildiz, I.F., Lee, A., Wang, P., Luo, M., Chou, W.: A roadmap for traffic engineering in SDN - OpenFlow networks. Comput. Netw. **71**, 1–30 (2014)
9. Simple Network Management Protocol. https://www.ietf.org/rfc/rfc1157.txt
10. NFC 3954: Cisco system NetFlow Services Export Version 9. http://tools.ietf.org/html/rfc3954.html
11. Phaal, P., Lavine, M.: sFlow Version 5. http://www.sfow.org/
12. van Adrichem, N.L.M., Doerr, C., Kuipers, F.A.: Opennetmon: network monitoring in openflow software-defined networks. In: Network Operations and Management Symposium (NOMS) (2014)
13. POX controller. https://www.noxrepo.org/pox/
14. Yu, C., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G., Madhyastha, H.V.: FlowSense: monitoring network utilization with zero measurement cost. In: Roughan, M., Chang, R. (eds.) PAM 2013. LNCS, vol. 7799, pp. 31–41. Springer, Heidelberg (2013)
15. Chowdhury, S.R., Bari, M.F., Ahmed, R., Boutaba, R.: PayLess: a low cost network monitoring framework for software defined networks. In: Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium, NOMS 2014 (2014)
16. Tootoonchian, A., Ghobadi, M., Ganjali, Y.: OpenTM: traffic matrix estimator for OpenFlow networks. In: Krishnamurthy, A., Plattner, B. (eds.) PAM 2010. LNCS, vol. 6032, pp. 201–210. Springer, Heidelberg (2010)
17. Suh, J., Kwon, T., Dixon, C., Rozner, E., Felter, W., Carter, J.: OpenSample: a low latency, sampling-based measurement platform for SDN, IBM report (2014)
18. Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., Sherwood, R.: On controller performance in software-defined networks. In: Hot-ICE12 Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (2012)
19. Khondoker, R., Zaalouk, A., Marx, R., Bayarou, K.: Feature-base comparison and selection of Software Defined Networking (SDN) controllers. In: World Congress on Computer Applications and Information Systtems (WCCAIS) (2014)
20. NOX controller. https://www.noxrepo.org/
21. Beacon controller. https://www.standford.edu/display/Beacon/Home/
22. FloodLight controller. https://www.projectfloodlight.org/floodlight/
23. Ryu controller. https://www.github.io/ryu
24. Open Network Operation System (ONOS). http://www.onosproject.org
25. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**, 269–271 (1959)
26. Iperf, The network bandwidth measurement tool. http://iperf.fr/