Application-aware Traffic Management for OpenFlow Networks

Seyeon Jeong, Doyoung Lee, Junemuk Choi, Jian Li and James Won-Ki Hong

Department of Computer Science and Engineering, POSTECH, Korea.
{jsy0906, dylee90, juk909090, gunine, jwkhong}@postech.ac.kr

***Abstract*—Software-Defined Networking (SDN) is an emerging networking paradigm aims to improve network management flexibility and efficiency. OpenFlow is the popular SDN de-facto standard, which has been prevalently adopted by both academia and industry for research and development purpose. OpenFlow provides rich programmable interface to network administrator to ease traffic monitoring and control. Because OpenFlow supports L4 network stack, it is feasible to provide application level traffic control by specifying TCP/UDP port number in flow rules. The major deficiency of the port-based traffic control is that it only provides the ability to control traffic from applications which have well-known TCP/UDP port numbers. In the case of port number change or dynamic (ephemeral) port allocation to an application, it is difficult to accurately control the application traffic. To be a solution, we propose an application-aware traffic management method by integrating Deep Packet Inspection (DPI) function with SDN controller. To show the feasibility, we design and implement Firewall and Bandwidth Manager applications based on the proposed management method. The applications perform on top of ONOS [1] controller, and FTP rate control example is shown to prove the feasibility of the proposed flow management method.**

Keywords—Software Defined Network, DPI, Traffic Management

I. Introduction

Nowadays Internet has become an essential element in our lives. People can use Social Network Service (SNS) and check their e-mail anytime and anywhere using Internet. Along with its influence, network technology also has advanced dramatically. However, still many deficiencies should be improved in the network area. For example, current networks consist of a huge number of network devices with a different specification and interface protocol for each vendor. It means that a network administrator should learn and follow the individual protocols to change a network policy or configuration. It also implies an impediment to network agility.

As the scale of Internet grows, traffic management becomes an important issue. Networking services require as much as possible bandwidth for their performance. Performance of video streaming service or file transfer application gets better as allocated bandwidth becomes larger while other traffic loses its allowance in the restricted resources. Thus, mediating those demands is an important role of a network operator to improve Quality of Service (QoS). There has been several literature work on traffic management to improve QoS. Wang et al. proposed an application oriented traffic management solution [2], while Sezer et al. introduced another traffic management approach [3] by controlling behavior of application traffic.

In order to improve the flexibility and management problems in traditional networks, SDN has been introduced. One of the important concepts of SDN is that it separates control plane and data plane which are combined in a traditional network device. Data plane consists of switches and routers which are dependent on control plane's decision about how to handle or forward packets. SDN controller, as a representative entity of the control plane, manages and monitors data plane entities in a centralized manner. To accomplish these roles, a demand for standard communication between a controller and switch resulted in the release of OpenFlow protocol [4] as a de-facto standard for SDN.

Traffic management can be implemented by either dropping packets or adjusting packet output rate of switch ports. The most representative use cases of traffic management are firewall and traffic shaper. With rich management features provided by OpenFlow, realizing traffic management functions in OpenFlow-based network could be much simpler than that of traditional network. In practice, traffic management in OpenFlow network is performed per application (service), and it can be realized by inserting flow rules with specific TCP/UDP port numbers. For example, to control HTTP traffic, we can insert a flow rule with TCP port number 80 and required action sets to the flow. The main deficiency of traffic management based on port numbers is that it only provides the ability to control traffic from applications which have well-known port numbers. In the case of port number change or dynamic (ephemeral) port allocation to an application, it is difficult to run the scheme. To resolve this issue, in this work, we propose an application-aware traffic management method by integrating Deep Packet Inspection (DPI) function with OpenFlow controller. The main idea of the proposed management scheme is that we classify application traffic using off-platform DPI instances, and feed the classification result into OpenFlow controller to correlate the result with flow rules in order to accurately control specific application traffic. Since by far DPI is one of the most accurate ways to classify application traffic, the proposed management scheme can also provide precise traffic management capability in OpenFlow networks. To show the feasibility of the proposed management scheme, we design and implement two representative applications - Firewall and Bandwidth Manager based on the proposed management scheme. With those two applications, the network administrator can specify the list of applications (traffic) that he wants to block or forward, and required upper bound of transmission rate for the applications.

To evaluate performance of the proposed management scheme, we chose ONOS [1] as a reference SDN controller to implement the applications. ONOS is a distributed SDN controller motivated by performance, scalability, and availability requirements for large operator networks. ONOS has being considered one of the most promising SDN controllers due to its flexible programmability and rich core APIs. In terms of programmability, it supports OSGi-based framework allowing administrators to run controller applications selectively according to their policy. It also offers core APIs on each SDN component based on its well-defined architecture. We implemented Firewall and Bandwidth Manager as an ONOS application and evaluated their feasibility by showing the use case of controlling transmission rate of FTP traffic.

The remainder of this paper is organized as follows. Section II presents related work and background. In section III, the detailed design and implementation of the proposed scheme are described. Performance testing is shown at section IV and we conclude this paper at section V.

II. Related Work

A. SDN Controller

Since the concept of SDN emerged, a large number of SDN controllers have been introduced. OpenDaylight (ODL) [5] features multiple South Bound Interface (SBI) support as a plugin. With the SBI support, ODL application developers can focus on implementing controller applications without knowledge of the protocol specifics. Ryu [6] is a component-based SDN controller which is written in Python. Ryu provides software components with well-defined APIs. It allows developers to create new applications for the purpose of controlling and managing the underlying network. Floodlight [7] is a Java-based OpenFlow controller. It offers a module loading system to extend and enhance the controller more easily.

B. Traffic Management in SDN

There are many researches on traffic management exploiting SDN features. FlowGuard [8] is a SDN-based firewall that provides control over traffic in a packet or flow level. It also considers and avoids possible flow rule collisions in installing a new flow rule under a certain firewall policy. It seems to be a more advanced and compatible firewall implementation with SDN architecture than other built-in firewalls in existing open source controllers. However, it does not support any application or protocol-level functionalities based on traffic classification.

As an effort to the application-aware SDN, M Jarschel [9] proposed an application buffer-based traffic engineering on YouTube traffic over an OpenFlow network. The research showed that performance improvement coming from multi-path routing for the traffic but it has a limitation on the specific application traffic, YouTube.

In Atlas [10], Zafar Qazi presents a self-contained application-awareness system over SDN using machine learning approaches rather than DPI. The system applies a machine learning algorithm to classify application traffic from the packets stored in the switch buffer. However, it cannot control transmission rate of a specific application traffic in a fine grained manner. It just adjusts drop precedence of packets according to a service type group.

Meter table and Meter band are most recent QoS related extensions since OpenFlow 1.3. They can be used to limit packet transmission rate of OpenFlow switch ports. A meter band specifies an upper bound of transmission rate and a set of actions to be performed for associated flows once they exceed the specified rate limit [11]. A meter table stores a collection of meter bands and selects one of the bands with the highest rate lower than the measured rate for the associated flow [12]. We used these features in our scheme to implement the rate limit function more effectively.

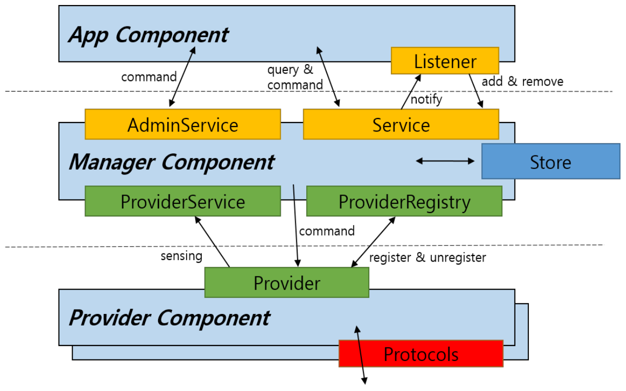


Fig. 1. ONOS Architecture

C. ONOS Architecture

Our proposed system allows OpenFlow-based network to apply firewall and rate limit policies to incoming traffic depending on its application or protocol. An application name of a flow is identified by either TCP/UDP port number or DPI result on it. Policies related to each network function can be specified in a predefined format by a network administrator through a web-based user interface for policy management. In Fig. 1, the App component on which our system is placed has an interface to receive a certain packet header or aggregated information of flows such as OpenFlow Packet-In messages, flow or meter statistics from each Manager component. The Manager component implements one of core SDN abstractions such as Flow, Topology, Routing and so on. The Provider component stands for a SBI implementation such as OpenFlow. The App component (ONOS application) decides network policies depending on its desirable functions using data, queries and commands from each Manager component. These network policies should be divided and translated into related commands for each Manager component. Then the commands are delivered to the OpenFlow Provider component which constructs desirable flow rules to be installed into the underlying switches in order for actual deployment of the network policies. The layered architecture of ONOS provides more flexible programmability on SDN to application developers.

III. Design and Implementation

In this section, we introduce the design and implementation of the proposed system. The architecture of the system shows up in the Fig. 2. Traffic Management System which includes Firewall and Bandwidth Manager is an ONOS application. The system also utilizes DPI instances. Most simple way to run DPI over a network is placing a packet mirroring machine in front of each ingress switch to copy packets for DPI and relay original packets to the switch at the same time. However, in our approach, we assume that the ingress switch is a high-end general purpose machine whose performance is enough to deploy DPI and OpenFlow switch functions in itself and each of them does own work in order on incoming packets [13]. This assumption can imply the use of network function virtualization concept and lead to flexible operation of the system by using OpenFlow features of the switch. For example, DPI offloading is applicable based on flow rules to distribute traffic across multiple DPI instances. When a new flow arrives at an ingress switch, DPI function on the switch performs DPI and then sends the result (application name of the flow) to the system by using REST API. Based on the architecture and other OpenFlow features described in the implementation section later, Traffic Management System can perform traffic management in terms of how much transmission rate should be limited to a certain flow and whether the flow should be forwarded or not. Those decision makings conform the firewall and rate limit policies which can be specified by a network administrator using the user interface implementation based on ONOS web UI.

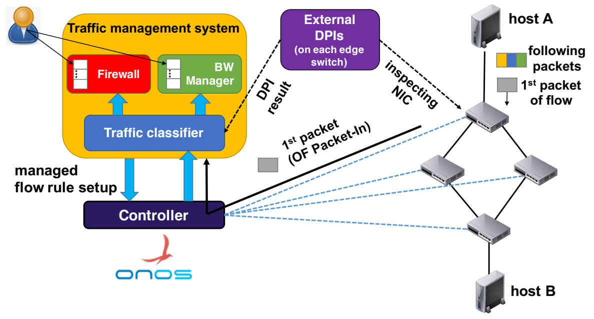


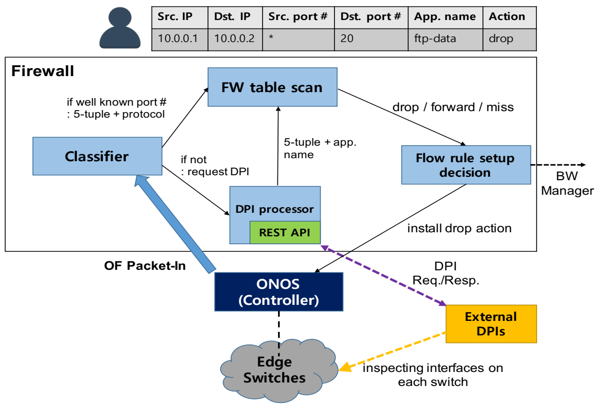
Fig. 2. Overview of Application-aware Traffic Management System

A. Testbed Setup

In our implementation, we use Mininet [14] to emulate an OpenFlow-based network equipped with single ONOS controller. We deploy several Ofsoftswitch13-CPqD [15] instances as OpenFlow software switches on the testbed to utilize the OpenFlow Meter table implementation. nDPI [16] as a DPI instance of the system is an open source DPI library capable of inspecting over 200 application protocols with high accuracy while it can operate as a standalone DPI box. Each nDPI instance operates on an ingress switch and if any request from the system occurs, it returns the DPI result of current incoming flow with a timestamp. Their communication is based on REST API. In the following sections, we elaborate on two network functions in the system, Firewall and Bandwidth Manager, including their functional requirements and interfaces with other components of the system.

B. Firewall

Generally, for OpenFlow based network in the reactive mode [17], there would be no corresponding flow rules in a switch when a new flow comes in. In this case, an OpenFlow Packet-In message containing header values from layer 2 to layer 4 of the first (arrived) packet of the flow is delivered to the connected controller from the switch in order to conform a policy descision. After that, the controller parses the message and extracts information about the packet such as source/destination IP addresses, TCP/UDP port numbers and protocol identifier to decide a routing path. Firewall in our traffic management system (Fig. 3) utilizes this 5-tuple information of the flow to identify an application protocol by comparing it with the internal well-known port number table (port-based classification). If it does not have a well-known port number due to port number chagne or dynamic port allocation, Firewall requires the DPI instance to report a DPI result including application or protocol name of the flow. To deliver the DPI result, REST API is used between them (DPI-based classification). Specifically, from the Fig. 2, all the flows generated from a host should pass the DPI instance (function) deployed in the connected ingress switch (function) machine so that the instance can analyze a flow and send an application or protocol name of the flow to the system with a timestamp. After that, the Firewall compares the 5-tuple information from the initial Packet-In message to the DPI result with the timestamp consideration for accuracy. Therefore, support for synchronous DPI request/response is needed for the comparison process in spite of delay increase. Finally, by using one of the two approaches, port or DPI-based classification, Firewall can be aware of an application or protocol name of the flow, so it can handle the identified flow according to a corresponding (matching) policy after scanning the firewall rule table.



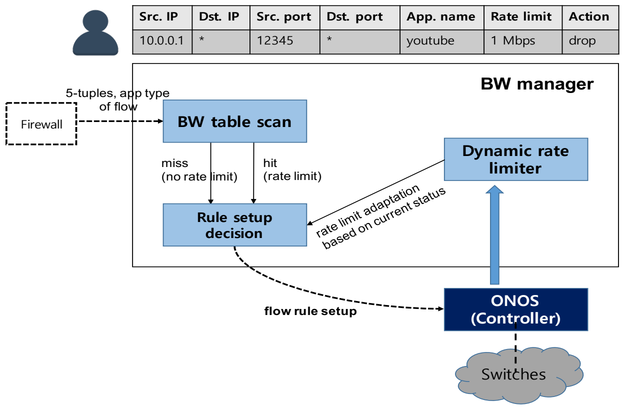
Src. IP	Dst. IP	Src. port #	Dst. port #	App. name	Action
10.0.0.1	10.0.0.2	*	20	ftp-data	drop

Fig. 3. Design of Firewall

Firewall maintains a firewall rule table created by a network administrator. The specification of a firewall rule consists of <source IP address, destination IP address, source port number, destination port number, application name, action>. If a firewall rule requires a certain flow to be blocked, Firewall asks for the controller to drop all the packets of the flow, and then the controller sends a OpenFlow Flow-Mod message to the ingress switch to install a flow rule for drop of the packets. Consequently, flows under the policy cannot enter the network anymore from the ingress switch. Meanwhile, flows allowed to enter the network according to related firewall rules can be handled by the next management stage, Bandwidth Manager.

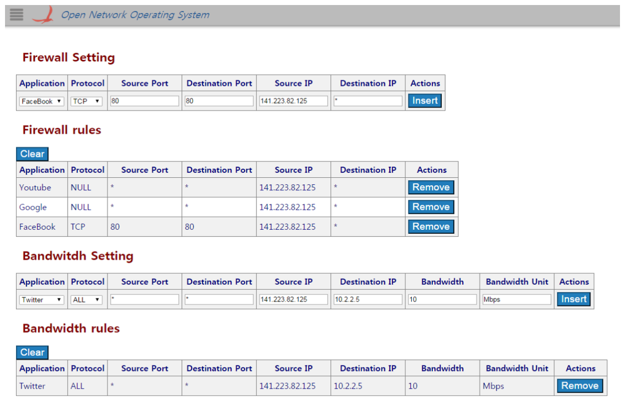
C. Bandwidth Manager

Bandwidth Manager in the Fig. 4 stores rate limit rules as a table as Firewall does. Each rule is composed of 5-tuple information, application or protocol name, and transmission



Src. IP	Dst. IP	Src. port	Dst. port	App. name	Rate limit	Action
10.0.0.1	*	12345	*	youtube	1 Mbps	drop

Fig. 4. Design of Bandwidth Manager



Firewall Setting

Application	Protocol	Source Port	Destination Port	Source IP	Destination IP	Actions
FaceBook	TCP	80	80	141.223.82.125	*	Insert

Firewall rules

Clear

Application	Protocol	Source Port	Destination Port	Source IP	Destination IP	Actions
Youtube	NULL	*	*	141.223.82.125	*	Remove
Google	NULL	*	*	141.223.82.125	*	Remove
FaceBook	TCP	80	80	141.223.82.125	*	Remove

Bandwitdh Setting

Application	Protocol	Source Port	Destination Port	Source IP	Destination IP	Bandwidth	Bandwidth Unit	Actions
Twitter	ALL	*	*	141.223.82.125	10.2.2.5	10	Mbps	Insert

Bandwidth rules

Clear

Application	Protocol	Source Port	Destination Port	Source IP	Destination IP	Bandwidth	Bandwidth Unit	Actions
Twitter	ALL	*	*	141.223.82.125	10.2.2.5	10	Mbps	Remove

Fig. 5. Web GUI

rate limit values (in Mbps) for a flow. An administrator can fill up the specifications through the web-based user interface in the Fig. 5. Bandwidth Manager performs traffic management by restricting transmission rate of flows. If there is a rate limit rule for a certain (identified) flow in the rule table, Bandwidth Manager installs related flow rules into switches on the routing path of the flow to be decided by Routing component of ONOS controller after this stage. Each of the flow rules is associated with several OpenFlow Meter bands. The Meter bands stand for the specified transmission rate limit values above and force the corresponding flow to be forwarded from the switches at a lower rate than the specified value, such as 100 Mbps or 0.5 Mbps.

The controller requires each switch to report meter and flow statistics periodically and then relays each report to Dynamic Rate Limiter in Bandwidth Manager. Each entry of meter statistics consists of meter id associated with certain flow rules and the total number of packets and bytes affected by the metering. Dynamic Rate Limiter can change the currently applied meter band dynamically according to the flow statistics of the metered flow. For example, on a link with 100 Mbps capacity, suppose there are several in-flight flows with a 30 Mbps average throughput and the number of packet drops due to associated meter bands is excessive. Then, it is obvious that too many packet drops occur unnecessarily under the current 30% link utilization. Thus, the Dynamic Rate Limiter can conclude that the currently applied meter bands decrease QoS of the in-flight flows excessively. Therefore, the Dynamic Rate Limiter decides each of the flows should be affected by a larger meter band. Because a network administrator can register several meter bands for a certain flow, the limiter can select the larger one and replace the old meter with the new larger one by sending the OFPT_METER_MOD messages to the related switches while it notifies the changes to the administrator. Computation on the link utilization after the adaptation also should be considered. This operation is implemented with the help of OpenFlow Meter table, Meter band and meter/flow statistics.

Finally, flow rules to apply firewall and rate limit rules for the flow should be installed into the switches on the routing path. The flow (traffic) then moves around the network under the intended policies from the administrator.

IV. Evaluation

To evaluate our proposed system, we constructed the system on a Mininet testbed emulation by using a hardware server equipped with hexa-core 2.67GHz Intel Xeon CPU and 20 GB of RAM. Both the sender and receiver host were connected with the OpenFlow-enabled software switch (Ofsoftswitch13-CPqD [15]). The switch had flow rules to apply firewall and rate limit policies on certain flows from the connected ONOS controller. A DPI instance (nDPI [10]) located in the same machine with the switch inspected packets back and forth between the two hosts and sent the results to the system. Evaluation scenario was as follows; 1) First, we established a FTP session between two hosts. We observed that FTP control packets with the TCP port number 21 were identified by the well-known port number table in the system and then the ONOS controller installed related flow rules for the flow into the switch. 2) Sender host started to transmit a 100 MB dummy file over the established FTP session. Chunks of the file were encapsulated in several FTP data packets with the TCP port number 20, but in this case, simply for checking the functionality, the system was modified to identify them after receiving the DPI result indicating that the flow is FTP data packets rather than retrieving the well-known port number table. 3) After 10 seconds from the beginning of transmission, we applied a rate limit policy up to 0.5 Mbps for the flow (ftp-data) and then changed the limit value to 0.2 Mbps at the 20 second. In the Fig. 6, measured throughput at the receiver host was about 1.2 Mbps without rate limit during the first 10 seconds. The new 0.5 Mbps rate limit was applied at the 10 second but it took effect on about 2.5 seconds later. Similarly, the effective time for the 0.2 Mbps rate limit was observed at the receiver host 2.5 seconds later. We could estimate that the major reason of the delay was caused by the amount of time to process the flow inside the system including additional blocking time for receiving the DPI result over REST API. The delay was enough time for the consecutive packets of the flow to saturate the buffer of the switch right before the actual effect of the rate limit. This condition could give unexpected high throughput measurement at the receiver host during a few seconds. The delay can vary with complexity of the network and throughput measurement program.

Current OpenFlow Meter table implementation supports simple packet drop and adjustment of drop precedence by

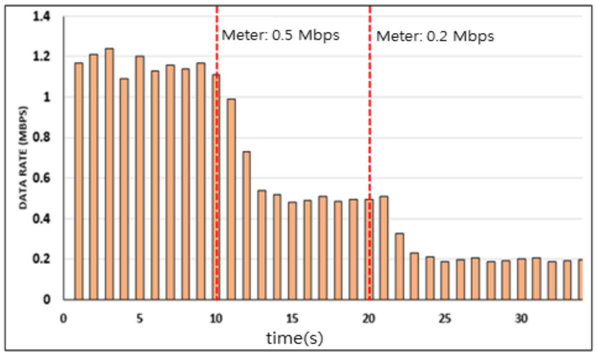


Fig. 6. FTP Throughput Measurement

marking the DSCP field in the IP header for the excessive packets [12]. In our evaluation, we used the simple packet drop scheme and observed repeated patterns that once measured throughput at a moment had been larger than applied rate limit value (meter), next measurement decreased temporarily according to packet drops of the excess, and then it recovered up to the limit value again.

V. Conclusion

In this paper, we have proposed the application-aware flow management system including two network functions, Firewall and Bandwidth Manger. The system is implemented as an ONOS application interacting with DPI instances for traffic classification. In the OpenFlow-based network operating in the reactive mode, when a switch has no flow rules for a newly incoming flow, it sends a Packet-In message with header values of the first packet of the flow to the connected controller. The controller then checks the header values to decide a routing path of the flow. By using this characteristic, we developed Firewall which is responsible for determining whether the switch should block or forward the flow. If the header information is not enough for the flow to be classified as a certain application traffic, it sends a request for DPI analysis of the flow to connected DPI instance. Similarly, Bandwidth Manager can use the same information about the flow to apply a rate limit policy in order to restrict bandwidth of a certain flow and guarantee more bandwidth for other preferable flows in the given link capacity. Therefore, the proposed system can manage network traffic over SDN at a lower cost compared with traditional approches for the same function but more overheads such as packet redirection to a DPI box and traffic conditioner.

Although, the major functions of the system work well in the simple testbed in the evaluation, there are rooms for improvements. Particularly, a testing on a large network with more complex topology and diverse traffic should be performed in terms of scalability. More various policies from network administrators also should be considered in terms of compatibility.

For the future work, we will further improve the rate limit function to operate more wisely based on current network condition. It may include additional algorithms to select optimal rate limit policy adaptively and more tuning parameters for Dynamic Rate Limiter in the Section 3.

Acknowledgement

This work was supported by the ICT R&D program of MSIP/IITP, Republic of Korea. [B0190-15-2011, Korea-US Collaborative Research on SDN/NFV Security/Network Management and Testbed Build]

References

[1] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[2] W.-H. Wang, M. Palaniswami, and S. H. Low, "Application-oriented flow control: fundamentals, algorithms and fairness," *Networking, IEEE/ACM Transactions on*, vol. 14, no. 6, pp. 1282–1291, 2006.

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

[4] Open Networking Foundation., *OpenFlow Switch Specification Version 1.0.0*, Std., Dec. 31, 2009.

[5] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 1–6.

[6] "Ryu: Component-based software defined networking framework." [Online]. Available: http://osrg.github.io/ryu/

[7] "Floodlight: Open source software for building software-defined networks." [Online]. Available: http://www.projectfloodlight.org/floodlight/

[8] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: building robust firewalls for software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 97–102.

[9] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "Sdn-based application-aware networking on the example of youtube video streaming," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*. IEEE, 2013, pp. 87–92.

[10] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in sdn," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 487–488.

[11] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "Policycop: an autonomic qos policy enforcement framework for software defined networks," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. IEEE, 2013, pp. 1–7.

[12] P. M. Mohan, D. M. Divakaran, and M. Gurusamy, "Performance study of tcp flows with qos-supported openflow in data center networks," in *Networks (ICON), 2013 19th IEEE International Conference on*. IEEE, 2013, pp. 1–6.

[13] T. Wood, K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, "Toward a software-based network: integrating software defined networking and network function virtualization," *IEEE Network*, vol. 29, no. 3, pp. 36–41, 2015.

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

[15] "Ofsoftswitch13-cpqd openflow 1.3 software switch." [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[16] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "ndpi: Open-source high-speed deep packet inspection," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*. IEEE, 2014, pp. 617–622.

[17] M. P. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE, 2013, pp. 1009–1016.