APIs for QoS configuration in Software Defined Networks

Caba, Cosmin Marius; Soler, José

Published in:
Proceedings of IEEE NetSoft 2015

Link to article, DOI:
10.1109/NETSOFT.2015.7116157

Publication date:
2015

Document Version
Early version, also known as pre-print

APIs for QoS configuration in Software Defined Networks

Cosmin Caba, José Soler
DTU Fotonik – Networks Technology & Service Platforms Group
Lyngby, Denmark
{cosm, joss}@fotonik.dtu.dk

***Abstract*—The OpenFlow (OF) protocol is widely used in Software Defined Networking (SDN) to realize the communication between the controller and forwarding devices. OF allows great flexibility in managing traffic flows. However, OF alone is not enough to build more complex SDN services that require complete control and management of the data plane (e.g. configurations of ports, queues, etc.). The current work contributes to the SDN ecosystem with the implementation of a plugin for the OVSDB protocol, for an existing SDN controller (SDNC). OVSDB complements OF with management functionality such as configuration of devices, ports, queues, etc. An Application Programming Interface (API) for dynamic configuration of QoS resources in the network devices is implemented herein, by using the capabilities of OVSDB. Further, the paper demonstrates the possibility to create network services with coarse granularity on top of the fine granular services exposed by the QoS configuration API at the SDNC. A series of tests emphasize the capabilities and the performance of the implemented QoS configuration API.**

Keywords—Software Defined Networking; OpenFlow; QoS; network management; Open vSwitch; application programming interface

I. Introduction

Software Defined Networking (SDN) is a disruptive paradigm which proposes decoupling the control and data planes inside the network devices. The control plane decision logic is located in a centralized entity termed *SDN Controller* (SDNC), while the network devices become simple forwarding elements. The communication between the SDNC and the forwarding elements in the network is realized through a standardized interface, the Data-Controller Plane Interface (D-CPI), while the communication between the SDNC and the applications is realized through the Application-Controller Plane Interface (A-CPI) [1]. The D-CPI enables programmability of the entire network from the SDNC.

The programmability introduced by SDN facilitates the automation of several operations (e.g. resource provisioning, traffic optimization, etc.), thus simplifying the management of the network [2]. The capabilities offered by the SDNC at the A-CPI have a direct impact on the network services and applications built on top of it, and on the extent to which the management functions can be automated. To this end, the A-CPI must provide a wide range of functionality. Nevertheless, the functionality exposed at the A-CPI depends on the protocols implemented on the D-CPI and the features supported by the forwarding elements.

There are several standardized protocols that have QoS capabilities, available at the D-CPI (e.g. OpenFlow, NETCONF) [3] [4]. However, there is no standardized API at the A-CPI to include QoS features. This is mainly because the applications QoS requirements vary greatly, making it difficult to achieve a unique, standardized abstraction at the A-CPI to satisfy all the requirements. More specifically, some applications need a low level API for QoS control (e.g. traffic engineering application that needs access to device specific QoS configuration), while others need a higher level API (e.g. multimedia applications demanding connectivity between two endpoints).

This paper proposes an API for QoS configuration (named *QoS Config API* herein) that allows applications to configure priority queues on the ports of data plane devices. In order to support the operations exposed by the QoS Config API, the OVSDB protocol has been added at the D-CPI of an existing SDNC [5]. Additionally, the QoS Config API allows flexible configuration of priority queues on individual device ports, on every port of a device, or on all the ports in the network at once. Furthermore, the paper presents an application that creates a higher level network service over the low level QoS Config API. This emphasizes the need to have APIs with various granularities at the A-CPI in order to support applications with different requirements.

Since the OVSDB management protocol is defined for Open vSwitch (OVS) software switches [6], the QoS Config API is mainly targeted towards SDN deployments that are based on OVS. Therefore, the QoS Config API proposed herein allows the services and applications built on top of the SDNC to make use of the full set of QoS features available in OVS devices. This opens the possibility for creating innovative services and applications that need QoS support from the data plane, thus contributing to advancing the SDN ecosystem. The proposed QoS Config API is demonstrated and tested, and the results show the capabilities and the performance of the API.

The remaining of the paper is structured as follows: section II discusses other research works in the area of QoS and SDN. Section III describes the architecture of the implemented system and section IV continues with details about the module for QoS configuration. Section V presents the test setup and the obtained results. Section VI concludes the paper.

II. Related Work

Several research works have discussed the topic of QoS in the context of SDN. In [7], the SDN paradigm is applied to provide bandwidth on demand for inter-datacenter networks.

978-1-4799-7899-1/15/$31.00 ©2015 IEEE

The bandwidth allocation fairness among different traffic classes and the increase in network utilization are evaluated. The results show significant improvements in network throughput, over traditional technologies such as MPLS-TE. The classification of traffic in the data plane is based on priority queueing [7].

Another area in SDN which has received significant attention comprises the APIs exposed by the SDNCs through the A-CPI, towards external services and applications. There is little work describing APIs for QoS configuration similar to the ones proposed herein. The SDNC described in [8] offers a QoS API that is based on the capabilities of OpenFlow (OF). However, there is no detailed description of the abstractions provided through the API. The SDN-based QoS frameworks described in [9] and [10] focus on high level QoS-related policy enforcement into the data plane using OF. The policies are targeted towards network administrators or applications.

While the QoS mechanism presented in [11] is similar to the one proposed herein, the difference lies in the focus of the paper. The work described in the current paper is focused more on defining the right granularity of the APIs for QoS configuration rather than describing the internals of the implementation as in [11]. Moreover, section V of the current paper discusses some of the challenges of this work such as the state distribution among different layers of the SDN architecture, and its impact on the performance and the implementation complexity.

Overall, the proposed QoS Config API allows external services to access the QoS configuration capabilities built into the SDNC. Hence, the API proposed in this paper is not targeted towards applications that require network resources with specific QoS characteristics (e.g. multimedia applications), but towards services that need access to data plane QoS capabilities (e.g. priority queue configuration). Such a service could, for instance, consume the QoS Config API in order to provide a coarser granularity service (e.g. bandwidth on demand) to higher layer applications (e.g. database replication).

The initial version of OF (i.e. v1.0) [3] offers a limited set of operations to control the QoS in the data plane devices. More specifically, it offers the possibility to instruct the forwarding devices to forward a traffic flow on a certain output queue. However, the configuration for the queues must be conveyed to the devices through another management protocol such as NETCONF [4], etc. Recent versions of OF (i.e. v1.3.0) offer more capabilities for QoS configuration. More specifically, by using the *Meter* table it is possible to rate limit traffic flows, or to mark them with a specific Differentiated Services Code Point (DSCP) value [12].

III. ARCHITECTURE

Figure 1 illustrates the architecture of the proposed SDNC platform. The data plane consists of OVS software switches, which are deployed in physical servers running Linux. Several OVS instances run in a single physical machine. The configuration information regarding the state of each OVS switch is fetched from a database residing in the same machine as the OVS switches. The configuration can be remotely installed in the database using the OVSDB protocol, allowing external entities to control the state of each OVS switch. The state which is of interest herein is the configuration of priority queues on the output ports of the switches.

The proposed platform is based on Floodlight [13]. The protocol originally supported by Floodlight on the D-CPI is OF v1.0. For the current paper, the OVSDB protocol has been added to Floodlight, to extend it with capabilities beyond those offered by OF. The figure shows only the internal SDNC modules that are relevant for the work described in this paper. The modules depicted in grey are re-used from Floodlight while the other two modules are newly added.

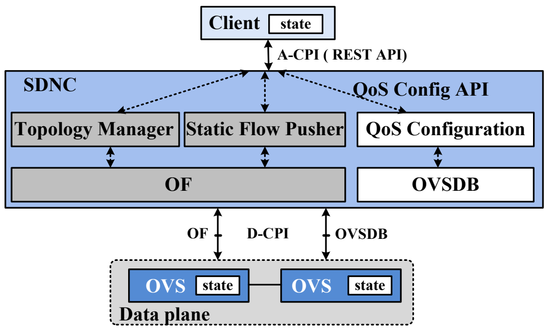

Fig. 1. Architecture of the SDNC

The *OF* and *OVSDB* modules realize the communication on the D-CPI using the OF and OVSDB protocols. The implementation of the OVSDB module contains parts of the open source module with the same name from Open Daylight [14]. The *Topology Manager* module maintains a representation of the underlying network topology and offers a REST API for querying the state of the topology (e.g. switches, links, etc.). The *Static Flow Pusher* module uses the primitives exposed by the OF module to offer capabilities for installing OF entries using the REST API.

The core of the work described in this paper consists of the *QoS Configuration* module (QoS Config). This module provides a set of methods for managing the state of the OVS instances with respect to QoS. It uses the subset of the features offered by the *OVSDB* module which are related to QoS (i.e. priority queue configuration). The services provided by the *QoS Config* module can be consumed as Java APIs by internal SDNC modules or as REST APIs by external entities (i.e. services and applications). The API is termed QoS Config API throughout the paper.

The *Client* application (Figure 1) demonstrates the possibility to create higher level network services on top of the QoS Config API exposed at the A-CPI. The *Client* may be implemented in a stateful way, by querying the SDNC through the REST API and keeping information about the network resources (e.g. links, ports, etc.).

IV. QOS CONFIGURATION MODULE

The purpose of the *QoS Config* module is to abstract the low level primitives for configuring QoS on the output ports of the switches. Hence, it translates the OVS data model to a simpler model available through the QoS Config API.

A. Data plane abstraction

The left side of Figure 2 illustrates the object model of an OVS switch (also termed Node) [15]. The model is stored as tables in the OVS database, and it is accessed by the SDNC (thus by the *QoS Config* module) through OVSDB. On each port of an OVS switch, a QoS object can be defined. The QoS object specifies the maximum rate that can be shared among the priority queues configured on that port. In the current implementation, this rate is configured to be the capacity of the outgoing link. Several priority queues can be attached on each QoS object. As described in [15], the possible parameters for a queue in the OVS database are: the minimum guaranteed rate, the maximum rate, the permitted burst size, the priority of the queue, and the DSCP value. The priority field indicates how the available bandwidth is shared among the priority queues. If configured, the DSCP value is inserted into all the packets transmitted through the queue. Complex data plane QoS mechanisms (e.g. DiffServ) can be implemented through certain configurations of the aforementioned parameters [16].

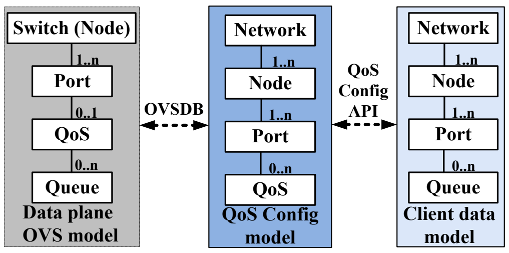

Fig. 2. QoS abstractions realized through the QoS Config module

On the right side of Figure 2, the object model available at the client, through the QoS Config API, is illustrated. The QoS Config API enables management of priority queues for three topological elements: network, node, and port. The *QoS Config* module maps the data model available to the clients to the OVS data plane model by keeping state about the topology and QoS objects configured on each port (middle of Figure 2).

B. State distribution among different layers

The *QoS Config* module does not keep state about the queues configured in the data plane devices (Figure 2). Instead, it keeps a mapping between each port of a node and the QoS configured on that port. Using this mapping, the *QoS Config* module can access the queues in the OVS model stored in the OVS database. The decision to keep only the state about QoS objects in the *QoS Config* module is a compromise between simplicity of the implementation and performance of the QoS Config API.

Keeping the state about the queues in the data plane alleviates some of the challenges related to data consistency. The configuration of the QoS objects inside the OVS model does not change over time. However, for a dynamic data plane QoS mechanism, the configuration of the queues changes frequently. If the *QoS Config* module would keep also the state of the queues, then complex mechanisms would be needed to maintain the state consistent with the OVS model, due to the frequent changes. Nevertheless, the decision impacts the performance of the QoS Config API since the data plane must be queried several times to access the configuration data for the queues. Finally, since the *QoS Config* module keeps only a limited amount of state, it becomes easy to extend the QoS Config API with new parameters that may be supported by other data plane devices, without impacting the performance of the API.

C. Operations available through the QoS Config API

The methods exposed by the *QoS Config* module (as Java and REST APIs) are shown in Figure 3. There are three levels of granularity for configuring priority queues: port, node, and network (also illustrated on the right side of Figure 2). The node and port (i.e. *NodePort* in Figure 3) association uniquely identifies a port on a given node. For each type of topological element, there are four methods defined: *get*, *add*, *modify* and *remove*. Each method takes as arguments the identifiers of the topological elements they refer to: a node is identified by its Datapath Identifier (DPID), a port by its port number, and a queue by its queue number. The *add* and *modify* methods take as argument also a queue configuration as shown in Figure 3 (right side). The API does not enforce additional constraints apart from those imposed by OVS [15]. The semantics of the four types of methods are as follows:

1) *Get:* returns all the queues configured on an element.

2) *Add:* a new queue is added on all the ports contained by the topological element the method refers to. For a port, a single queue is added on that port. All the newly added queues have the same configuration, which is passed as a parameter to the method. The reply contains the numeric identifiers (i.e. queue numbers) of the new queues.

3) *Modify:*all the queues with a certain number, which are configured on a certain topological element, are modified to a new configuration (passed as argument to the method).

4) *Remove:*removes the identified queues.

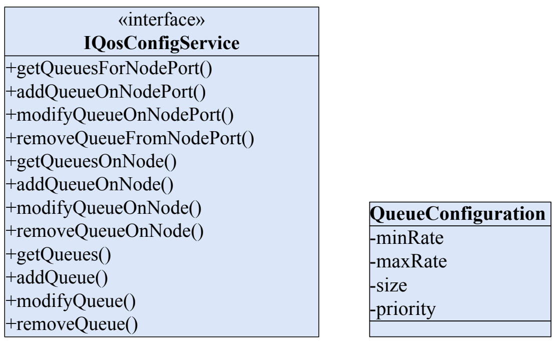

Fig. 3. Interface for the QoS Config API and class for queue configuration

To guarantee the application level QoS it is necessary to augment the QoS Config API with additional functionality. For example, a service for bandwidth on demand (BoD) may be implemented by leveraging the QoS Config API. The BoD service would then support applications QoS requirements.

V. Test and Results

The implementation has been tested in different configurations to emphasize the capabilities and the performance of the QoS Config API. The *Client* application is an example of an extension to the QoS Config REST API. This application keeps state about the network topology and the queues configured in the network, and by using this state information, it realizes higher level abstractions for QoS-aware services: it creates QoS slices (Figure 4). A QoS slice is defined by a set of priority queues that are configured on each port in the network. Consequently, a slice has an amount of bandwidth allocated to it, given by the rate with which the packets are transmitted out of each priority queue. The slices are created through the QoS Config REST API by configuring the priority queues in the data plane forwarding devices. Hence, the services offered by the *Client* application are QoS slices. Test traffic (TCP flows) is generated within the QoS slices to illustrate the behavior of this service.

OVSDB provides the functionality for port and queue configuration in the network devices. The *QoS Config* module abstracts this low level functionality, and offers fine granular services for configuring QoS on various network elements (i.e. network, nodes, and ports in Figure 4). Further, the client adds another abstraction layer for the network resources by implementing constructs that better reflect certain service logic such as QoS slices (Figure 4).

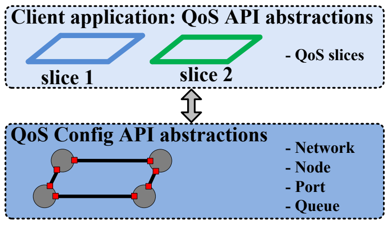
Client application: QoS API abstractions
- QoS slices
slice 1
slice 2
QoS Config API abstractions
- Network
- Node
- Port
- Queue

Fig. 4. Abstractions implemeted by the Client Application on top of the QoS Config API

The tests have been performed on a distributed testbed, which is an extension of the Distributed OpenFlow Testbed (DOT) framework [17]. Figure 5 shows a logical view of the testbed used herein, which consists of two physical servers: server 1 and server 2. Server 1 contains two Virtual Machines (VMs): (1) VM1 runs the *Client* application, and (2) VM2 runs the SDNC. A second purpose for the client is to manage the test lifecycle, define and execute the tests, by coordinating the traffic generators (Figure 5). Server 2 contains the virtualized test network comprising the OVS switches (sw1 - sw4), and a set of VMs (h1 - h3). The VMs that are attached to the switches contain the traffic generators that are remotely instructed by the *Client* application regarding the traffic flow generation.

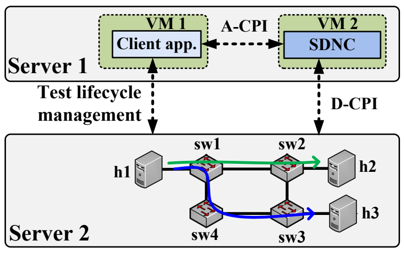

Fig. 5. Test setup

Two tests have been performed to demonstrate the QoS Config API. The first test focuses on demonstrating the capabilities of the API by managing the bandwidth allocated to different flows in real time, and the second test focuses on assessing the performance of the API.

A. Test 1

For the first test, two QoS slices are created by the *Client* application using the QoS Config REST API. Slice 1 has been allocated 7 Mbps bandwidth on all the ports, and Slice 2 has been given 3 Mbps. As illustrated in Figure 5, two traffic flows are generated: (1) the green flow (from h1 to h2) is forwarded within Slice 1, and (2) the blue flow (from h1 to h3) is forwarded within Slice 2.

Figure 6 shows the measured throughput during the test, for three of the links in the network. The duration of the test is approximately 100 seconds. As it can be observed in the figure, the measured throughput complies with the bandwidth allocation for the first half of the test. The green flow, routed within Slice 1 and forwarded on the link sw1 - sw2 (black curve in Figure 6) has a throughput of approximately 7 Mbps. Alternatively, the blue flow forwarded over the links sw1 - sw4 and sw4 - sw3, has a throughput of approximately 3 Mbps.

After the first 50 seconds (phase 1), the bandwidth allocation for the slices is changed such that Slice 1 is reduced to 2 Mbps and Slice 2 is increased to 5 Mbps. This change is performed through the QoS Config REST API, by modifying the characteristics of the rate limiting queues that belong to Slice 1, which are configured on all the ports in the network. The change in bandwidth allocation is marked by a short transition period (Figure 6), which is also influenced by the latency in the monitoring mechanism. The second phase, following after the transition period (Figure 6), shows that the measured throughput continues to comply with the newly configured bandwidth allocations.

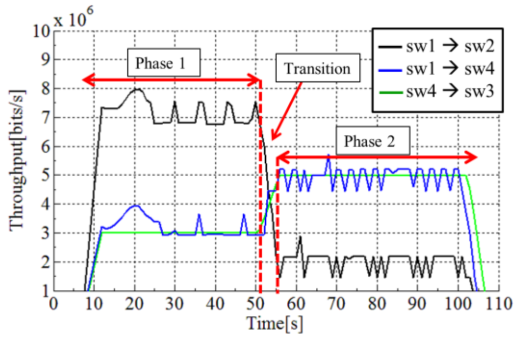

Fig. 6. Results for test 1

The results for the first test confirm that the implemented QoS Config API offers the possibility for dynamically controlling the bandwidth allocated to different traffic flows in the data plane, by using rate limiting queues. Moreover, the *Client* application implemented for this test demonstrates the possib11ility of creating high level abstractions (i.e. QoS slices) from the APIs exposed by the SDNC.

B. Test 2

The purpose of the second test is to assess the performance of the operations offered by the QoS Config API with respect to the response time. The methods assessed here are the first four in Figure 3. The methodology used for the test is to execute requests through the QoS Config Java API, and measure the round trip time (RTT) for the requests. The RTT comprises the processing inside the SDNC (both ways) and the RTT for the communication with the data plane through OVSDB. To identify the contribution of the communication with the data plane, the RTT spent on the OVSDB channel (including the processing in the switch) is measured also separately. 500 executions have been performed in order to have a statistical proof.

The test results, depicted in Figure 7, show that it takes on average 7 ms to execute one Add, Modify, or Remove request, whereas a Get request takes approximately 14 ms. For the Get request there is significantly more data about the queues carried over the OVSDB channel (thus the RTT is higher on average), and there is also more processing of this data within the SDNC. The average RTT for the OVSDB channel for the duration of the test, for all request types, is 2 ms (Figure 7).

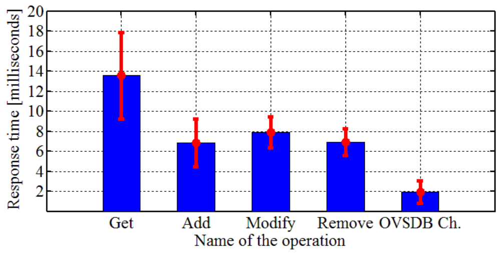

Fig. 7. QoS Config REST API response times (with standard deviation)

The results show that the current performance of the QoS Config API is good enough to enable dynamic configuration of QoS in the forwarding devices. For scenarios where the SDNC installs flow entries in a reactive manner, it is possible to perform QoS configurations with a small impact on the flow setup delay. For example, adding a rate limiting queue for a new flow delays the set up process with only 7 ms.

While the RTT for the OVSDB channel cannot be readily improved, it is possible to reduce the overall number of OVSDB requests. This can be achieved by multiplexing several operations in the same OVSDB message, and it is especially useful when several queue configurations have to be performed simultaneously. Overall, the API proposed here performs better that other reported similar APIs [11].

VI. Conclusion

This paper describes the implementation of a fine granular API for configuring QoS (using rate limiting queues) on OVS switches, using the OVSDB protocol. Additionally, higher level network abstractions (with coarser granularity) are created on top of the fine granular QoS Config API. The QoS Config API opens the possibility for creating innovative services that bridge the gap between the high level application demands and low level capabilities exposed by the controller at the A-CPI. Such services can be implemented as internal controller modules using the Java version of the QoS Config API, or as external entities using the QoS Config REST API.

The SDNC platform presented herein, together with an OVS-based virtualized network environment, enables researchers to make use of more advanced QoS features, beyond the features offered through OF. Moreover, the OVSDB module can be used for other configuration and management operations (e.g. tunnel configuration, link aggregation, etc.).

The test results demonstrate the capabilities, and give an indication about the performance of the QoS Config API. For future work, the SDNC will be extended with higher level abstractions based on the QoS Config API. These will provide integration with QoS-aware applications such as VoIP clients.

The implemented SDNC platform will be available to the open source community. The intent is to allow other researchers and developers to experiment with the QoS features available in OVS, through the QoS Config API, thus to contribute to the SDN ecosystem.

References

[1] Open Networking Foundation, "SDN Architecture", issue 1, June 2014.

[2] M. Boucadair, C. Jacquenet, "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, March 2014.

[3] Open Networking Foundation, "OpenFlow Switch Specification, version 1.0.0", 2009.

[4] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, "Network Configuration Protocol (NETCONF)", IETF RFC 6241, June 2011.

[5] B. Pfaff, B. Davie, "The Open vSwitch Database Management Protocol", IETF RFC 7047, December 2013.

[6] http://openvswitch.org/, accessed November 2014.

[7] C.Y. Hong, et al., "Achieving High utilization with software-driven WAN", Proc. ACM SIGCOMM, Hong Kong, China, pp. 15-26, 2013.

[8] K. Wonho, et al., "Automated and Scalable QoS Control for Network Convergence", Internet Network Management conference on Research on Enterprise Networking, 2010.

[9] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for Software Defined Networks", IEEE SDN for Future Networks and Services, Trento, Italy, pp. 1-7, November 2013.

[10] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, S. Krishnamurthi, "Participatory networking: an API for application control of SDNs", ACM SIGCOMM Computer Communication Review, 43, (4), pp. 327-338, 2013.

[11] D. Palma, et al., "The QueuePusher: Enabling Queue Management in OpenFlow", Third European Workshop on Software-Defined Network, September 2014.

[12] Open Networking Foundation, "OpenFlow Switch Specification, version 1.3.0", 2012.

[13] http://www.projectfloodlight.org/floodlight/, accessed November 2014.

[14] https://wiki.opendaylight.org/view/OVSDB_Integration:Main, accessed November 2014.

[15] Open vSwitch manual, http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf, accessed November 2014.

[16] A.V. Akella, K. Xiong, "Quality of Service (QoS)-Guaranteed Network Resource Allocation via Software Defined Networking (SDN)", IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC), pp. 7-13, August 2014.

[17] A. R. Roy, M. F.l Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "Design and management of DOT: A Distributed OpenFlow Testbed", IEEE Network Operations and Management Symposium (NOMS), pp. 1-9, May 2014.