Dynamic Security Traversal in OpenFlow Networks with QoS Guarantee

Yu-Jia Chen, Feng-Yi Lin and Li-Chun Wang
Department of Electrical and Computer Engineering
National Chiao Tung University, Taiwan

***Abstract*—To provide the essential network protection, data flows are forced to pass through the desired sequences of security devices (middleboxes) in a datacenter, which is called security traversal. In this paper, we identify the problem of quality of service (QoS) guarantee in security traversal and propose a scheme to dynamically change the security traversal path. The dynamic security traversal scheme is achieved by the proposed optimal security traversal with middlebox addition (OSTMA) mechanism which poses and solves the considered QoS-guaranteed problem as a constrained shortest path problem (CSP). Besides, we design a system framework with a centralized security traversal controller adopting the proposed OSTMA mechanism, which leverage the capability of OpenFlow networks to dynamically monitor the network condition information and reconfigure the security traversal path. Finally, our experiment shows results show that the proposed dynamic security traversal scheme can achieve QoS requirements while still reserving the scalability and flexibility of distributed security middleboxes.**

***Keywords*-Cloud datacenter, Security traversal, OpenFlow networks**

I. Introduction

Cloud datacenter services have been increasing explosively in recent years. To provide the essential network protection in cloud datacenters, traditional security devices, such as firewall, intrusion detection system (IDS), intrusion prevention system (IPS), will be placed at a few choke points (e.g., core routers), and rely on routing to the desired security deployment to enforce security inspection [1]. For the purpose of extending the flexibility of security management, rule-based forwarding is proposed for user-defined security requirements [2]. That is, if a data flow comes from the untrusted source, then it must be traversed to certain security devices for security inspection according to user requirements. A user can further select the path or specify which security device that his/her packets should traverse. The datacenter provider thus forces the data flow through the desired sequences of security devices. Such process is called security traversal. For example, a receiver `R` defines that the data flow comes from sender `S` needs to be checked by the firewall and the IDS sequentially. After the data flow sent by sender `S`, it must go through the firewall and then the IDS. If the data flow is passed, then it is delivered to receiver `R`.

Security traversal issue in datacenter has been investigated in the literature. The concept of software middlebox, which is implemented by a distributed virtual machine (VM) dedicated to one or a set of security functions, is proposed to improve the scalability of security service in [3]. These software middleboxes can provide more dynamic flexibility for security traversal. However, since the security traversal path which is determined by these distributed security middleboxes is only based on nearby middleboxes and network condition information, the delay of such path cannot ensure the quality of service (QoS) guarantee all the time. Specifically, when unexpected burst requests of security inspection or security middlebox collapse occurs, the security traversal path cannot be dynamically changed to another path by these distributed security middleboxes, causing severe security service delay.

To overcome the problem of QoS guarantee in security traversal, we propose a dynamic security traversal scheme in software defined networking (SDN) environment. SDN is a networking paradigm where a logically centralized controller manages elements and behaviors of the entire network. SDN introduces new possibilities for experimental network to be easily implemented and tested via the the OpenFlow protocols [4] which is an open standard of SDN. The OpenFlow network consists of a centralized controller and OpenFlow switches for decision making and packet forwarding, respectively. The OpenFlow controller can monitor the network topology and the network traffic information by manipulating OpenFlow switches. Although SDN has been shown to provide the advantage of dynamic network traffic monitoring and flexible data flow rerouting, SDN has received no or little attention to providing QoS guarantee for security traversal. Thus, in this paper, we develop an optimal security traversal with middlebox addition (OSTMA) mechanism and a system framework with a centralized security traversal controller adopting the proposed OSTMA mechanism, which leverage the capability of OpenFlow networks to dynamically monitor the network condition information and reconfigure the security traversal path. The main contributions and novelties of this work are three-fold:

- We model the considered security traversal scenario as a constrained shortest path problem (CSP) and propose OSTMA mechanism to ensure QoS guarantee. Since the data flow is inspected by security middleboxes, the delay performance is dominated by the service loading of in-path middleboxes. Traditional

Corresponding author:allan920693@yahoo.com.tw
DOI:10.6159/IJSE.2014.(4-2).62

network delay measurement such as hop-count and timestamp are unsuitable for the considered security traversal. Thus, we propose to apply queueing theory combing network congestion measurement to evaluate the delay performance of security traversal path.

- We design a system framework adopting OSTMA mechanism and a corresponding process for OpenFlow networks. In this framework, we develop a middlebox monitor module in the OpenFlow controller by the open source libvirt API [5]. Moreover, we realize the proposed OSTMA mechanism with a security traversal engine in the OpenFlow controller, which leverages the capability of the OpenFlow controller to dynamically monitor the network condition information and reconfigure the security traversal path.
- Experiment results based on real and virtualized OpenFlow network environment show that the proposed dynamic security traversal scheme can achieve QoS requirements while still reserving the scalability and flexibility of distributed security middleboxes.

The rest of the paper is organized as follows. Section II describes the background of SDN. Section III discusses the related works of security traversal. Section IV presents the proposed OSTMA mechanism. In Section V, we demonstrate the proposed system framework for OpenFlow networks. Section VI shows the experiment result of the proposed scheme. We give our concluding remarks in Section VII.

II. Software Defined Networking (SDN)

Software defined networking (SDN) is a networking paradigm where a central software controller manages networking elements and dedicates the overall network behavior. In SDN, network devices become simple packet forwarding devices, while the intelligence or the control logic is implemented in the controller [6]. That is, SDN decouples the control software from packet forwarding devices, and uses a centralized controller capable of managing network traffic for the dynamic network environments. SDN has been shown to increase the flexibility of deploying network infrastructure, the programmability of controlling network device, and the reliability of providing network service [7].

OpenFlow is an open standard that enables researchers to easily implement SDN protocols via software in experimental networks. OpenFlow identifies and exploits a common set of functions and provides the protocol for programming the flow table that indicates the processing applied to any packet of a certain data flow [8]. The OpenFlow network consists of a centralized OpenFlow controller and OpenFlow switches for decision making and packet forwarding, respectively. The OpenFlow controller sends commands to manipulate all OpenFlow switches, maintains network topology information, and monitors the overall status of entire network. The OpenFlow switch forwards incoming packets according to the flow table set by the OpenFlow controller.The OpenFlow controller makes

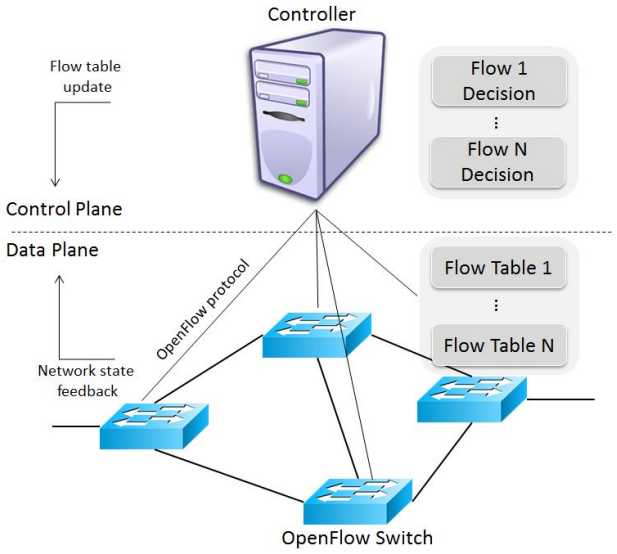

Fig. 1. The OpenFlow Architecture

the flow decisions and manipulates the overall network operation by updating the flow table in the OpenFlow switch as shown in Fig. 1. The flow table is a database that records flow entries, associated with commands from the OpenFlow controller to apply an action on a certain flow [9]. The flow entries in the flow table can be added and removed based on the OpenFlow protocol.

III. Related Work

There are many researchers focus on supporting security traversal in datacenters. Policy-aware switching proposes to integrate a number of security functions into a pswitch that is deployed in datacenters [10]. The pswitch maintains a rule table which is the pre-defined policy. Once the pswitch receives a packet, it forces the packet to traverse a sequence of security devices and then forward the packet to the next hop. It takes security devices off the data paths to improve flexibility and facilitate management and operation, but it has poor scalability and lacks of flexibility and resilience when traversing. A hybrid security architecture that hardware security devices and software security devices coexist in a datacenters is proposed in [3]. The software security device, called software middlebox, is implemented with a virtual machine (VM) dedicated to one or a set of security functions. A lable-based forwarding mechanism, which is included in a software middlebox agent, determines the next hop based on the label encapsulated in the packet header. These software middleboxes can provide more dynamic scalability for security services. The author of [11] shows that distributed software middlebox causes inefficient manipulation and proposes a centralized, unified control plane for managing data flow through distributed middleboxes in security traversal.

However, previous works do not consider the QoS issues for security traversal. Since these distributed software middleboxes can only determine the security traversal path based on the local network information, unexpected

TABLE I
COMPARISON OF THE PREVIOUS WORKS AND OUR PROPOSED SCHEME IN SECURITY TRAVERSAL

	Middlebox deployment	Scalability	Flexibility	QoS guarantee	Mechanism
Pswitch [10]	Centralized hardware	Low	Low	No	Policy-aware switch with centralized middleboxes
HSA [3]	Distributed software	High	Medium	No	Label-based traversal with distributed agents
Ours	Distributed software	High	High	Yes	Dynamic QoS traversal with a centralized controller

burst data flow or collapsed middleboxes may cause severe service delay. Thus, we propose to dynamically select optimal security traversal path to ensure QoS with centralized controller. Complete comparison between previous works and this work is presented in Table I.

IV. Optimal Security Traversal with Middlebox Addition (OSTMA) Mechanism

In this section, we model the considered security traversal scenario as a constrained shortest path problem (CSP) and show the proposed OSTMA mechanism to ensure QoS guarantee. Since the data flow is inspected by middleboxes, the delay performance is dominated by the middlebox loading. Thus we select the cost and delay metrics according to the network conditions as well as the middlebox loading to support the QoS requirements.

In our formulation, a network is represented as a directed simple graph $G(N, L, M)$, where N is the set of nodes and L is the set of all link, so that, $Link(i, j)$ is an ordered pair, which is outgoing from node i and incoming to node j. Let R_{st} be the set of all paths from source node s to destination node t. Specifically, for any path $r \in R_{st}$, we define cost function f_C and delay function f_D measured as

$$f_C(r) = \sum_{(i,j)\in r} c_{ij} \tag{1}$$

and

$$f_D(r) = \sum_{k=1}^{m(r)} d_k \tag{2}$$

, respectively, where c_{ij} is the cost metric for the $Link(i, j)$, d_k is the delay metric for the k-th middlebox, and $m(r)$ is the number of middleboxes on the r. We select c_{ij} as the weighted sum of network congestion measure and middlebox loading measure as follows,

$$c_{ij} = (1-\beta) g_{ij} + \beta s_k, 0 \le \beta \le 1, \forall (i,j) \in L, \forall k \in M \tag{3}$$

, where g_{ij} denotes the network congestion measure for the traffic on $Link(i, j)$, s_k is middlebox loading measure of the k-th middlebox, and β is the scale factor that provides the flexibility for different network condition. The formula for g_{ij} is defined as follows,

$$g_{ij} = \begin{cases} \frac{T_{ij} - 0.7 \times B_{ij}}{T_{ij}}, & 0.7 \times B_{ij} < T_{ij} \\ 0, & 0.7 \times B_{ij} \ge T_{ij} \end{cases} \tag{4}$$

where T_{ij} is the total measured traffic amount in bps and B_{ij} is the max achievable bandwidth in bps on $Link(i, j)$. We select

$$s_k = \frac{\lambda_k}{\mu_k} \tag{5}$$

which is defined as the middlebox loading intensity. And

$$d_k = \frac{1}{(\mu_k - \lambda_k)} \tag{6}$$

is the expected waiting time for the M/M/1 queueing model, where λ_k and μ_k is the input rate and the service rate of the k-th middlebox, respectively [12]. Then, the CSP problem can be formally stated as finding

$$r^* = \arg\min_r \{ f_C(r) \,|\, r \in R_{st}, f_D(m) \le D_{\max} \} \tag{7}$$

We obtain the minimum cost security traversal path r^* satisfying a specified QoS requirement from the solution of the CSP problem stated in (7). However, the CSP problem is known to be NP-complete [13], so there are heuristic and approximation algorithms in the literature. Therefore, to find the optimal security traversal path, we apply the Lagrangian Relaxation Based Aggregated Cost (LARAC) algorithm which is a polynomial time algorithm that efficiently finds a good path without deviating from the optimal solution in $O\left([n + m \log m]^2\right)$ time [14].

Figure 2 shows the system flow of the proposed OSTMA mechanism. First, a user defines the security and QoS requirement which specify the sequences of middleboxes and the delay constraint, respectively. Secondly, the system dynamically monitors the traffic of each link and the loading of each middlebox. These information are used to evaluate the delay performance of a path. After that, the system runs the LARAC algorithm to solve the CSP problem of (7). If there is no path that satisfies the QoS requirement, a new middlebox is added to the network and then the network information is re-monitored. Finally, the forwarding policy is set by the centralized controller.

V. System Framework for OpenFlow Networks

In this section, we present the system framework of dynamic security traversal for the proposed OSTMA mechanism in OpenFlow networks. The system framework, as shown in Fig. 3, consists of four modules in the SDN controller and OpenFlow switches. The system process can be classified into three stages: network condition information collecting, security traversal decision making, and traffic flow controlling. We then describe these stages as follows.

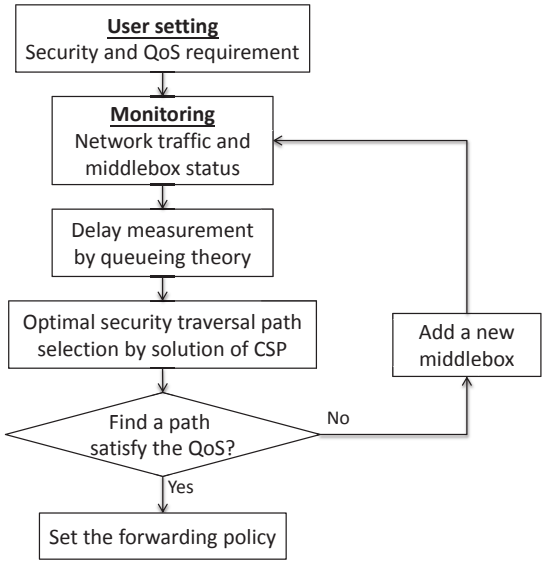

Fig. 2. System flow of Optimal Security Traversal with Middlebox Addition (OSTMA) Mechanism.

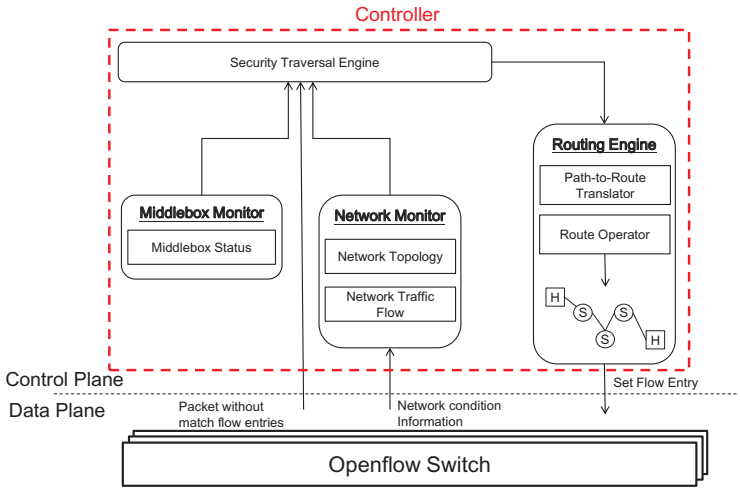

Fig. 3. OpenFlow-based dynamic security traversal framework.

TABLE II
AN EXAMPLE OF HOST-PORT-SWITCH TABLE

Host IP	Host MAC	Switch	Port number
10.0.0.1	66:d3:c2:48:e4:af	00:00:00:00:00:00:00:06	1
10.0.0.2	72:50:cf:b0:2e:42	00:00:00:00:00:00:00:07	1

1) Network Condition Information Collecting: In the first stage, the condition information of these middleboxes and the network is dynamically collected by the middlebox monitor module and the network monitor module, respectively. The middlebox monitor module records the CPU usage of the VM which serves as a middlebox. This information can be collected through the libvirt , which is an open source API that is used for managing cloud platform virtualization. The network monitor module monitors the network topology and the network traffic flow via the OpenFlow switches. We express the network topology as a host-port-switch table and a switch-link matrix. The host-port-switch table is used to record the information that end hosts connect to the OpenFlow switches. The host-port-switch table stores the IP address and the MAC address of the end host, the OpenFlow switch ID, and the port number to which the end host connects.Table II shows an example of the entries stored in this table. The switch-link matrix, which stores port mapping information of each connected OpenFlow switch, is used to record the information that the OpenFlow switches connect to each other. Table III shows an example of the entries stored in this matrix. The network traffic flow is calculated by the received and transmitted packets of each OpenFlow switch port.

TABLE III
AN EXAMPLE OF SWITCH-LINK MATRIX

	Switch 1	Switch 2	Switch 3	Switch 4
Switch 1	x	3	x	2
Switch 2	3	x	4	5
Switch 3	x	3	x	4
Switch 4	2	3	4	x

2) Security Traversal Decision Making: In this stage, the security traversal Engine is responsible for analyzing delay performance based on the network condition information collected by the previous stage and then dynamically determines the optimal security traversal path. According to the OpenFlow protocol, when a packet from a new data flow from an end host arrives at the OpenFlow switch the first time, the switch forwards the packet to the controller since there is no match with its installed flow entries. Upon receiving the packet, the security traversal engine in the controller extracts the header, and then checks the source/destination host addresses. Next, the security traversal engine evaluates the delay performance based on the network condition information. After that, the optimal security traversal path is determined by resolving a CSP problem subject to the user-defined security and QoS requirement. Finally, the security traversal engine sends the optimal path decision to the routing engine. To ensure the QoS guarantee, the engine periodically checks the delay performance of the path based on the current network condition. The engine will recalculate the optimal security traversal path once the path violates the QoS guarantee.

3) Traffic Flow Controlling: In this stage, the routing engine receives the optimal path decision from the security traversal engine, translates the logical path to the explicit network routes, and sets all the routes by updating the flow table in the OpenFlow switch. The path-to-route translator divides a path into multiple end-to-end network routes and computes the shortest route from one host/middlebox to another based on current network topology. Then the route operator converts the shortest route to flow entries and updates to each OpenFlow switch on the route.

VI. Experiment Result

We implement the proposed system framework of dynamic security traversal over a standard OpenFlow controller, Floodlight [15]. There are also several standard controller alternatives such as NOX and Beacon, but currently Floodlight is the most stable controller. Floodlight is a Java-based OpenFlow Controller and is open-source and Apache-licensed software, supported by a community of developers. It offers a systematic development architecture with abundant of application programming interfaces (APIs) which is easy to extend and enhance. The proposed OSTMA mechanism is implemented on top of the existing modules in the OpenFlow controller. The proposed mechanism is tested with a real network environment and a virtualized network environment. For the real network environment, we use an open-source software package (OpenWRT) [16] for OpenFlow v1.0 [4] with TP-Link TL-WR1043ND as the OpenFlow switch. And we set up the VMs as the end host and the middlebox. For virtualized network environment, we use Mininet [17] to emulate the entire network topology in Linux environment. The Mininet permits the specification of a network interconnecting virtualized devices including network devices, end hosts, switches and controllers. We use a Python script to create the network topology in Mininet. Both of the real and virtualized network environments are controlled by aforementioned Floodlight controller.

Figure 4 shows an experiment scenario with four security traversal paths from source to destination. We assume that the data sent by the source host needs to be checked by IDS and IPS sequentially. Upon receiving a packet in controller, the OpenFlow controller evaluates the delay performance based on the network condition and selects an optimal security traversal path from path one to path four according to the QoS requirement (e.g., 630ms). In the proposed dynamic security traversal scheme, the OpenFlow controller periodically checks the delay performance of the security traversal path every one unit time (i.e., re-traversal period=1). If the OpenFlow controller finds a QoS violation, it recalculates the security traversal path. Fig. 5 shows the delay performance of the four security traversal paths and the proposed scheme. The security traversal delay is the data transmission time from source to destination. It is observed that the proposed dynamic security traversal scheme can always ensure the QoS guarantee.

Figure 6 shows the delay performance of the four security traversal paths and the proposed scheme, similar to Fig. 5, but the OpenFlow controller periodically checks the delay performance of the security traversal path every five unit time (i.e., re-traversal period=5). It is observed that there are some QoS violations within the period of five unit time. However, it costs less path calculation and reconfiguration times. Table IV compares the QoS violation times and the path reconfiguration times for different re-traversal periods within 60 unit time. The result shows that, as the OpenFlow controller recalculates the security traversal path more frequently to ensure a more strictly QoS requirement, the more frequent path reconfiguration results a higher computing and communication overhead for the OpenFlow controller and switches. Therefore, it is crucial for a security service provider to determine a proper re-traversal period that takes the QoS requirement and the network condition into account.

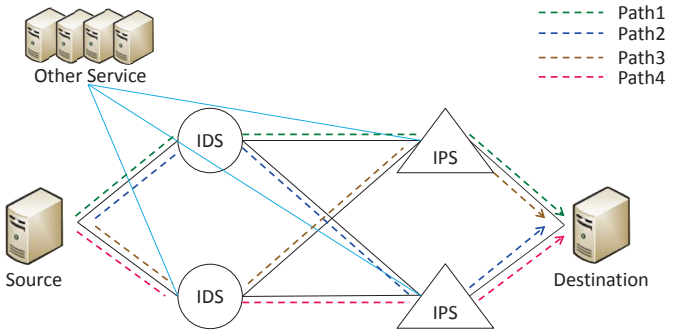

Fig. 4. Experiment scenario of security traversal paths.

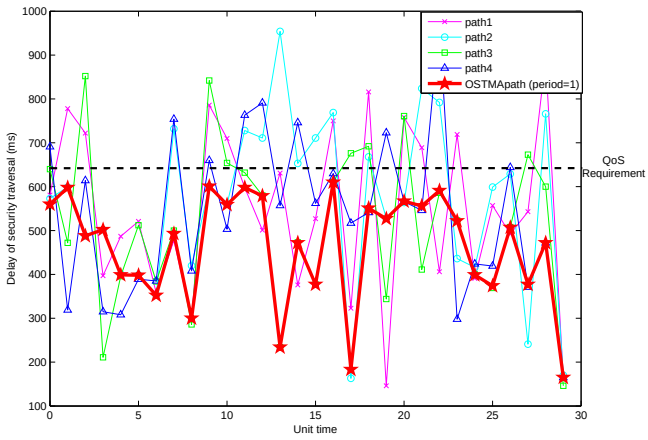

Fig. 5. Delay performance of the four security traversal paths and the proposed dynamic security traversal scheme with re-traversal period=1.

VII. Conclusiosn

In this paper, we propose a dynamic security traversal scheme for ensuring the QoS guarantee in OpenFlow networks. We identify the problem of QoS guarantee in security traversal as a CSP problem and propose an OSTMA

TABLE IV
Comparison of the QoS violation times and the path reconfiguration times for different re-traversal periods within 60 unit times.

	QoS violation times	Probability of QoS violation	Path reconfiguration times
Re-traversal period=1	0	0	22
Re-traversal period=5	15	0.25	4

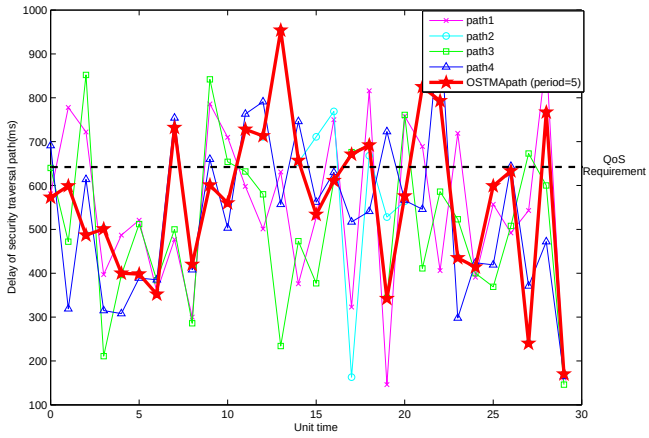

Fig. 6. Delay performance of the four security traversal paths and the proposed dynamic security traversal scheme with re-traversal period=5.

mechanism which can dynamically select the optimal security traversal path. Besides, we design a system framework adopting OSTMA mechanism and the corresponding process for OpenFlow networks. Our experiment results based on real and virtualized OpenFlow network environment show that the proposed scheme can ensure QoS guarantee. Our future work includes improving the delay performance by setting a proper scale factor β according to different network service behavior.

References

[1] "Cisco data center infrastructure 2.5 design guide," http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf.

[2] L. Popa, N. Egi, S. Ratnasamy, and I. Stoica, "Building extensible networks with rule-based forwarding (rbf)," in *In Proc. of USENIX Symposium on Operating Systems Design and Implementation*, 2010, pp. 379–392.

[3] H.-Y. Lam, S. Zhao, K. Xi, and H. J. Chao, "Hybrid security architecture for data center networks," in *IEEE International Conference on Communications (ICC)*, 2012, pp. 2939–2944.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[5] "Libvirt Team. libvirt: The virtualization api," http://libvirt.org.

[6] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[7] K. Jeong, J. Kim, and Y.-T. Kim, "Qos-aware network operating system for software defined networking with generalized openflows," in *IEEE Network Operations and Management Symposium (NOMS)*, 2010.

[8] O. M. M. Othman and K. Okamura, "Design and implementation of application based routing using openflow," in *Proceedings of the 5th International Conference on Future Internet Technologies*. ACM, 2010, pp. 60–67.

[9] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *IEEE Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2012.

[10] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, 2008, pp. 51–62.

[11] A. Gember, T. Benson, and A. Akella, "Challenges in unifying control of middlebox traversals and functionality," in *Proceedings of Large-Scale Distributed Systems and Middleware (LADIS)*, 2012.

[12] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of queueing theory*. Wiley. com, 2008.

[13] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[14] A. Juttner, B. Szviatovski, I. Mécs, and Z. Rajkó, "Lagrange relaxation based method for the qos routing problem," in *IEEE International Conference on Computer Communications (INFOCOM)*, vol. 2, 2001, pp. 859–868 vol.2.

[15] "Floodlight," http://floodlight.openflowhub.org.

[16] "OpenWrt Linux distribution for embedded devices," https://openwrt.org.

[17] 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*, 2010.