International Journal of Electronics, Electrical and Computational System
IJEECS
ISSN 2348-117X
Volume 6, Issue 7
July 2017

Academic Science

A Testbed Design of Multi-Tenant Overlay Transport Virtualization Using Open Source

Wen-Ping Lai*, Sung-Yu Wang, Yong-Hsiang Wang, Kuan-Chun Chiu
Department of Communications Engineering
Yuan Ze University, Taoyuan City, Taiwan

ABSTRACT

Large-scale network virtualization can leverage the benefits of cloud computing resources mobility and global service continuity, but it also needs to face the scalability problem with multi-tenancy, which cannot be solved by the conventional VLAN technology. This article proposes a testbed design of multi-tenant overlay transport virtualization using open source-based toolsets to realize VXLAN tunneling, which is a new overlay technology free of the scalability problem with multi-tenancy. This article also delivers justified design principles and detailed working mechanisms, including novelties such as network name space-based virtual machines (VMs) that can naturally extend to emerging container-based VM technologies such as Linux or Docker containers, and Open vSwitch-based VXLAN tunnel end points that cannot only serve for coloring (isolating) a quasi-unlimited number of multi-tenants, butalso be extensible to orchestration of containers. Our experimental results verify the VXLAN encapsulation function, confirm its multi-tenancy behavior, and observe substantial overhead effects of VXLAN tunneling in terms of latency and data throughput.

Keywords

overlay transport virtualization; multi-tenancy; VXLAN; network name space; container

1. INTRODUCTION

The wide and ever-increasing deployment of cloud-based technologies has reshaped the global application landscapes, from mature business networking using *virtual private networking* over the Internet, to innovation of daily lives based on global *data centers' interconnection*. The underlying technology is based on a fundamental concept called *overlay transport*, making remote sites be connected as if they were adjacent logically via some sort of tunneling protocol. On the other hand, the rapid development of *virtualization* for both computer and networking improves server usage in global data centers, reduces power consumption, lowers operation and maintenance cost, and increases service deployment flexibility. These benefits do not only make virtualization to be widely applied, but also poses a great challenge for *overlay transport virtualization* (OTV). The main complications of realizing OTV lie in establishing a connection tunnel across an integrated environment made of not only physical and *virtual machines* (VMs) but also physical and virtual networks.

When data centers accommodate more and more tenants, they have to face the challenge of the significantly increasing number of VMs. On a legacy Layer 2 *local area network* (LAN), *virtual local area network* (VLAN) is the dominant isolation technology for multi-tenants, each of which can be identified with a unique VLAN tag. However, the VLAN tag field is only 12 bits long and can identify 4096 VLANs at most, which is obviously not enough for millions of VMs. This is known as the *scalability problem* with VLAN. In addition, to flexibly schedule computing resources of data centers, VMs' migration across devices, areas or even data centers must be supported. Furthermore, to ensure service continuity, VMs must keep their IP and MAC addresses intact after any migration. All these requirements cannot be met by the VLAN isolation.

To face these challenges, *virtual extensible local area network* (VXLAN) [1] has been developed. It defines a 24-bits header field for VXLAN Network ID (VNI) to identify each tenant and this solves the scalability problem with VLAN: up to 16 million (equivalently quasi-unlimited) isolated Layer 2 virtual network

tenantscan be classified. As an OTV tunnel, VXLAN applies MAC-in-UDP capsulation, which encapsulates the *original* Ethernet frames (sent from a VM) into UDP packets, and then adds *outer* IP and Ethernet headers of the physical network on to the UDP packets. The encapsulated packets can thus be routed as normal IP packets over the underlay IP-routing network. With VXLAN, VMs on remote Layer 2 virtual networks can be viewed as logical neighbors, and are free to migrate across devices, areas, or even data centers while ensuring their service continuity. In addition, being an overlay technology, no VM migration involves the need to adjust the underlay network configurations. Since VXLAN decouples virtual and physical networks, tenants are able to plan their own virtual networks without the need to worry about packet delivery over the underlay physical network. In fact, only the VXLAN *tunnel end point* (TEP) *devices* need to learn and identify the MAC addresses of VMs, and this greatly reduces the number of MAC addresses to be learned, thus substantially improving the forwarding performance of the entire network. All the above features make VXLAN greatly improve the operating revenue and also simplify network management of data centers.

One of the salient advantages of VXLAN lies in that it has gained multi-vendor support (by VMware, Cisco, Arista, Cumulus Networks, Broadcom, Intel and Redhat) in a relatively short period, and this makes it an ideal choice for multi-vendor deployment and particularly suitable for *north-south* traffic, that might need to flow out of the data center to the Internet for some sort of appliance or service such as a load balancer, firewall or intrusion detection system, etc. As for *east-west* traffic (from VM to VM), *stateless transport tunneling* (STT) [2], based on MAC-in-TCP-like capsulation, is perceived as a better choice for higher efficiency by leveraging the *TCP segmentation offload* (TSO) capability equipped with today's commodity *network interface cards* (NICs), despite its lack of multi-vendor support (mainly proposed by VMware, still at its draft stage for IETF RFC). It is well expected that the performance difference between VXLAN and STT should eventually disappear, given the great momentum behind VXLAN and the trend that a new generation of NICs will also allow other tunneling encapsulations to be used without disabling the TSO capability. *Network virtualization using generic routing encapsulation* (NVGRE) [3], mainly proposed as by Microsoft, applies MAC-in-GRE encapsulation to address the scalability problem with VLAN, where the lower 24 bits of the GRE header are used to represent the *tenant network identifier* (TNI), thus also supporting for up to 16 million tenants. Two major differences come with NVGRE: the good side is that, unlike VXLAN, the support of broadcast via IP multi-cast is only optional to NVGRE, leaving it up to the handlingof a yet undefined control plane protocol and making it a more scalable solution; the bad side is that it offers little ability for multi-path load balancing, which has been solved in VXLAN by using a hash of the inner MAC frame as the UDP source port.

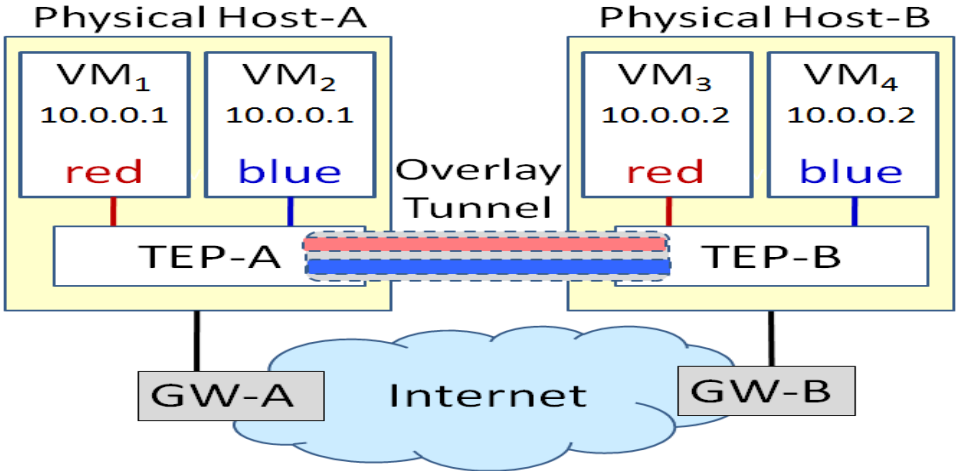

Fig 1: Overlay transport virtualization accross the Internet

To address the above problems brought up by OTV, this article designs a minimum testbed for realizing the concept of OTV, as illustrated in Fig. 1, where the realization of a VXLAN tunnel across two Open vSwitch(OVS)-based [4] TEPs is demonstrated to allow remote VMs communicate with each other via the VXLAN tunnel. In addition, multi-tenancy over the VXLAN tunnel is also demonstrated to achieve

administrative isolation of two logical virtual Ethernet segments over the VXLAN tunnel. It is worth mentioning that all the realizations above are open source-based, compared to that the current implementation practices are mostly proprietary [5]. In addition, the *network name space*-based VMs, instead of the *hypervisor*-based VMsare used, leading to potential innovations of VM services brought by new VM technologies such as Linux container (LXC) or Docker container.

The remainder of this article is organized as follows. Section 2 gives the design principles and working mechanisms of thetestbed, followed by experimental results and analyses in Section 3. Section 4 concludes the article and outlooks potential extensions of this testbed design.

2. DESIGN PRINCIPLESAND WORKING MECHANISMS

The goal of this study is to propose a minimum testbed design of realizing the concept of OTV for serving as an open source-based teaching or research platform. Fig. 1 demonstrates a typical conceptual design of OTV across the public physical underlay network such as the Internet, where the colored overlay tunnels are logically built between TEPs to represent the encapsulated physical networking for communication between remote VMs of the same tenant, represented by the same color. In order to serve as a teaching or research platform, the following testbed design principles are considered, together with their working mechanisms.

- *Using network name spaces for VMs*

In terms of VMs, *hypervisor*-based virtualization, such as VMware and VirtualBox, is a mature technology with a wide variety of commercial use such as cloud services. However, the provision process of such a *hypervisor*-based VM is long (including installation of a guest OS and its initialization time) and its operation consumes some portion of system resources inflexibly due to the fact a pre-determined allocation strategy is needed before operation. On the other hand, *container*-based virtualization, such as LXC and Docker [6, 7], is no more characteristic of *OS containment*, but featured by *application containment* with minimum associated libraries, where all the Guest (container) machines share the same OS kernel with the Host machine and thus achieve fast provision time and light weight use of system resources. Remarkably, the Linux kernel-based *name spaces* play a common trick for *isolation* of the Host machine's system resources, and *control groups* (Cgroups) for *control* and *management*. Thus, *using network name space-based VMs* becomes a good design principle for considering a further extension of this testbed design for potential innovations of VM services brought by *container*-based VM technologies such as LXC or Docker.

- *Using veth-pair-links and OVS for virtual networking*

In terms of virtual networking, several virtual wiring entities such as *virtual links* and *virtual bridges/switches* need to be considered before establishing an OTV tunnel over the public underlay network, such as the Internet. Linux provides native tools such as*virtual Ethernet pair* (veth-pair) to play as a virtual link, and a native Linux bridge as a virtual bridge or virtual switch. Note that a *veth-pair* has two virtual NICs at both the ends of the virtual link, which can be respectively inserted two *network* name space-based VMs to form *direct linking* between them, or each of such VMs can be individually linked to a virtual switch, such as Linux or OVS bridge, to form *bridged linking*. It is obvious that these wiring entities together address the problem of how to link virtual hosts in the same virtual LAN within a single physical machine, and these wiring entities also allow high design flexibility for constructing a desired virtual LAN topology. Note that the OVS is needed for establishing an OTV tunnel between two remote VMs belonging to the same VLAN, as detailed below.

- *Using VXLAN for OTV tunneling*

As just mentioned, OVS is responsible for OTV tunneling between two remote VMs of the same VLAN, because it can support both VXLAN and NVGRE. In this study, VXLAN is adopted due to the rationale argued previously: VXLAN has the most multi-vendor supports and it can catch up with STT with a promising momentum in its efficiency. As shown in Fig. 1, two OVS bridges are adopted to serve as the VXLAN TEP devices, namely TEP-A and TEP-B, which are responsible for the coloring mechanism (i.e.,

encapsulation and decapsulation) of packets from the *red* or *blue* tenant so that the two tenants are well isolated without collision of IP addressing, say the 10.0.0.1's in both the *red* and *blue* tenants.

3. EXPERIMENTAL RESULTS AND ANALYSES

To realize the proposed testbed design, a lab-based network topology and configuration for VXLAN-tunneling is setup in Fig. 2, as a practice of Fig. 1. According the proposed design principles and working mechanisms, the four VMs are all implemented by *network name spaces*, denoted as NS_1, NS_2, NS_3 and NS_4, classified into two tenants: red and blue, and the corresponding VLANs are bridged by the two OVS bridges, serving as the two TEPs of the VXLAN tunnel, denoted as TEP-A and TEP-B. Each tenant is identified with its own VLAN ID: 100 for *red*, and 200 for *blue*. Likewise, the same values are also used as VNIs, through the VXLAN tunnel's coloring mechanism. In addition, both the IP and MAC addresses of each virtual host and physical host are labeled to allow VXLAN's functional verifications, such as *encapsulation* and *multi-tenancy*. In addition, performance impact of VXLAN's encapsulation is also measured. The measurements include the header analysis of the VXLAN's overlay tunneling by the *wireshark* toolset, as well as latency and throughput tests by the *ping* and *iperf* toolsets respectively. To simplify the realization, the underlay network is a physical LAN, say 10.0.0.0., to mimic the Internet without losing generality.

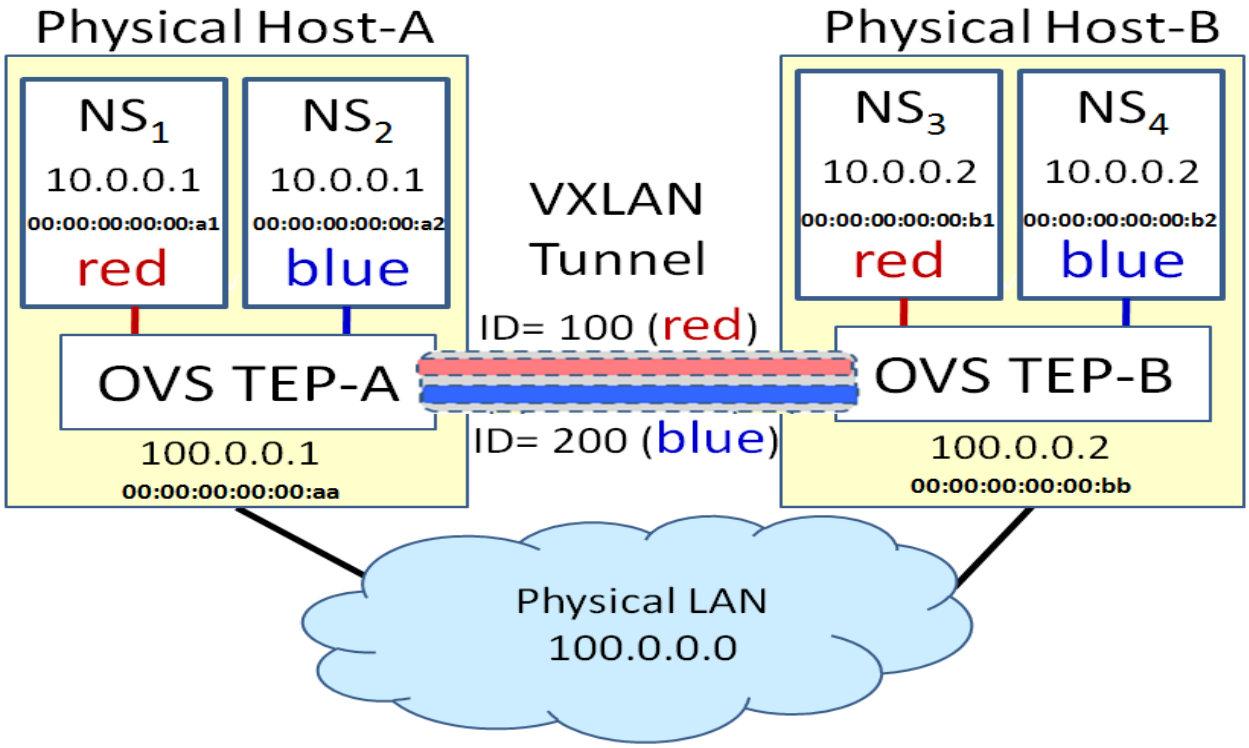

Fig 2: Realization of VXLAN-based OTV across a physical LAN environment

- *Header analysis of the VXLAN's overlay tunneling*

The header analysis of the VXLAN's overlay tunneling effect is presented in Fig. 3, where the overall stack structure of involved protocols in a Ethernet frame (from outside in) are: Ethernet | IP | UDP | VXLAN | Ethernet | IP | ICMP | data, for an ICMP-based *ping* request packet from NS_1 (IP = 10.0.0.1, MAC = 00:00:00:00:00::a1) to NS_3 (IP = 10.0.0.2, MAC = 00:00:00:00:00::b1) via the VXLAN tunnel (VNI = 100 for the *red* tenant), which is overlaid across between the two physical hosts (100.0.0.1 and 100.0.0.2) and over the physical LAN 100.0.0.0. Such a protocol structure analysis verifies the intrinsic feature of MAC-in-UDP encapsulation for VXLAN tunneling.

International Journal of Electronics, Electrical and Computational System
IJEECS
ISSN 2348-117X
Volume 6, Issue 7
July 2017

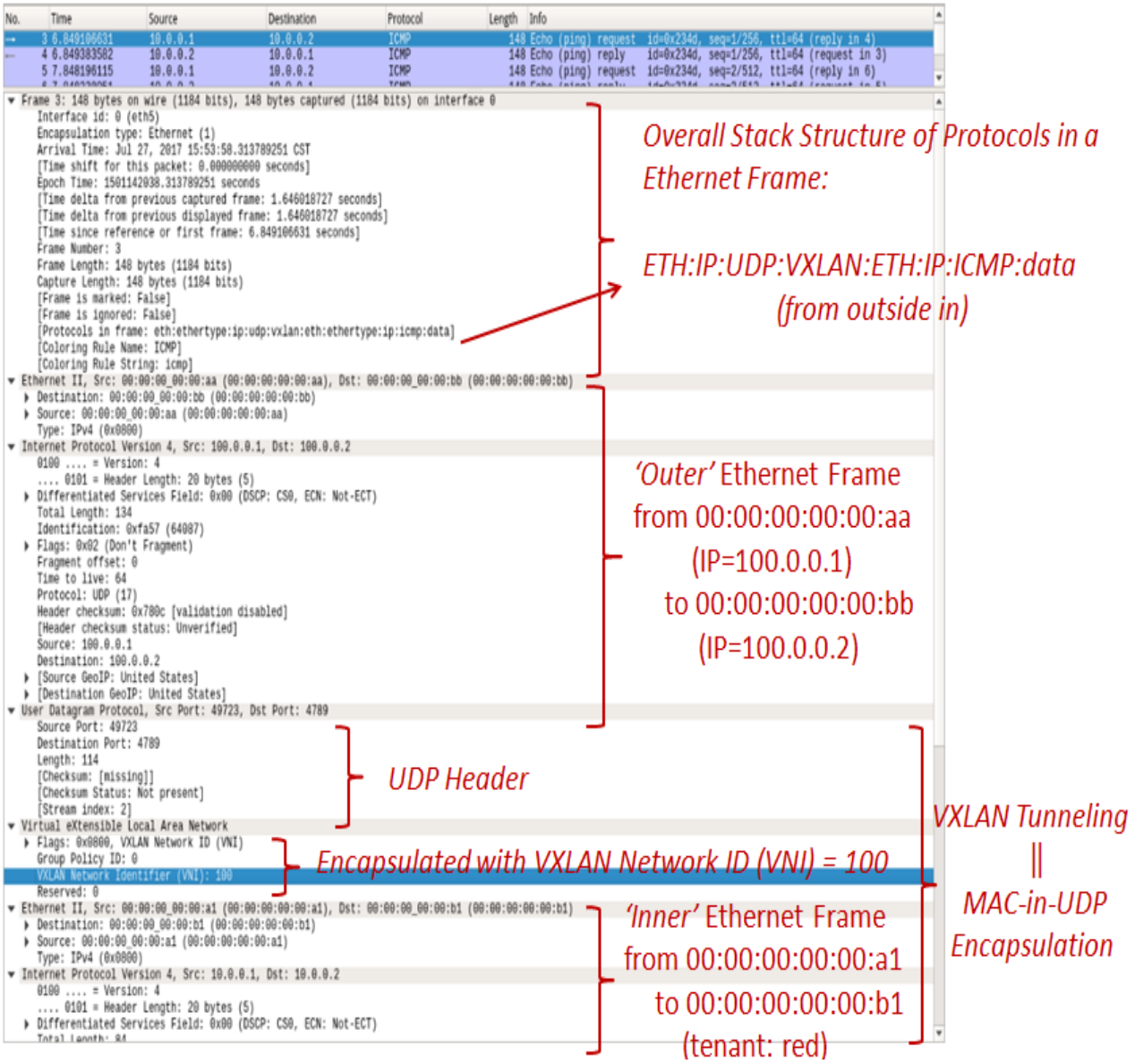

Fig 3:Wireshark's packet capture information and analysis for VXLAN tunneling

- *Multi-tenancy verification of VXLAN tunneling*

Recall that Fig. 3 is analyzed using *ping* test between NS_1 and NS_3 for tenant *red* (VNI = 100), and a similar test has also been verified between NS_2 and NS_4 for tenant *blue* (VNI = 200). As another test method with more intuition is that the *ping* test can *fail* whenever the *ping requested* VM's virtual NIC (within the same tenant) is turned off. Such a *failure effect* has been observed for both the tenants. For instance, when only the virtual NIC of NS_3 is turned off, the *ping* test between NS_1 and NS_3*fails* for tenant *red*, but the *ping* test between NS_2 and NS_4*still succeeds* for tenant *blue.* Vice versa, it fails for tenant *blue* but succeeds for tenant *red* if only the virtual NIC of NS_4 is turned off.

- *Performance Evaluation of VXLAN tunneling*

To study the performance impact of VXLAN tunneling due to its MAC-in-UDP encapsulation mechanism, Table 1 summarizes both the overhead effects due to the extra protocol headers, in terms of *round-trip*

*latency*by *ping* and *data throughput* by *iperf*. Compared to those performance tests conducted under the same network topology without VXLAN tunneling, the overhead effects are seen *substantially* both in *ICMPround-trip latency* and *TCP throughput*, based on 100 measurements for each effect, where the quoted uncertainty are standard errors with 68% confidence level, recalling that the underlay network for all the tests is a physical LAN made of Gbps-level Ethernet NICs. Averagely speaking, VXLAN increases the extra round-trip latency by 0.016 ms, and slows down the data throughput by 38 Mbps.

4. CONCLUSIONS AND OUTLOOK

In the study, an open source-based testbed design for realizing the concept of *overlay transport virtualization* using VXLAN tunneling was proposed with justified design principles and detailed working mechanisms, where *name space*-based VMs together with the Linux *veth-pair* virtual links and the OVS virtual bridging can enhance the design flexibility of this testbed for teaching or research purposes. The experimental results clearly verified the features of MAC-in-UDP encapsulation and the multi-tenancy behavior for the VXLAN overlay tunneling. Substantial overhead effects on latency and data throughput were both observed, and these are within expectation due to the fact that extra protocol headers need to be added for layingvirtual Ethernets over the physical IP transport network.

As an outlook, such a testbed design can further be enriched by newly emerged *container*-based VM technologies such as Docker, and their orchestration partner called Kubernetes [8], both invented and promoted by Google for next generation networking such as software defined networking and network function virtualization [9, 10]. In particular, micro-services overlaid by VXLAN tunneling would be very interesting for future networking innovations.

Table 1. Overhead effects due to VXLAN's encapsulation

Effect	Tests between NS_1-NS_3	
	without VXLAN	with VXLAN
Latency (ms)	0.203 ± 0.003	0.219 ± 0.003
Throughput (Mbps)	934 ± 0	896 ± 0

ACKNOWLEDGMENT

This work was partly supported by Taiwan's Ministry of Education for the advanced-technology course promotion project for mobile broadband under grant 105-2700-010.

REFERENCES

[1] Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, IETF RFC 7348, 2014.
[2] A Stateless Transport Tunneling Protocol for Network Virtualization (STT), IETF draft-davie-stt-08, 2016.
[3] NVGRE: Network Virtualization Using Generic Routing Encapsulation, IETF RFC 7637, 2015.
[4] Linux Foundation Collaboration Projects. Open vSwtich [Online]. Available: http://openvswitch.org
[5] Virtual Extensible LAN (VXLAN) Best Practices, Cisco White Paper, 2016.
[6] D. Bernstein, "Containers and cloud: from LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
[7] I. Miell and A. H. Sayers, *Docker in Practice*. New York: Manning, 2016.
[8] Kubernetes on CoreOS Container Linux Documentation[Online]. Available: https://coreos.com/kubernetes/docs/latest/
[9] D. Kreutz, P. E. Verissimo, and S. Azodolmolky, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
[10] Y. Li and M. Chen, "Software-defined network function virtualization: a survey," *IEEE Access*, vol. 3, no. 1, pp. 2542–2553, 2015.