Design of Virtual Gateway in Virtual Software Defined Networks

Doyoung Lee*, Yoonseon Han†, and James Won-Ki Hong*

*Department of Computer Science and Engineering, POSTECH, Pohang, Korea
{dylee90, jwkhong}@postech.ac.kr
†Division of IT Convergence Engineering, POSTECH, Pohang, Korea
seon054@postech.ac.kr

***Abstract*—Network virtualization is a technique that abstracts the underlying physical infrastructures into multiple isolated networks. Currently, network virtualization based on Software-Defined Networking (SDN) has attracted interests from industry and academia to utilize limited network resources by using benefits of SDN. SDN has useful features such as programmability, flexibility, and agility. In order to virtualize networks in SDN, a network hypervisor intercepts and modifies OpenFlow messages so that it provisions multiple virtual networks, virtual Software-Defined Networks (vSDNs). However, existing SDN-based network hypervisors do not provide an easy-to-use method to connect a created vSDN with external networks. It limits the usefulness of vSDNs. To resolve this problem, we propose a virtual gateway for external connectivity in vSDN. The proposed virtual gateway is implemented using ONOS virtualization subsystem. The virtual gateway is able to provide external connectivity and other useful network functions such as firewall, traffic shaping, and load-balancing. To demonstrate the feasibility of virtual gateway, we evaluate round trip time and deployment time to show a connectivity and overhead of the virtual gateway deployment.**

***Keywords*—Software Defined Networking (SDN), Network Virtualization, Network Hypervisor, Virtual Gateway, Virtual Network Management, virtual SDN (vSDN)**

I. Introduction

Network virtualization [1] is an attractive technique to flexibly and efficiently utilize limited network resources such as CPU or memory of network devices and bandwidth of each link. Over the past decade, server virtualization has been used widely in the real world to make more efficient use of server resources, physical space of servers as well as electricity [2]. For instance, a hypervisor is implemented for server virtualization. Hypervisor is a platform to monitor resources of a physical machine and allocate resources to each virtual machine operating on the physical machine. This concept makes it possible to efficiently utilize resources of a physical machine. Likewise, a network hypervisor is emerged to virtualize a physical network. With a network hypervisor, virtual networks can be created by sharing physical network infrastructures [3]. In order to satisfy virtualization requirements, Software-Defined Networking (SDN) can be a good solution due to its useful features including a global view of the network. Moreover, an SDN controller can provide effective functions to manage virtual Software-Defined Networks (vSDNs).

SDN is a new networking paradigm which decouples the control plane from the data plane. SDN provides a centralized control plane, an SDN controller, to manage the underlying forwarding network devices. An SDN controller provides a flexible and efficient interface to enable a network manager to apply network functions rapidly. On the other hand, owing to the decoupling, a communication protocol between the control plane and the data plane is required. For that reason, OpenFlow [4] is defined and used as a *de-facto* standard protocol in SDN. Moreover, an SDN controller has a global view of the network, so it is possible to monitor network resources and manage the networks dynamically. With SDN, a network hypervisor placed between the control plane and the data plane intercepts and modifies OpenFlow messages for network virtualization.

Currently, there exist SDN-based network hypervisors such as FlowVisor [5], Vertigo [6], FlowN [7], and OpenVirteX [8]. They are used to build isolated vSDNs. These network hypervisors build vSDNs based on slicing. Network slicing is a concept to imply that actions in one slice do not negatively affect other slices, even if they share the same underlying physical hardware [9]. Since each vSDN operates on its network slice, each vSDN is isolated. However, those hypervisors only focus on creating vSDN. In addition, they do not provide a method to connect vSDN with external networks, external connectivity.

Accordingly, several solutions have been emerged for external connectivity such as virtual router using OpenStack [10] and Microsoft's virtual machine manager (VMM) [11]. However, they just deploy a virtual machine as a virtual router and attach an additional interface which provides network address translation (NAT). Since their virtual machine is used for external connectivity, it is difficult to apply various network functions including traffic shaping, load-balancing, and firewall. In addition, to the best of our knowledge, there is no solution to provide external connectivity for vSDN by SDN-based network hypervisor. For that reason, we propose a concept and design of a virtual gateway. The proposed virtual gateway is deployed in vSDN, and it provides methods to handle packets which need to traverse through external

978-3-901882-98-2

networks. Moreover we provide an easy-to-use way to deploy the virtual gateway via ONOS [12]. Specifically, we leverage ONOS virtualization subsystem.

The rest of this paper is organized as follows. Section II reviews related work. Section III describes the overall architecture of the proposed virtual gateway. Section IV presents an implementation of the proposed virtual gateway. Section V shows evaluation. Finally, Section VI presents the conclusion and future work.

II. Related Work

One of main components for network virtualization in SDN is an SDN controller. An SDN controller can be a tenant controller of each virtual network, and also provide methods for network virtualization itself. For instance, OpenDayLight [13] and other similar controllers provide sub-systems to build virtual networks. However, those controllers would allow the control of virtual network slices only via the application-control plane interface (A-CPI). Thus, these SDN controllers could not communicate transparently with their virtual network slices [14]. In order to overcome this limitation, SDN-based network hypervisors have been implemented. A network hypervisor is a critical component to abstract the underlying physical infrastructures.

FlowVisor [5] is one of the oldest network hypervisors in SDN. FlowVisor defines a slice as a set of flows, one flow is called flowspace of slice, and 10 fields are defined by OpenFlow V1.0. FlowVisor distinguishes slices by setting the flowspace not to overlap among different slices. Each slice is assigned to one tenant, and tenants can manage the allocated slice separately using an OpenFlow-based controller. Thus, tenants can have their own virtual network based on slices. Beyond the FlowViosr, Vertigo [6] was implemented based on FlowVisor. Vertigo allows tenants to define virtual links within their network slice, allowing them to create topologies per tenant. While FlowVisor's network slice is represented only as a subset of the physical network, Vertigo can create a slice of all physical network subsets as well as abstract the physical network into a single big switch. However, since Vertigo is also based on FlowVisor, it has the problem of not providing completely isolated address space to tenants. FlowN [7] introduced a database management system to manage the mapping between physical and virtual networks. Using OpenFlow technology, each tenant can run network control logic directly on their controller, reducing the burden on physical switches. The mapping information between physical and virtual networks is designed to be manageable in an expandable form. By introducing a new network abstraction model, the network can be abstracted and it allows each tenant to construct an arbitrary network topology. FlowN uses a container-based network virtualization method that shares a single FlowN controller across multiple tenants rather than assigning controllers independent of each tenant. This has the disadvantage that the control logic of each tenant must be embedded in the FlowN controller. On the other hand, OpenVirteX (OVX) [8] allows to assign different OpenFlow controllers to each tenant, allowing network managers to configure and control the tenant-specific virtual network. OVX is located in the virtualization layer in the virtual network composed of the physical layer, the virtualization layer, and the network operating system layer, and performs mapping of virtual resources and physical resources with the help of network embedder. Moreover, OVX is a network virtualization platform designed to be highly scalable. OVX is capable of N:1 mapping between virtual resources and physical resources, providing flexibility of virtual network configuration.

Most network hypervisors are deployed between the control plane and the data plane as a proxy. In addition, they mainly focus on creating virtual networks by slicing. However, they do not consider a method to connect each virtual network with external networks. If a virtual network needs to connect with other networks, a general method is to deploy a virtual machine which has an additional interface connecting with external networks and then deliver packets through that machine. Although it can provide external connectivity, it is insufficient as a network solution and it has limitations to deal with all packets which traverse via the virtual machine.

III. Proposed Architecture

vSDN is deployed by creating virtual devices and links mapping to physical infrastructures. Thus, to provide external connectivity for vSDNs, firstly, a physical network should have a physical gateway connecting with external networks. After that, a virtual gateway component is created and then embedded on the physical gateway. The virtual gateway does not need to be embedded on the physical gateway directly. Instead, the virtual gateway can be embedded in one of OpenFlow switches of the physical network. After embedding the virtual gateway, a network hypervisor should install flow rules and additional actions to deliver packets to a physical gateway. Our proposed concept is presented in Fig. 1.

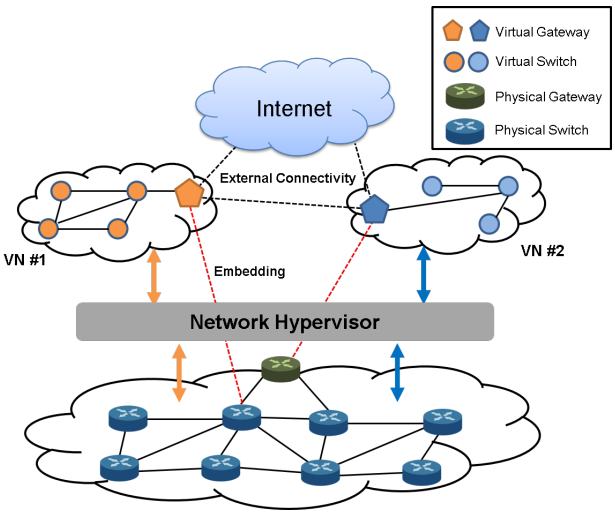

Fig. 1. Overall design

A. Requirements

To connect a virtual network with external networks, an easy-to-use way enabling the connectivity is essential. Thus, we propose and design a virtual device as a virtual gateway. The proposed virtual gateway allows vSDNs to connect with external networks. When a packet which should be delivered to or from external networks exists, the virtual gateway must deal with the packet. Since each host of virtual network usually is assigned its private IP address, the virtual gateway should provide NAT function to avoid address conflict. Otherwise, a physical network itself can provide the feature, that is, a virtual gateway can just deliver packets through the physical gateway. For that reason, a main requirement of the proposed virtual gateway is to connect with a physical gateway and apply additional functions which should be used for external connectivity. Moreover, we should guarantee that our proposed method does not negatively affect on virtualization principles. The detailed requirements are as below.

- Virtual network embedding mechanism for mapping a virtual gateway to a physical gateway
- Tunneling based on VLAN tagging
- Maintaining network virtualization principles
- NAT function to avoid address conflict
- Forwarding application for virtual network

B. SDN-based Network Hypervisor Approach

SDN-based network hypervisor is responsible for network virtualization by intercepting and modifying OpenFlow messages. Due to the feature of SDN, the network hypervisor can also have the global network view. The network hypervisor consists of multiple layer to support virtualization. First, the hypervisor includes southbound interface (SBI) and southbound message handler to handle each SDN protocol's packet. Based on OpenFlow, network hypervisor manages physical network resources such as devices, links, ports, and so on. Those resources are abstracted and provided for the upper layer as objects. Through the abstracted resource, virtualization layer can deal with them. A main role of the virtualizaion layer is to virtualize physical network in terms of three concepts including address virtualization, topology virtualization, and control plane isolation. Address virtualization enables each tenant can use independent IP address without considering other networks. Topology virtualization provides a virtual topology to each tenant. Thus, a tenant can have virtual network topology based on its requirement. Finally, control plane isolation is responsible for connecting each virtual network with only allowed tenant controller. Our virtual gateway leverages the topology virtualization to create and connect. The overall SDN-based network hypervisor design is shown in Fig. 2.

C. Virtual Gateway Design

Whenever there are packets which are to be delivered to external networks, a virtual gateway delivers them through a tunnel by VLAN tagging, and then the physical gateway forwards the packets to the external networks. Since each vSDN

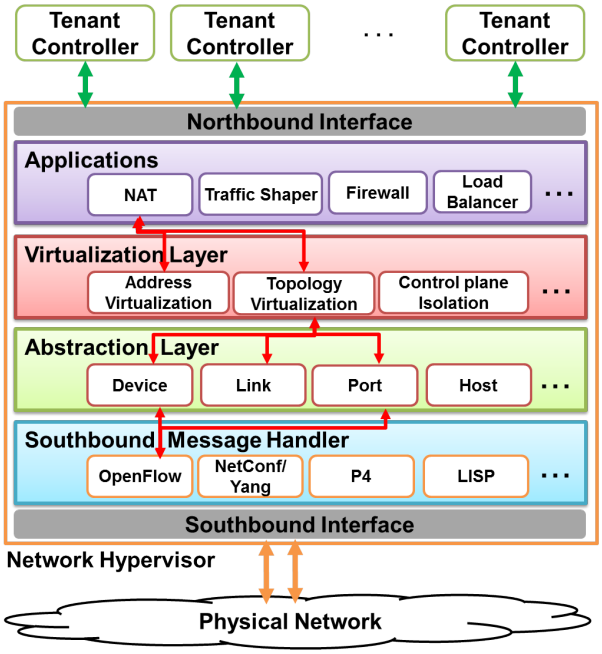

Fig. 2. Network hypervisor design

uses independently private IP address, NAT function should be provided. This is because packets of each virtual network should use public IP address when they connect with outside. With NAT, a virtual gateway translates and identifies packets and then forwards them to a destination. Moreover, a virtual gateway should provide additional functions such as traffic shaping, load balancing, and firewall to handle traffics. In order to provide the functions, a virtual gateway has components to store policies how to apply the functions. As a virtual gateway is created and configured by a network hyperviosr, these policies are also deployed by a network hypervisor. Through the functions, a virtual gateway allows each tenant to operate virtual networks satisfying their requirements. Additionally, a virtual gateway is one of virtual devices, so flow rules must be installed. Thus, it has a virtual flow rule table for virtual flow rules. Virtual flow rules should be installed by referring function policies to handle traffics properly. In order to forward packets through the virtual flow rules, virtual gateway has to translate them to physical flow rules and install to the physical infrastructures. For that reason, a virtual gateway includes a flow rule translator. Finally, flow rules with proper actions are installed in the underlying infrastructures, and it can handle traffics. A design of virtual gateway is presented in Fig. 3.

IV. Implementation

The proposed virtual gateway is an easy-to-use way enabling vSDN to connect with external networks. It is embedded automatically onto a physical gateway and handles all traffics which transverse between vSDN and external networks. In this paper, we implemented the proposed virtual gateway as a virtual device, that is a virtual switch. Virtual device has virtual ports, virtual links, and provides connectivity with hosts. To

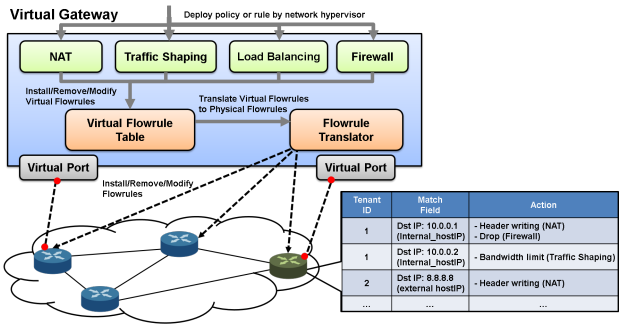

Fig. 3. Virtual gateway design

virtualize physical network, virtual devices should be mapped to physical devices, that is, virtual network embedding. An SDN-based network hypervisor is responsible for this embedding process. We exploited ONOS virtualization subsystem as a network hypervisor. The virtualization subsystem is a network framework included in ONOS. ONOS has a layered architecture, and each layer has a specific components which are application, manager, and provider. As a network hypervisor, it provides virtualization layers including virtual services and providers named *Default*Providers*, such as *Default Virtual Network Provider* and *Default Virtual Packet Provider* [15]. Since this virtualization subsystem is a part of ONOS, it also exploits other subsystems. For example, to identify each edge port and generate address resolution protocol (ARP) request messages which are required in our proposed method, several existing services such as *EdgePortService*, *PacketService* can be combined. By using this feature, a process for virtual gateway deployment was implemented.

When we need to deploy a virtual gateway, the first step is to find physical interface/port which is a connect point connecting with external networks. This step is for embedding a virtual gateway to the connect point. Owing to the global view of SDN controller, ONOS can get all port information. Since a network device can have multiple ports, there are a number of ports in the network. Thus, tracking all port information is an inefficient method. Instead, we get only edge ports of each switch. The edge port information is provided by *edgePortService* of ONOS. However, this information is insufficient to identify an interface connecting with outside. This is why we generate ARP requests via edge ports. We assume that a default physical gateway IP address is announced. By using the IP address, each ARP request packet is generated and forwarded to receive a reply message. Accordingly, one of ports can receive reply ARP message if the port connects with physical gateway which provides external connectivity. After the reception, ONOS creates a virtual gateway and adds virtual port mapping to the identified physical port. As a result, a virtual gateway which has external connectivity is provisioned. The detailed process is presented in Algorithm 1. In addition, we exploited *PacketService* to allow virtualization subsystem to listen data packets received from network devices and to emit data packets out onto the network. This enables a network hypervisor to generate ARP request packets and receive ARP reply packets. Basically, *PacketService* emits packets through a connect point which consists of network device ID and a port number. Thus, virtualization subsystem can send a packet through a connect point. Moreover, ONOS itself provides a *OutBoundPacket* service to build a packet emitting onto the networks. By using this service, we can generate ARP request packets. After sending ARP request messages, it should listen an event which ARP reply messages are arrived. By default, each network device has a default rule to deliver ARP packets to the controller [16]. Thus, we deployed a *PacketProcesser* which is responsible for listening a packets from a network devices and processing them. This *PacketProcessor* is added when the first ARP request packet is sent, and then it is removed at least one ARP reply packet is returned. We extended a command line interface (CLI) to provide a way for virtual gateway deployment. Thus, external connectivity of each vSDNs can be provided optionally. If there is a vSDN which does not want to connect with external network, it is possible. The overall service relationship is shown in Fig. 4.

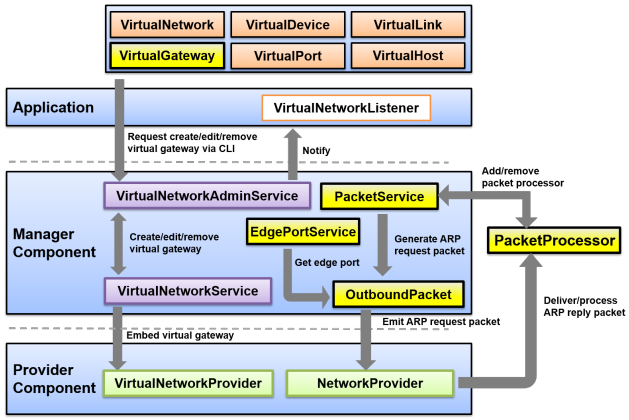

Fig. 4. Service relationship for virtual gateway deployment

ONOS virtualization subsystem also provides a simple virtual network embedding mechanism based on port-based mapping by isolating each virtual network using VLAN tags. After virtual network embedding, ONOS needs to install virtual flow rules for packet forwarding in the virtual network. The virtual flow rules should be translated and then deployed in each physical device. For example, a flow from port 1 of virtual device 1 out through port 3 could be changed into the flows from port 2 of physical device 1 out through port 1 of physical device 2. To construct a path connecting physical device 1 and 2 for a virtual flow rule, ONOS virtualization subsystem introduce a notion of *Internal Routing Algorithm*. By obtaining internal routing path using *Internal Routing Algorithm*, the substrate topology virtualization can be achieved. Additionally, we implemented forwarding application on ONOS to achieve flow rule deployment of each virtual network.

Algorithm 1 Virtual gateway deployment

Require: Input ($vNetId$, $vDeviceId$, $Gateway\_ip$)
Ensure: Output ($vGateway$)

```
edgePorts ← edgePortService
for current_port ∈ edgePorts do
    Generate ARP_Request_Msg
    ARP_Request_Msg.dst_ip ← Gateway_ip
    if Receive ARP_Response_Msg then
        port ← Response_Input_Port
        Create_vDevice(vDeviceId)
        Create_vDevice_Port(vDeviceId, vPortNum)
        Bind_vPort(vDeviceId, vPortNum, port)
        break
    end if
end for
```

V. Evaluation

To evaluate the proposed virtual gateway, we built a emulated SDN testbed using the Mininet emulator. This testbed ran on a bare metal server (hexa-core 3.47 GHz Intel Xeon CPU with 48 GB RAM). In this testbed, we created a network topology shown in Fig. 5. It is a k-level tree topology. According to the k value, the number of switches is determined. We built network topology from k=1 to k=7, so the number of switches is from 1 to 127. Since each host of the network uses a private IP address, we deployed NAT component between the physical gateway and the Internet. As a result, all traffics which needed to transverse the external networks are translated via the NAT component.

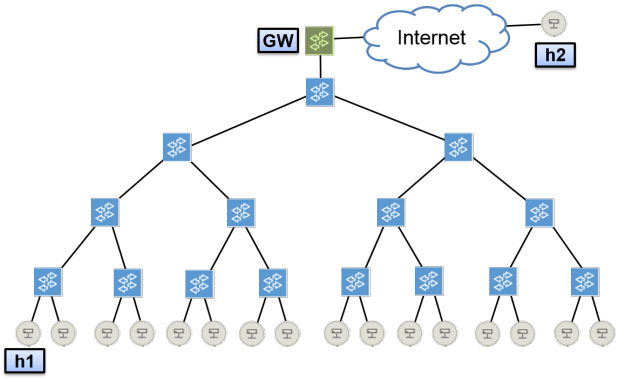

Fig. 5. Testbed (k=4)

A. Round Trip Time

vSDN which has the virtual gateway can communicate with external networks. To show the external connectivity, we generated ICMP ping messages between two hosts that one of host is located in vSDN and the other is located in external network. In addition, we experimented it on 4-level tree topology and deployed an external host as a virtual machine through Google cloud platform [17]. Thus, they are connected via the Internet as shown Fig. 5. In this condition, h1 generates ICMP ping messages 100 times to the external host, h2. Moreover, we indirectly show how virtual network affects network performance compared to physical network. For that reason, we measured latency, Round Trip Time (RTT), by considering 3 scenarios 1) without network virtualizaion in physical network, 2) with a virtual gateway in vSDN (big switch), and 3) with a virtual gateway in vSDN (physical network clone). According to the experiment, RTT is measured as shown Fig. 6. We measured RTT in terms of initial RTT and average RTT. The initial RTT includes installation time for virtual flow rules. When a virtual gateway is not deployed in vSDN, all requests targeting to external networks fail to receive reply messages. On the other hand, if a virtual gateway is deployed, ping messages can be delivered from h1 to h2 and RTT is measured. On the physical network, initial RTT is 116ms and average RTT is 75.44ms. In contrast, initial RTT is 163ms and average RTT is 75.56ms on vSDN as a big switch. Finally, intial RTT is 166ms and average RTT is 129.92ms on vSDN as a physical network clone. Since the scenario 3 is more complex vSDN, both measurement results are increased. According to the result, we can conclude that the virtual gateway allows vSDN to connect with the external network. Furthermore, it shows that when traffic is delivered to the external network based on virtual network, it increases latency. This is why the virtualization gives a virtualization overhead to the physical network.

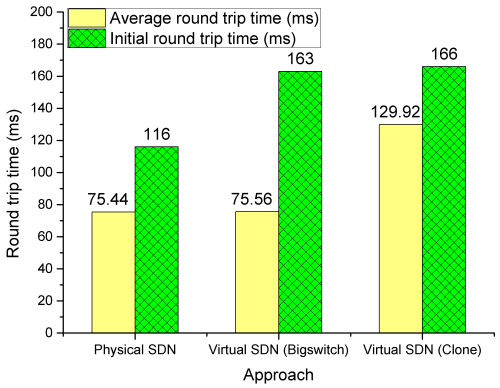

Fig. 6. Round trip time

B. Deployment Time

In order to evaluate overhead for virtual gateway deployment, we measured deployment time when virtual gateway creation is required through CLI. After request, a virtual gateway is created and it finds a physical port connecting with external network and maps to the port. We measure the period from a request and to the end of mapping process as shown Fig. 7. If the number of switches are small, from 1 to 15, the average deployment time is quite small as well. They are 21.5ms, 21.8ms, 20.2ms, and 24.5ms respectively. However, if there are many switches, deployment time increases as

well. In this case, average times are 172ms, 2304ms, 4221ms respectively when the number of switches is from 31 to 127. This is because the number of switch increases, edge ports increase as well. Thus, a network hypervisor needs more time to identify each edge port and generate an ARP request packet via the port.

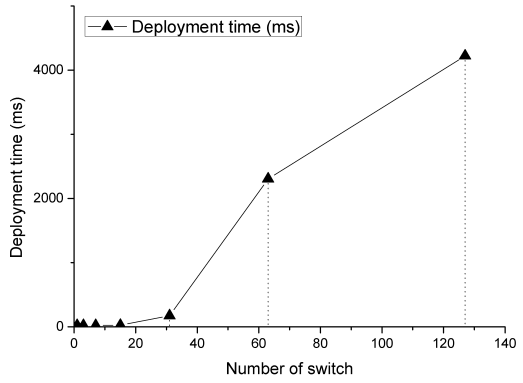

Fig. 7. Deployment time

VI. Conclusion

Network virtualizaion with SDN has attracted significant attention in the network area. With the useful features of SDN and SDN-based network hypervisor, vSDNs can be created. However, existing SDN-based network hypervisors do not provide a way for external connectivity as a network solution. Thus, in this paper, we have proposed a virtual gateway in vSDNs for external connectivity. A virtual gateway is embedded on a physical network and it roles as a virtual device. Our contribution is to propose a concept of virtual gateway which can be easily created and embedded to allow each vSDN to connect with external networks.

Our future work includes defining functional requirements for a virtual gateway in more detail and implementing services for them. In this paper, we initially have shown a feature and functionality of virtual gateway using SDN-based network hypervisor, ONOS virtualization subsystem. Thus, the top priority task as a future work is to implement network functions provided by a virtual gateway such as traffic shaping, firewall, and load-balancing. Additionally, current virtual gateway only provides port-mapping based embedding mechanism which does not consider current utilization of the underlying network. For that reason, it needs to support automated and optimal virtual network embedding mechanism. Furthermore, since virtual networks run on the underlying physical network infrastructure, failures from the physical network can propagate to the running virtual networks. Likewise, if a physical device mapping to a virtual gateway has a problem, virtual networks lose external connectivity. In order to minimize the problem, we need to provide a resilience solution. Finally, we plan to improve the proposed virtual gateway using results obtained from additional experiments, and evaluate in terms of various performance matrices.

Acknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2017-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion). This work was also supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2017-0-01717, Research and Development of SDN-based Multi-protocol support network virtualization and virtual network snapshot technologies).

References

[1] N. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communications magazine*, vol. 47, no. 7, 2009.

[2] J. Daniels, "Server virtualization architecture and implementation," *Crossroads*, vol. 16, no. 1, pp. 8–12, 2009.

[3] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, "Network virtualization: a hypervisor for the internet?" *IEEE Communications Magazine*, vol. 50, no. 1, 2012.

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

[5] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, vol. 1, p. 132, 2009.

[6] R. D. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "Vertigo: Network virtualization and beyond," in *Software Defined Networking (EWSDN), 2012 European Workshop on*. IEEE, 2012, pp. 24–29.

[7] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[8] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, G. M. Parulkar *et al.*, "Openvirtex: A network hypervisor." in *ONS*, 2014.

[9] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous *et al.*, "Carving research slices out of your production networks with openflow," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 129–130, 2010.

[10] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[11] Virtual machine manager (vmm). [Online]. Available: https://docs.microsoft.com/en-us/system-center/vmm/

[12] 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.

[13] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. IEEE, 2014, pp. 1–6.

[14] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 655–685, 2016.

[15] Y. Han, T. Vachuska, A. Al-Shabibi, J. Li, H. Huang, and W. Snow, "Onvisor : Towards a scalable and flexible sdn-based network virtualization platform on onos," *International Journal of Network Management*, 2017.

[16] R. Di Lallo, G. Lospoto, M. Rimondini, and G. Di Battista, "How to handle arp in a software-defined network," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE, 2016, pp. 63–67.

[17] Google cloud platform. [Online]. Available: https://cloud.google.com/