Libera for Programmable Network Virtualization

Gyeongsik Yang, *Member, IEEE*, Bong-yeol Yu, Heesang Jin, and Chuck Yoo, *Member, IEEE*

Abstract—Current network virtualization allows tenants to have their own virtual networks. However, new demands to "program" virtual networks at a finer granularity have arisen in that tenants want the ability to provision and control switches and links in their virtual networks. This study proposes a new concept called programmable network infrastructure-as-a-service (p-NIaaS) model. The p-NIaaS model enables tenants to program their own packet processing logic and monitor network status from any virtual network infrastructure, which is not possible with the current network virtualization. This article presents *Libera* network hypervisor that implements the p-NIaaS model. *Libera* overcomes the shortcomings of existing network hypervisors such as scalability, VM migration support, and flexibility. The evaluation shows that *Libera* platform is highly scalable and effectively supports VM migration. We also present the overheads of *Libera*. *Libera* incurs up to 11 percent overhead in comparison with a non-virtualized network, which we believe is promising in the first prototype of the p-NIaaS model.

I. Introduction

Cloud datacenters (DCs) are becoming integral infrastructures for IT services. Major services like "*Netflix*" and "*GE*" are operated on public clouds, such as "*Microsoft Azure*" and "*Amazon Web Services (AWS)*." Public clouds employ various service models such as infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) to meet tenant demands [1]. Each service model provides computing infrastructures, such as virtual machines (VMs) or containers, and software environments for tenants.

Because network traffic flows over switches and routers, cloud networking requires network virtualization (NV), where the network infrastructures (NIs), such as ports, links, and switches, are virtualized and provided in the form of a virtual network (VN). Tenants are provisioned with VNs using NIs. NVP (VMware) [2] and VFP (Microsoft) [3] are good examples of NV in DC.

The main objective of the current NV, exemplified by platforms such as NVP and VFP, is to automate the provision of VNs so that the time required to configure a VN is significantly reduced [2], [3]. However, the current NV does not allow tenants to manage their own VNs. Virtual NIs[1] for tenants, such as virtual switches (vSwitches), ports, and links, are created and controlled by DC administrators rather than by tenants directly. This contrasts with server virtualization, where tenants are empowered to provision, control, and monitor (in short, to program) all of their computing resources.

The demand for tenant programmability of virtual NIs has arisen for a variety of reasons. One is software-defined networking (SDN), whose premise is to separate the control plane from the data plane, allowing the central controller to manage the data plane. Tenants are now aware of SDN and want to control virtual NIs via controllers, posing a new demand to cloud providers [4]. Other demands are to provide topology provisioning, custom packet processing logic, and network monitoring per tenant [5].

The reason for the lack of tenant programmability is that current NV does not expose virtual NIs to tenants. Therefore, tenants do not have any direct programmability. In addition, VNs are based on overlay so that tenants can install their desired network policies only to edge switches, but not to other virtual NIs. Therefore, the applicable policies are quite restricted, and tenants have very limited programmability over virtual NIs.

This article explores how to extend NV to the level of server virtualization so that tenants can program their virtual NIs per their needs. To this end, we propose a novel service model

The authors are with the Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea.

[1]In this article, NI is used as a generic term; virtual NI and physical NI are used accordingly in the context.

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
This is an accepted version. Final article—DOI: 10.1109/MCOM.001.1900290

called programmable NI-as-a-service (p-NIaaS) that empowers tenants to program virtual NIs directly. With the p-NIaaS model, tenants can specify and create their own VN topologies. In addition, p-NIaaS allows tenants to manage their NIs using SDN controllers.

We implement the p-NIaaS model through a network hypervisor (NH) called *Libera*. It creates multiple virtual SDNs while assuring isolation between them. *Libera* differs from existing NHs in three aspects: scalability, VM migration support, and flexibility.

This article is organized with the background, motivation, related work, and concept of the p-NIaaS model. Then, *Libera* and its evaluation results are presented. Finally, we conclude this article.

II. Motivation and Related Work

Here, we explain the current NV and the necessity of p-NIaaS. Then, we review existing NHs, identifying their limitations. Subsequently, we introduce the related work to the p-NIaaS model and the novel differences of *Libera*.

A. Datacenter network virtualization

The state of the art in the current NV is well illustrated by NVP [2] and VFP [3]. NVP and VFP deploy a central network orchestration system that automates VN creation. A VN is based on overlay networking, such as virtual local area networks (VLAN), generic routing encapsulation (GRE), and stateless transport tunneling (STT), which generates a tunnel between the edge switches. NVP installs virtual NIs on the software edge switch. If a tenant requires L2 switching, NVP adds a flow rule (FR) for L2 switching on the edge switch. VFP has accelerated software switch performance by offloading packet processing tasks to the SmartNIC in hosts.

However, these NV technologies have limitations in VN programmability. First, tenants can only request the installation of policies at the edge switches because the technologies are based on overlay. In addition, the types of policies are restricted – for example, for Azure and AWS, only private IP addresses (network address translation) or firewall policies are allowed [6], [7]. Other policies, such as matching arbitrary packets, attaching a new header, or rewriting a specific packet header field, are not permitted.

Second, tenants cannot provision their specific VN topologies between the edge switches because a single orchestration determines the topology between edge switches. Moreover, once a topology is determined, the orchestration system installs the required NIs on the core switches based on DC policies, which leaves no room for tenants to program and control NIs.

Other limitations include:

- Custom packet processing logic [5]: A streaming service may want to program the functionality of content caching proxies in the network because such custom packet processing can enhance service quality. Such custom processing needs to implement content storing operations at the necessary NIs, dynamically redirect traffic to proxy-enabled NIs, and/or install appropriate policies, none of which are allowed in the current NV.
- Network monitoring: A tenant may want the monitoring of traffic and virtual NI states to diagnose service performance bottlenecks. However, in the current NV, performance can be measured only in a VM or container. To accomplish this goal, a tenant must be allowed to monitor performance in an end-to-end manner, including switches and ports.

The above examples are not imaginary but are plausible scenarios that can benefit tenants. It is clear that NV needs to be extended to make NIs programmable, which has led us to develop the p-NIaaS model.

B. SDN network virtualization

Virtual SDNs have been studied with NHs [8]. An NH is located between the SDN controllers and the physical NI. It abstracts the underlying physical NIs and provides address and topology abstractions, such as virtual IP, MAC addresses, switches, and ports. SDN controllers send control messages that contain virtual address and topology information so the NH should interpret them to avoid collisions between tenants. For example, the NH converts topology information, such as virtual port numbers and switch IDs, into physical ones.

Existing NHs successfully implement these basic functionalities, but they do not fulfill the

diverse new demands for clouds. A brief explanation of how *Libera* overcomes these identified shortcomings follows, with details to be explained later in the article.

1) Scalability: The scalability of a NV platform is an important consideration [5]. Scalability refers to the consumption of a physical switch's memory, the CPU cycles of the NH, and the bandwidth between the NH and switches (control channel bandwidth). The existing NH is known to increase the consumption of CPU cycles by up to 170 percent, and to increase the physical memory consumption of a switch by up to 390 percent compared to current NV solutions [9]. This lack of scalability hinders the market acceptance of NH. Therefore, the p-NIaaS model requires a mechanism to enhance scalability. *Libera* introduces the virtualization of FRs, which improves NH scalability up to eight times.

2) VM migration support: In clouds, VM migration is frequently performed to avoid service failure or to save energy. However, existing NHs do not provide any mechanisms that supports network reconfiguration for VM migration [8]. *Libera* implements network reconfiguration which installs forwarding paths between the hosts without the intervention of tenants. This is done through the transparent mapping of the NIs.

3) Flexibility: Existing NHs can only support certain type of physical switches (pSwitches), predominantly OpenFlow 1.0. However, because new switches that constitute SDN networks have recently been developed, the p-NIaaS model should be flexible enough to accommodate the new switches and cover switch-specific differences, such as control message syntax. Therefore, *Libera* is designed to accommodate new switches and their architecture differences effectively.

C. Related work

Table I compares *Libera* with related work in terms of VN programmability. NVP [2] and VFP [3], which are based on overlay, are the common approaches currently applied in existing DCs. OpenStack Neutron is also widely used. Neutron controls the network connections between compute nodes in an OpenStack-based cloud system and allows the external SDN controller to be used for the control. SONA (based on ONOS) [10] and Neutron-ODL (based on OpenDayLight) [11] are examples. SONA and Neutron-ODL automate the creation of overlay connections between edge switches, like NVP and VFP. Similar to NVP and VFP, policy installation on every vSwitch and custom VN topology provisioning are impossible because physical network is controlled only by a cloud orchestrator. Network monitoring is possible for only edge switches. Further, *Libera*'s components for cloud networking, such as scalability and VM migration support, are not fully deliverable. Only VM migration support is possible on Neutron-ODL.

The term "network-as-a-service (NaaS)" takes on different meanings as follows: NaaS from Cisco [12] focuses on network management efficiency, which has a different scope from *Libera*. Hardware-based network programmability has been proposed in NaaS by Costa *et al*. [13]. It installs FPGA-based specialized hardware (called NaaS box) to switches and permits the direct control of tenants. The NaaS box provides a certain level of programmability to tenants but is quite different from *Libera* in the following aspects: First, although NaaS box aims to provide programmability, it is not based on the notion of "VN." Therefore, isolation between tenants is not provided, which is a critical limitation for clouds. The authors said in the paper that the isolation between VNs has been left as future research. Also, the scope of programmability that NaaS box provides is limited. For instance, custom VN topology cannot be provisioned because the tenants have direct access to the NaaS box. Policy installation and network monitoring are possible only for the NaaS box-installed switches. Furthermore, NaaS box does not provide scalability or VM migration support, both of which are essential for clouds. The term "NaaS" has been used without the standard meaning elsewhere. Therefore, this paper uses p-NIaaS to clearly refer to the programmability of VNs in clouds.

III. p-NIaaS model

Figure 1 presents the components and interactions of the p-NIaaS model. Arrows I1, I2, and I3 in Fig. 1 illustrate the interactions among tenants, *Libera* hypervisor, and physical network. The core of the p-NIaaS model is *Libera* hypervisor, which provides a VN to each tenant. A tenant

TABLE I. Related work comparison[a].

	Approach	Policy installation on all vSwitches	Custom VN topology provisioning	Network monitoring on all virtual NIs	Scalability	VM migration support
NVP [2]	Overlay	×	×	×	×	×
VFP [3]	Overlay	×	×	×	✓	✓
SONA [10]	Overlay	×	×	×	×	×
Neutron-ODL [11]	Overlay/L2/L3	×	×	×	×	✓
NaaS box [13]	Specialized hardware	○	×	○	×	×
Libera	NH	✓	✓	✓	✓	✓

[a] ✓: fully supported, ○: partially supported, ×: not supported

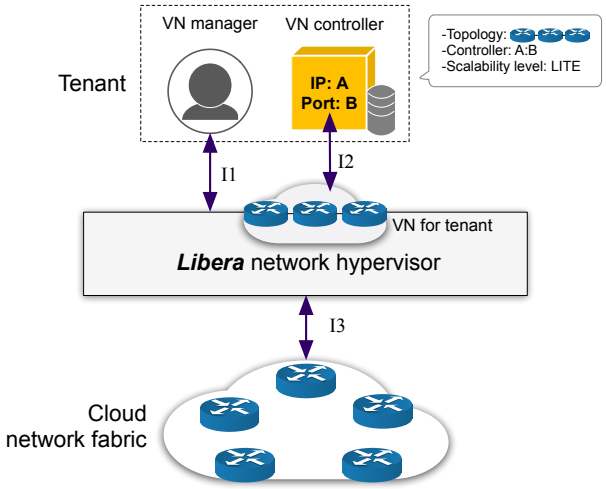

Fig. 1: p-NIaaS model.

uses the VN manager and the VN controller as follows.

A. *VN manager*

The VN manager (VNM) of each tenant is the operator of the VN and makes various requests to *Libera*. There are two types of requests: topology provisioning and scalability level. The requests from VNM goes through I1 (Fig. 1).

1) Topology provisioning: The VNM initiates the creation of a VN with a specific topology and virtual NIs. Virtual NIs include all of the NIs constituting the VN, such as vSwitches, ports, and links. The VNM can designate the type of a vSwitch to be either OpenFlow, white-box, or P4. The VNM can also create multiple ports on each vSwitch. Similarly, a virtual link can be created by connecting two virtual ports.

2) Scalability level: *Libera* hypervisor provides FR reduction in order to enhance scalability. The VNM can define the degree of reduction.

B. *VN controller*

After the VNM creates a VN and its virtual NIs, the tenant uses the VN controller (VNC) for VN operation. The VNC can program the vSwitch or virtual port by sending control messages to *Libera* through I2 (control channel). *Libera* provides a control channel for each vSwitch. Because the vSwitch belongs to only one VN, the control messages of each VN are separately delivered to *Libera*. For FRs, the VNC can install desired FRs to any vSwitches at any time so that packets are forwarded or dropped dynamically. The VNC can also program its custom switch logic on a vSwitch if the switch allows a programmable logic (e.g., white-box switches). The VNC can also gather statistical information from vSwitches or virtual ports. Such programmability is not feasible with the current NV in DC.

We have designed *Libera* to use any existing SDN controller as VNC without modification. This means that virtual NIs are exposed to the VNC as SDN switches and links.

IV. *Libera* Components

Upon receiving requests from the VNM and control messages from the VNC, *Libera* provides the requested VN services. Figure 2 shows the components of *Libera*.

Libera's VNM handler in Fig. 2 delivers the VNM's requests to the proper *Libera* compo-

nents. Topology provisioning requests are delivered to the flexibility component, and scalability level requests are given to the scalability component. Similarly, control messages from the VNC are processed in *Libera* as follows. First, the control message is processed in the flexibility component, and subsequently, passed to the scalability component and VM migration support component. So, the *Libera* components process control messages in isolation.

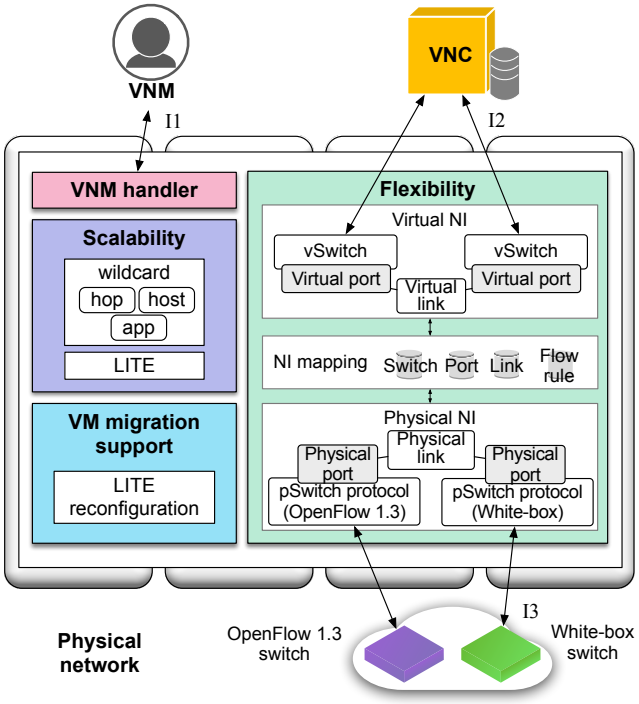

Fig. 2: *Libera* architecture.

A. *Flexibility component*

The goals of the flexibility component are to provide mapping changes dynamically and to support various pSwitches. To this end, *Libera* has a virtual NI layer, NI mapping layer, and physical NI layer.

1) Virtual NI layer: This layer provisions VN topologies with virtual NIs. When the VNM creates a vSwitch, the vSwitch is designed to be compatible with existing SDN controllers. Therefore, the vSwitch emulates two generic operations of the SDN switches: 1) the control channel and 2) the message handler. The control channel delivers the control messages (I2 in Fig. 2). The message handler interprets the control message syntax and parses the messages, according to the virtual NI's type (e.g., OpenFlow 1.3). In addition, this layer creates a virtual port for each vSwitch upon requests. When the virtual link is created, it also stores the end-points of the virtual link that are two virtual ports.

2) NI mapping layer: This layer introduces the notion of virtual FRs and physical FRs. Virtual FRs are installed in vSwitches, and physical FRs in pSwitches. This layer makes FR mapping between virtual and physical FRs. The detailed description of virtual and physical FRs and their mappings are discussed in [9]. With the notion of FR mapping, *Libera* can dynamically change NIs associated with the mappings. For instance, when a host migrates, the host is connected to a new pSwitch, and the forwarding path for the migrated host needs to be changed. Therefore, the vSwitches in the VN affected by the VM migration should be mapped with the new pSwitches.

The NI mapping layer updates and installs the FRs in the pSwitch. Then this layer modifies the FR mappings accordingly keeping the FRs in vSwitches unchanged. With this mapping change, *Libera* can provide VN services seamlessly even when a host migrates dynamically. Note that the existing NHs do not provide FR mapping so they cannot support the dynamic change of NIs.

3) Physical NI layer: This layer aims to support various pSwitches (e.g., OpenFlow 1.0, 1.3, and white-box switches). Existing NHs mostly support only one pSwitch type (OpenFlow 1.0). However, each pSwitch has a different packet processing sequence and its control message syntax. To support the various pSwitches, *Libera* should include a separate implementation for each pSwitch considering its architectural differences. In *Libera*, the physical NI layer implements switch abstraction protocol that covers all the pSwitch-specific details. The switch abstraction protocol is designed to be extensible and is provided in the form of API to other *Libera* components. This layer also translates the address or topology information of control messages from other components to the pertinent pSwitch for VN isolation. With this abstraction protocol and translation, *Libera* can support various pSwitches and make it possible for other *Libera* components to implement their logic, agnostic to the pSwitch architectures.

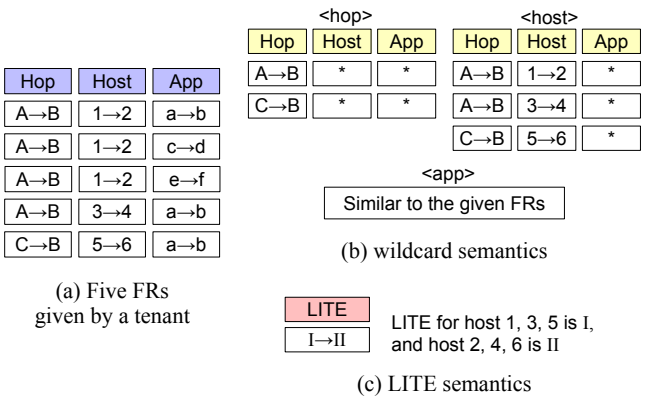

Fig. 3: FR reduction semantics.

B. Scalability component

This scalability component enhances the scalability of *Libera* by reducing the number of FRs. This component does not reduce all FRs. If FRs exist for dropping packets for certain matched port fields, the reduction of FRs by changing its match fields can create conflicts. The reduced rules cannot correctly distinguish the packets because the match fields of the rules have been modified. To avoid this, the scalability component only reduces the FRs for "packet forwarding" operations that sends packets from one port to another port of a switch, which is free from policy conflicts. The reduction is achieved through two reduction semantics: wildcard (hop, host, and app) and location-and-tenant-based identifier (LITE). Figure 3 shows how reduction semantics are applied to the five FRs.

1) Wildcard: Wildcard works on the match fields of each rule. When a FR matches the packet header field, its actions are applied to the matched packets. Therefore, the number of matched FRs can vary depending on the match fields. For instance, by matching the hop pair, five rules (Fig. 3a) are reduced to two (<hop> in the upper-left of Fig. 3b) because the host pair and application (app) pair are wildcard'ed. Similarly, matching the host pairs reduces them to three rules (<host> in the upper-right of Fig. 3b). The detailed mechanisms are discussed in [9].

2) LITE: LITE consists of a tenant ID and an edge switch identifier to which a host of the tenant is attached. If a host is attached to a switch whose ID is 0a, and the host belongs to tenant 1, the LITE would be 0x010a. *Libera* allocates one LITE per host and uses the LITE values of the source and destination hosts instead of the IP addresses. Therefore, packets can be grouped with the same LITE pair at the ingress edge switch. Packets are then forwarded based on the LITE pair at the core switches; thus, LITE reduces the number of FRs because rules that match individual flows are aggregated. Figure 3c illustrates the effect of the LITE reduction when hosts 1, 3, and 5 are attached to the same switch, while 2, 4, and 6 are connected to a different switch. Hosts 1, 3, and 5 are assigned the same LITE, value I; 2, 4, and 6 are assigned value II, resulting in a single rule being installed (Fig. 3c). LITE's full design is described in [14].

C. VM migration support component

The VM migration support component achieves network reconfiguration by utilizing the LITE scheme [14]. When one host migrates, this component examines the new location information (edge switch) of the migrated host and creates a new LITE for the host. Subsequently, the component updates FRs for core switches based on the new LITE. This component then changes the mappings of the vSwitch to the new edge and core switches using the NI mapping layer (of flexibility component) to enable seamless VN services. The mappings of the virtual FRs in the vSwitch are also updated to the new LITE-based rules.

This scheme conceals the explicit migration (from the VNC) and reconfigures VNs at the NH level. This reduces the delay in reconfiguration because the migration event is not propagated to the VNC, and the necessity of preparing for the VM migration is eliminated from the VNC. Detailed algorithms are found in [14].

V. Proof of Concept

Libera is implemented on top of OpenVirteX (OVX) [8], which is an open-source NH. The source-code of *Libera* is released through GitHub[2].

We evaluate *Libera*'s components for the scalability and VM migration support. The overheads of the entire *Libera* system are presented in terms of FR processing time, memory consumption, and the number of control messages. All experiments

[2]The detailed execution flow and instructions to run *Libera* are included in https://github.com/os-libera/Libera.

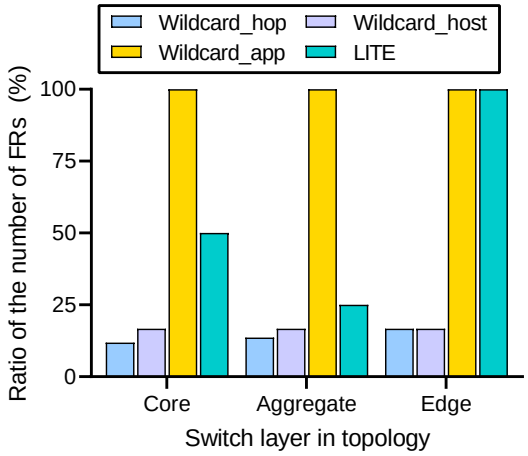

Fig. 4: Ratio of the number of FRs per reduction semantics.

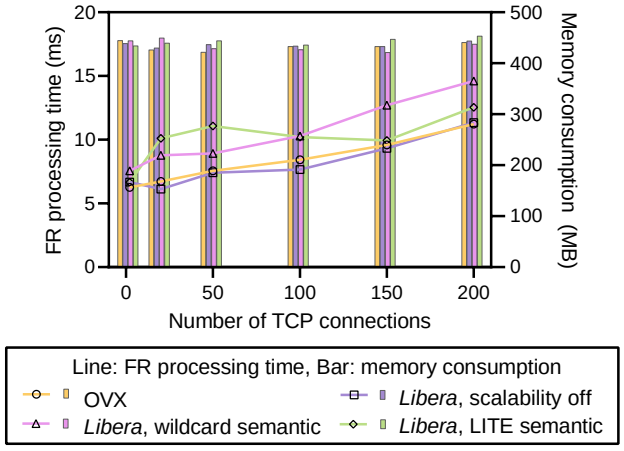

Fig. 5: FR processing time and memory consumption.

are conducted in a 4-ary fat-tree topology which consists of core, aggregation, and edge switches[3] We use different switches: OpenFlow 1.0, 1.3 software switches, and OF-DPA-based hardware white-box switches to demonstrate the flexibility of *Libera*. Two servers are used to run *Libera* and ONOS as VNC. All results are measured more than 20 times to obtain stable values.

Libera is based on the design of our previous studies [9], [14] but is implemented to realize the p-NIaaS model as a complete system. The hardware switch-based evaluations and overheads of the entire *Libera* system have not been reported.

A. Scalability

For scalability evaluations, we use OpenFlow 1.0 software switches (Open vSwitch) with five tenants that have six TCP connections each. This is based on *Libera*'s programmability. For each tenant, *Libera* first allows the tenant to provision its own VN topology as a 4-ary fat-tree topology. Second, *Libera* lets the tenant use ONOS to program its VN. Third, the tenant creates FRs that match six addresses (source/destination ethernet, source/destination IP, and source/destination port addresses) and installs the rules at all vSwitches. This programmability is not possible with OpenStack-Neutron [10], [11] or Cisco's NaaS [12].

Figure 4 shows the decrease in the number of FRs per reduction semantics. The y-axis is the number of FRs normalized to the number of rules of OVX. The maximum reduction is eight times with the wildcard_hop semantics at the core switch. These results are promising because, given the same amount of memory, eight times more tenants are deployable with our hypervisor.

B. VM migration support

We use hardware switches to measure network reconfiguration time to evaluate the efficiency of *Libera*'s VM migration support. Four OF-DPA-based white-box switches are used (one Edgecore AS-6712 switch and three AS-5712 switches). The tenant provisions its VN as a 4-ary fat-tree topology similar to the scalability experiments, and ONOS's reactive forwarding is used to create FRs, which is possible with the *Libera*'s programmability. The FRs match the four address fields (source/destination ethernet and source/destination IP addresses).

When a host moves to another edge switch, the reconfiguration time takes 10.5 ms on average to reset the forwarding path. Note that VNC is not engaged in VM migration, which makes ONOS be used unmodified as VNC. VM migration in clouds requires service downtime ranging from milliseconds to seconds [15]. Thus, *Libera*'s network reconfiguration time is satisfactory.

[3]Both core and aggregate switches in the fat-tree topology correspond to the core switches in previous sections.

C. Overheads

1) FR processing time and memory consumption: The overheads of *Libera* are compared to OVX with OpenFlow 1.3 Open vSwitches. Figure 5 presents the processing time per flow rule and the memory consumption of *Libera* when 2, 10, 50, 100, 150, and 200 TCP connections exist. The results are measured for four settings: 1) "OVX," 2) "*Libera* with scalability component off," 3) "*Libera* with wildcard semantic," and 4) "*Libera* with LITE semantic." The scalability component is involved for "*Libera* with wildcard semantic," and for both the scalability and VM migration support components for "*Libera* with LITE semantic."

In Fig. 5, the line graph represents the FR processing time. The FR processing time increases as the number of TCP connections increases. The maximum increased time of *Libera* compared with "OVX" is 2.18 ms ("*Libera* with wildcard semantic"). However, this processing for FRs occurs only once when network connections are created, which makes the increase not an issue. Also, when the FRs are pre-installed, the FR processing time can be eliminated.

The bar graph in Fig. 5 shows the memory consumption. All four settings exhibit similar memory overheads, consuming 436.7 MB on average. The difference between the maximum and minimum memory consumption is 7 percent of the maximum consumption, which is not significantly high. The results prove that *Libera* does not require much memory.

2) Number of control messages: The p-NIaaS model requires control messages to traverse an additional layer, *Libera* hypervisor, which inevitably increases the number of control messages. We collect control messages generated by the VNC and *Libera* in the setting of the scalability experiments. The results show that control messages for FR installation per tenant add bandwidth overhead up to 11 and 25 percent with the wildcard_hop and wildcard_end semantics, respectively. This means that although the control messages are generated twice (first between the VNC and the vSwitches and secondly between *Libera* and the pSwitches), the bandwidth does not double. This is because the scalability component reduces the control messages between *Libera* and the pSwitches, while the messages between the VNC and vSwitches remain unchanged.

VI. Conclusion

This article proposes a concept called the p-NIaaS model (programmable network infrastructure-as-a-service). The model is to give tenants the new ability to program virtual NIs in their virtual networks, which is not possible with existing hypervisors. To prove the concept, we build *Libera* and evaluate its performance and overheads. A key advantage of *Libera* is that it runs with existing SDN controllers without modification while providing the full programmability of virtual NIs. Our results show that *Libera* achieves scalability many-fold higher than the existing network hypervisors. The outcome indicates that *Libera* minimizes the virtualization overhead in the control plane. So, we believe that this article reduces the gap between virtualized and non-virtualized SDN, which brings the network hypervisor a step closer to market acceptance.

References

[1] R. De Souza Couto *et al.*, "Network design requirements for disaster resilience in IaaS clouds," *IEEE Commun. Mag.*, vol. 52, no. 10, 2014, pp. 52–58.

[2] T. Koponen *et al.*, "Network Virtualization in Multi-tenant Datacenters," *USENIX NSDI*, 2014.

[3] D. Firestone, "VFP: A Virtual Switch Platform for Host SDN in the Public Cloud," *USENIX NSDI*, 2017.

[4] M. Jammal, *et al.*, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[5] H. Alshaer, "An overview of network virtualization and cloud network as a service," *Int. J. Netw. Manag.*, vol. 25, no. 1, 2015, pp. 1–30.

[6] "Azure Virtual Network | Microsoft Docs." https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview.html, accessed October 28, 2018.

[7] "What Is Amazon VPC? - Amazon Virtual Private Cloud." https://docs.aws.amazon.com/vpc/latest/user

guide/what-is-amazon-vpc.html, accessed October 28, 2018.
[8] A. Blenk *et al.*, "Survey on Network Virtualization Hypervisors for Software Defined Networking," *IEEE Commun. Surveys & Tutorials*, vol. 18, no. 1, 2016, pp. 655–685.
[9] G. Yang *et al.*, "FlowVirt: Flow Rule Virtualization for Dynamic Scalability of Programmable Network Virtualization," *Proc. IEEE 11th Int'l. Conf. Cloud Comput.*, 2018.
[10] F. Foresta *et al.*, "Improving OpenStack Networking: Advantages and Performance of Native SDN Integration," *IEEE Int. Conf. Commun.*, 2018.
[11] "OpenStack and OpenDaylight: An integrated IaaS for SDN and NFV." https://www.openstack.org/assets/presentation-media/OpenStack-and-OpenDaylight-Integrated-IaaS-for-SDN-and-NFV.pdf, accessed August 13, 2019.
[12] "Why you should consider networking as a service - Cisco." https://www.cisco.com/c/en/us/solutions/enterprise-networks/network-as-service-naas.html, accessed August 11, 2019.
[13] P. Costa, M. Migliavacca, and A. L. Wolf, "NaaS: Network-as-a-Service in the Cloud," *Proc. 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.
[14] G. Yang *et al.*, "LiteVisor: A Network Hypervisor to Support Flow Aggregation and Seamless Network Reconfiguration for VM Migration in Virtualized Software-defined Networks," *IEEE Access*, vol. 6, 2018, pp. 65945–65959.
[15] R. W. Ahmad *et al.*, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *J. Netw. Comput. Appl.*, vol. 52, 2015, pp. 11–25.

ACKNOWLEDGMENT

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2015-0-00288, Research of Network Virtualization Platform and Service for SDN 2.0 Realization and No. 2015-0-00280, (SW Starlab) Next generation cloud infrastructure software toward the guarantee of performance and security SLA). This research was also supported by Next Generation Engineering Researcher Program of National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (No. NRF-2019H1D8A2105513), and a Korea University Grant.

Gyeongsik Yang (ksyang@os.korea.ac.kr) He received B.S., M.S., and Ph.D. degrees in computer science from Korea University in 2015, 2017, and 2019, respectively. In 2018, he worked as a research intern at Microsoft Research Asia. He is currently a research professor at Korea University. His research interests include network virtualization, datacenter networking, and SDN.

Bong-yeol Yu (byyu@os.korea.ac.kr) He received B.S. and M.S. degrees in computer science from Korea University in 2016 and 2019, respectively. His research interests include network virtualization, datacenter networking, and SDN.

Heesang Jin (hsjin@os.korea.ac.kr) He received B.S. degree in computer science from Kookmin University in 2018 and is currently an M.S. student at Korea University. His research interests include network virtualization, traffic engineering, and SDN.

Chuck Yoo (chuckyoo@os.korea.ac.kr) He received B.S. and M.S. degrees in electronic engineering from Seoul National University and M.S. and Ph.D. degrees in computer science from the University of Michigan, Ann Arbor. He worked as a researcher at Sun Microsystems. Since 1995, he has been at the College of Informatics at Korea University, where he is currently a professor. His research interests include server/network virtualization and operating systems.