A Case for SDN-based Network Virtualization

Gyeongsik Yang, Changyong Shin, Yeonho Yoo, Chuck Yoo
Department of Computer Science and Engineering
Korea University
Seoul, Republic of Korea
ksyang@os.korea.ac.kr, cyshin@os.korea.ac.kr, yhyoo@os.korea.ac.kr, chuckyoo@os.korea.ac.kr

***Abstract*—Network virtualization (NV) becomes an essential technology in cloud computing that isolates network flows for tenants. However, because existing NV technologies like overlay do not enable tenants to directly program (i.e., provision, control, and monitor) network resources, software-defined networking (SDN)-based NV (SDN-NV) has been proposed. Despite its great benefits, SDN-NV has been believed to bring considerable overheads due to the network hypervisor (NH). However, to date, there is no definite performance evaluation that proves the overheads of SDN-NV. To this end, this paper comprehensively investigates the performance and overheads of SDN-NV. Our experiment results reveal that SDN-NV provides the data plane performance comparable to or even better (up to 10.5× better TCP throughput) than the existing NV technologies. Also, the results on NH show that its overheads remain mostly constant, even when the number of switches, virtual networks, or network flows increases. In short, our evaluation indicates that the overhead of SDN-NV should not deter its practical use in datacenters.**

***Index Terms*—Network virtualization, Performance evaluation, Cloud computing, Software-defined networking**

I. Introduction

Network virtualization (NV) is indispensable in cloud computing for providing various network services within the physical infrastructure. NV is responsible for providing isolated network connections—called virtual networks (VNs)—between virtual machines (VMs) and containers. Typically, NV is realized through overlay networking, which attaches additional headers in front of each packet. In overlay networking, however, the tenants are not allowed to control the data plane switches because only data center operators configure overlay networking. To overcome this limitation, NV-based software-defined networking (SDN) has been proposed. SDN is a network architecture that centralizes the control functionalities of switches into an SDN controller (control plane). Then the switches become packet-processing devices and operate on the actions given by the SDN controller. As the SDN controller enables the central management of all switches, the NV of the switches becomes feasible, which introduced SDN-based NV (SDN-NV). SDN-NV places a network hypervisor (NH) above the physical network (PN) of switches (Fig. 1). The NH abstracts the underlying PN and creates multiple VNs in the context of SDN. Thus, each tenant controls its VN through its SDN controller (VN control plane in Fig. 1). This way, SDN-NV opens up new possibilities for tenants that can make VNs programmable [1].

Recent studies added the various functionalities to SDN-NV [2] (e.g., network slicing [3], policy composition [4], address virtualization [5], platform scalability [6], [7], and features for datacenters such as VM migration support [1], [8] and performance management [9]). However, the performance results published so far pose a serious challenge to SDN-NV in that it still needs to prove its practical value beyond research efforts. It is like the early days of (server) virtualization technology, which has been around for a long time but was rarely used in practice due to its overhead. When the overhead was proved to be rather small [10], virtualization technology took off and became a fundamental technology for cloud computing. This paper aims to comprehensively evaluate the overhead of SDN-NV to prove whether it is small enough to be deployed in cloud computing infrastructures. Thus, we conduct a critical review of evaluations in previous studies and categorize them based on PN and NH.

First, PN consists of a set of network devices (switches) that forward packets. Previous studies showed the PN performance, but they have critical limitations. First, the key performance of the PN, such as network throughput or packet latency, varies with the number of network switches or flows (which we call variables). Yet, the performance metrics have been measured over only a small range of variables (e.g., one to ten in the number of switches [8]), which is far from realistic scenarios. In addition, evaluations have been conducted with only a single tenant, although the SDN-NV system provides multiple VNs for multi-tenancy. More importantly, the PN evaluations have not been compared with existing NV technologies such as overlay (§II-C1). Such a comparison is essential because SDN-NV is a new generation of NV that enables the control of VNs by tenants. Thus, it is necessary to compare the performance of the PN with that of the existing NV technologies because PN performance determines the user service qualities.

Second, the NH is another source of the SDN-NV overhead because the NH puts additional processing in the control plane. We find that, for the NH, its overheads over the number of flows have not been benchmarked thoroughly. Also, previous studies have evaluated only a single tenant. In addition, most

This work was supported by Institute of Information & communications Technology Planning & Evaluation grant funded by the Korea government (Ministry of Science, ICT, MSIT) (2015-0-00280, (SW Starlab) Next generation cloud infra-software toward the guarantee of performance and security SLA). This research was also partly supported in part by the National Research Foundation of Korea (NRF) funded by the MSIT (NRF-2019H1D8A2105513) and by Basic Science Research Program through the NRF funded by the Ministry of Education (NRF-2021R1A6A1A13044830).
Corresponding author: Chuck Yoo.

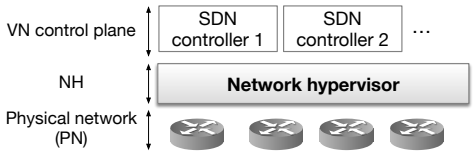


Fig. 1: SDN-NV architecture.

studies have evaluated NHs with the out-of-date version of OpenFlow (OF) and NV technologies. To summarize, the performance evaluation of previous studies on NHs has missed out key aspects of performance, multi-tenancy, and up-to-date system specifications (§II-C2).

Therefore, it still remains whether the overhead of SDN-NV is small enough to be deployable. To this end, this paper comprehensively investigates the performance and overhead of SDN-NV technology, bringing the following differences and novelties compared with the previous studies:

- Comparison with existing NV technologies, especially two dominant technologies for NV, overlay and NAT [11]. Our experiments are carried out with containers because it is the de-facto in cloud computing due to its high portability and little overhead [12].
- All-inclusive benchmarking on the data plane and NH covering the aspects missed in the previous studies—experiments over the number of switches, flows, and VNs and also multi-tenant environments.

Our benchmarking includes numerous measurements (more than 300 cases), and key results are as follows:

- Surprisingly, in the data plane performance, SDN-NV achieves 10.5× higher throughput (§III-B) and 33% better per-packet latency than overlay. Furthermore, our results show that SDN-NV has superior data plane performance over the existing NV technologies (§III-C).
- The overheads (message processing delay and CPU cycles) of NH mostly stay constant as the number of switches, VNs, and flows increase.
- The NH creates multiple VNs and processes flow rules for them. Therefore, it is expected that when the number of tenants increases, the NH becomes overloaded so that the processing delay on each flow rule can increase. Interestingly, our results show that NH can reduce its processing delay when the number of tenants increases (VNs). We discuss the results and reasons in §IV-B2.

II. Background

A. SDN-NV

SDN-NV consists of three layers (Fig. 1): 1) PN composed of physical switches, 2) NH, and 3) VN control plane that runs SDN controllers of tenants. Each tenant can request the creation of its VN topology, which includes the virtual switches and links that connect switches and their hosts. The NH receives the request and allocates the VN resources for the topology. After the VN resources are allocated, the tenant can manage and control the resources.

To illustrate how SDN-NV works, we explain the packet forwarding in SDN-NV. When a new packet arrives at the physical switch, the switch requests the NH for a flow rule to process the packet through a "`PACKETIN`" message. NH then delivers the `PACKETIN` to the corresponding SDN controller by determining which tenant the packet is destined for. Upon receiving the `PACKETIN`, the SDN controller calculates a path that forwards the packets and creates flow rules for the switches in the path. Then, the flow rules are wrapped as `FLOWMOD` messages and sent to NH, and NH installs the flow rules in the switches. The messages, such as `PACKETIN` and `FLOWMOD`, are called control messages. Between the switches and SDN controllers, the NH includes address virtualization mechanisms, which will be explained in the next section.

B. Address Virtualization Mechanisms in SDN-NV

The key difference between NH schemes lies in address virtualization. Address here refers to the network address of hosts, such as IP addresses. SDN-NV allows tenants to select arbitrary addresses for their hosts (virtual address) [2], and NH performs address virtualization to avoid address conflicts between tenants. Existing NHs proposed four kinds of address virtualization schemes: slicing, TID embedding, address mapping, and locator embedding.

First, slicing [3] divides the address space (e.g., of IP) and allocates a subspace to each tenant; thus, each tenant can use the addresses in the subspace, not an arbitrary network address. So, as the number of tenants increases, the subspace that each tenant uses becomes limited. Second, TID embedding embeds a tenant ID (TID) in every packet. Physical switches distinguish the virtual addresses through the tenant IDs. For example, FlowN used VLAN for implementing TID [13], and Libera used an existing field of the TCP/IP header (e.g., source MAC address) to embed the TID [1]. Third, address mapping [5] maintains a mapping between the virtual address and the physical address. So, each virtual address is linked to one physical address. Therefore, tenants can use their own virtual addresses via address mapping. Lastly, locator embedding uses a combination of TID and the switch's address (called LITE). This scheme supports the migration of a virtual host, which frequently occurs in datacenters for resilience and energy efficiency [8], [14]. With the LITE values, physical switches distinguish packets based on the switches where the packets come from. Note that for slicing, TID embedding, and address mapping, NH immediately processes each flow rule when it arrives at the NH. Conversely, for locator embedding, NH should wait for all the flow rules constituting a path between the source and destination hosts in order to know LITE values.

C. Literature Review

Seven major studies on SDN-NV are summarized in Table I. We categorize the studies into two subcategories: 1) benchmarking on SDN-NV and 2) proposing enhanced functionalities to SDN-NV. We specify the evaluation setting for each study that shows the methodology (simulation or emulation) and southbound interface (OF 1.0 or 1.3). The rest of Table I shows the evaluation metrics for the PN and NH, which are

TABLE I: Previous SDN-NV Studies Comparison in Terms of Performance Evaluation

Categorization			Benchmarking		Proposing enhanced functionalities to SDN-NV				
Study			This paper	perfbench [15]	FlowVisor [13]	FlowN [3]	OVX [5]	LiteVisor [8]	Libera [1]
Evaluation setting			Emulation (OF 1.3)	Simulation (OF 1.0)	Emulation (OF 1.0)	Emulation (OF 1.0)	Emulation (OF 1.0)	Emulation (OF 1.0)	Emulation (OF 1.3)
PN	Flow	Throughput	✔(■ ■ ▲ ● ○)		✔(□ ▲ ●)			✔(■ ▲ ●)	
		Per-packet latency	✔(■ ■ ▲ ● ○)						
	Switch	Switch CPU	Not focused		✔(□ ▲ ●)				
		# of flow rules						✔(■ ▲ ●)	✔(■ ▲ ● ●)
NH	Message processing delay	PACKETIN	✔(■ ■ ■ ▲ ●)	✔(■ ▲ ●)	✔(□ ▲ ●)				
		PACKETOUT	✔(■ ■ ■ ▲ ●)		✔(□ ▲ ●)				
		FLOWMOD	✔(■ ■ ■ ▲ ●)	✔(■ ▲ ●)				✔(■ ▲ ●)	✔(■ ▲ ● ●)
		PACKETIN −FLOWMOD	Available			✔(■ △ ● ○)	✔(■ △ ●)		
	Message throughput	PACKETIN	Not focused	✔(□ ▲ ●)					
		PACKETOUT		✔(□ ▲ ●)					
		FLOWMOD		✔(□ ▲ ●)					
	Resource consumption	CPU	✔(■ ■ ■ ▲ ●)	✔(■ ■ ▲ ●)	✔(■ ■ ▲ ●)				
		Memory	Not included						✔(■ ▲ ● ●)

marked with checks (✔). The symbols in Table I have the following meanings.

- Rectangles: experimental variables—the number of VNs (■), number of physical switches (■), number of flows in PN (■), and no variations (□).
- Triangles: topology virtualization—multiple VNs created and also physical switches are shared by multiple VNs (▲), multiple VNs created but physical switches are not shared (△), and single tenant (▲).
- Circles: the level of comparison—comparison to the previous NHs (●), to existing NV technologies (○), and no comparison (●).

1) PN: PN benchmarking has two metrics: flow and switch. Measurements on flow are used for analyzing the performance of SDN-NV on each flow (e.g., TCP or UDP), and measurements on switches are used for analyzing the resource consumption (e.g., CPU and memory) that a switch requires for packet processing

For the flow metric (throughput and latency), the previous studies have the following missing points. First, the flow throughput can be changed differently on the number of VNs, physical switches, and flows (we call experimental variables or simply variables). However, paper [13] measured the flow throughput without any variable changes (rectangle □ of [13] in Table I). Also, the paper [8] measured flow throughput by changing only the number of switches. This paper conducts experiments on all three variables (i.e., switches, flows, and VNs [1]). Second, per-packet latency is critical in many applications such as data mining [16], but it has not been measured previously. This paper includes the measurements on per-packet latency. Third, all existing evaluations on PNs are conducted with a single tenant (triangle ▲ in Table I), indicating that the performance and overhead of multi-tenancy have not been thoroughly investigated. Thus, we evaluate the performance of PN with multi-tenants. Lastly, existing studies did not compare the PN performance to existing NV technologies. This paper compares SDN-NV with overlay and NAT in the container environment (§III-A). Note that we do not measure the switch metrics (i.e., switch CPU and memory) because such metrics are heavily dependent on switch architectures [17], [18]. Thus, we focus on measurements on flow metrics.

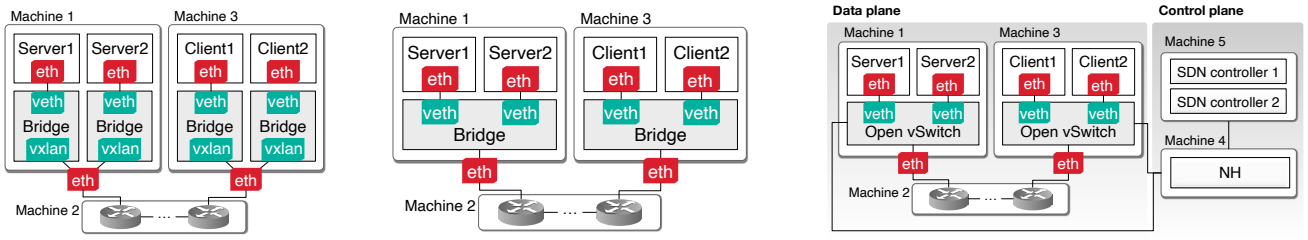
2) NH: For NH metrics, message processing delay, message throughput, and resource consumption are considered. First, message processing delay is the time for processing a control message within NH. It is measured for each message that NH processes, such as `PACKETIN`, `FLOWMOD`, and `PACKETOUT`. As shown in Table I, previous studies lacked experiments in terms of variables (rectangles) [8], [13]. Also, two studies [3], [5] measured the message processing delay of the messages without separation of messages (denoted as "`PACKETIN−FLOWMOD`" in Table I). This paper measures the processing delay of individual messages (i.e., `PACKETIN`, `FLOWMOD`, and `PACKETOUT`) for all the variables. Also, `PACKETIN−FLOWMOD` can be derived through the aggregation of messages.

Second, the message throughput is the number of messages processed per second by NH. Only one study [15] measured the message throughput because measuring the message throughput requires the arbitrary generation of messages from both PN and SDN controller, which is only possible in a simulation environment. Because this paper performs experiments in Fig. 2, where software switches (e.g., Open vSwitch), NH, and SDN controllers are utilized with actual network traffic, measuring the throughput is inevitably limited. Note that other studies in Table I did not measure the message throughput for the same reason. Rather, we focus on measuring the message delay, which exhibits the overheads.

Third, for the resource consumption of NH, two studies measured CPU cycles, and one study provided memory usage. In this paper, we measure both CPU cycles and memory consumptions but present the results on CPU cycles due to the page limit. We omit the results for memory consumption, but they tend to be very similar to CPU utilization.

Regarding the OF version, except Libera [1], all previous studies measured the NH overheads with OF 1.0. Considering that most SDN switches in datacenters support OF 1.1 or higher versions, the studies with OF 1.0 are in some sense obsolete. This paper conducts comprehensive evaluations based

[1]We check all the variables that have been used in previous studies, and in summary, consider a total of three—number of VNs, switches, and flows.

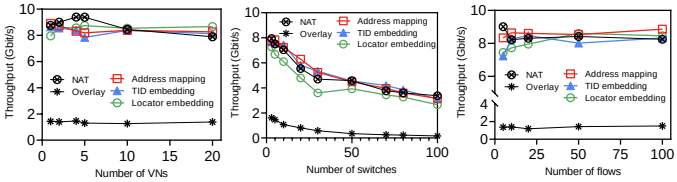


(a) overlay (b) NAT (c) SDN-NV

Fig. 2: Deployment of comparative groups on equipment.

TABLE II: Variable Changes for PN.

Changing variable	Switch	VN	Flow
Switch	3 to 100	1	1
VN	3	1 to 20	20
Flow	3	1	1 to 100



(a) VN (b) switch (c) flow

Fig. 3: Throughput of TCP connections.

on Libera with the OF 1.3 interface. Lastly, most studies are based on a single tenant (Triangle ▲ of [1], [8], [13] in Table I). Even in experiments with the multi-tenants [3], [5], each physical switch is dedicated to each tenant (Triangle △), which means that the PN is not actually virtualized. Therefore, this paper carries out experiments where several tenants share physical switches.

III. Empirical Benchmarking on PN

This section explains the evaluation methodology determined upon the literature review (§II-C1). Then, the evaluation results are presented and analyzed.

A. Evaluation Methodology for PN

Experiment environments. We run experiments with five comparative groups: 1) overlay, 2) NAT, 3) address mapping, 4) TID embedding, and 5) locator embedding. The first two are the popular NV techniques, and the latter three are for SDN-NV. Locator embedding is implemented with VLAN. Fig. 2 shows the experiment environments for the comparative groups. For overlay (Fig. 2a), a bridge is used with VxLAN and veth interfaces. Containers act as servers and clients of connections. Each container is bound to the bridge by a veth interface. Also, each bridge has a vxlan interface to perform packet encapsulation and decapsulation on each packet. In addition, for NAT (Fig. 2b), a bridge performs address translation on each packet. For SDN-NV (Fig. 8c), we use Open vSwitch (OVS) that supports the OF protocol. These settings are the representative and widely used ones for NV in containers [11]. The software switch (e.g., bridge or OVS) that processes the packets from the containers directly is located in Machine 1 and Machine 3. In Machine 1 and Machine 3, multiple containers are created to run TCP and UDP connections. Between Machine 1 and Machine 3, we create a number of software switches (OVS) in Machine 2.

Metrics. We measure throughput and latency on flow. Throughput determines the maximum amount of data that each server and client pair can transmit per second. We measure throughput using iperf3 with TCP flows. Each TCP segment size is set as the maximum transmission unit[2]. A TCP connection lasts 180 s, and the average throughput per second is shown. Latency is the round-trip time (RTT) between a server and client pair that shows the per-packet processing delay. We measure the per-packet latencies using sockperf with UDP packets as the pure overheads of NV technologies. For each delay experiment, the RTT measurement is carried out for 2 minutes in succession and repeated ten times.

Experiment variables. Table II lists the experimental variables for PN. For throughput, we vary the number of switches, flows, and VNs. The range of each variable is selected as the maximum possible range in the experimental machine. Table II means that when the number of switches changes from 3 to 100, a single TCP flow is generated with a single VN. In the case of variable VN, we create 20 flows with 3 switches, which produce 20 TCP flows to transmit the maximum number of packets. We evenly distribute 20 flows for all VNs. For the variable flow (i.e., 1–100), the number of switches is fixed at 3, and the number of VN is at 1.

However, for latency, we change only the number of switches. Latency measurements aim to compare the processing latency on each packet without network congestions. Because the latency is affected by background traffic (e.g., the addition of queueing delays), we do not evaluate latency in the varied number of flows and VNs.

B. PN Evaluation Results: Throughput

Fig. 3 plots the TCP throughput with experimental variables. Each dot is the aggregated throughput of the TCP connections. First, as the number of VNs increases (x-axis), Fig. 3a shows that the throughputs of four comparative groups maintain relatively consistent values: approximately 8 Gbits/s on average.

[2]We deploy different underlying software switches to enable NV technologies (e.g., bridge and OVS) as the NV technologies are deployed with them [11]. Also, we know that packet size affects performance because of the per-packet processing overheads whose variations depend on the switches. Such variations reduce by setting the packet size as the maximum transmission unit.

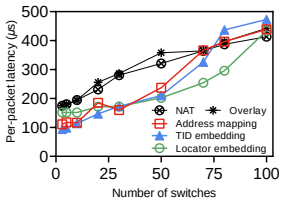


Fig. 4: Per-packet latencies (median).

However, overlay shows quite a poor throughput—1.38 Gbits/s on average.

In fact, the poor performance of overlay is investigated in previous studies [11], [19], [20] and is proven due to the heavy packet transformation using kernel features with bridges. To encapsulate and decapsulate packets for overlay (e.g., vxlan), the packet should traverse the networking stack of the kernel two times, which explains the poor throughput. On the other hand, NAT, address mapping, and TID embedding process packets only within the software switch, so their throughput is 5.8× higher than that of the overlay.

Note that locator embedding in SDN-NV also performs packet encapsulation and decapsulation. Specifically, locator embedding performs the packet encapsulation and decapsulation twice to include LITE values (§II-B) for the source and destination hosts on each packet. Locator embedding lowers its overhead by using SDN−enabled software switches (e.g., OVS)—OVS can offload the overheads of packet encapsulation and decapsulation on NIC (vxlan offloading); thus, the throughput of locator embedding approaches results similar to those of NAT, address mapping, and TID embedding.

Second, when the number of switches increases (Fig. 3b), the throughput decreases for all comparative groups. The throughput of overlay, NAT, address mapping, TID embedding, and locator embedding decreases by up to 90%, 58%, 59%, 59%, and 63%, respectively (comparing three switches and 100 switches). Also, between the five groups, overlay also shows the lowest throughput due to its encapsulation and decapsulation overhead—on average, the overlay throughput values are 88% lower than those of the other four groups.

Third, when the number of flows changes (Fig. 3c), all comparative groups show relatively constant throughput. Specifically, NAT, address mapping, TID embedding, and locator embedding show similar throughputs, 8–9 Gbits/s, while the overlay shows the lowest throughput—1.38 Gbits/s on average. The throughput of overlay is 83% lower than the average throughput of the other four comparative groups.

C. PN Evaluation Results: Per-packet Latency

We have measured the entire distribution of per-packet latencies, but due to the space limit, only the median latency (50%) is presented in Fig. 4. For all comparative groups, the latency broadly increases as the number of switches arises. By comparing the comparative groups, Fig. 4 shows that address mapping, TID embedding, and locator embedding have lower latencies than both NAT and overlay (e.g., 0.65×, 0.58×, and 0.73×, respectively, of averaged latencies of NAT and overlay) when the number of switches is small (l–50). This is because SDN-NV schemes are deployed with OVS, which has a better utilization than the kernel-based approach (i.e., bridge). Considering that typical connections in datacenters go through quite less than 50 switches [21], the results indicate that SDN-NV schemes have better per-packet latency than NAT and overlay in cloud computing.

On the other hand, as the number of switches increases, the latencies of address mapping, TID embedding, and locator embedding increase rapidly. Thus, all comparative groups show similar latencies (approximately 400–500 μs at 100 switches). The comparative groups have different implementations of NV, so that they should have produced a range of respective latencies. However, the latencies are amortized with the packet processing delays within the switches as the number of switches increases. Thus, the latencies of the comparative groups become similar when the number of switches is high.

TABLE III: Variable Changes for NH.

Changing variable	Switch	VN	Flow	PN topology	Hosts
Switch	**1 to 100**	1	1	Linear	Two for a VN
Flow	20	1	**1 to 200**	4-ary fat-tree	16 for a VN
VN	1	**10 to 70**	70	Linear	Two for a VN

IV. Empirical Benchmarking on NH

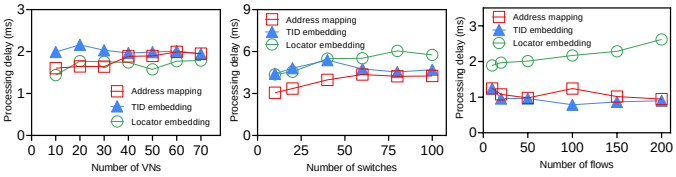
A. Evaluation Methodology for NH

Experiment environment. Different from the PN experiments (Fig. 2c) that create PN on three separated physical machines, PNs in NH experiments are created on a single machine. This is because NH experiments aim to see the overheads of NHs, not the PN, so we reduce the complexity of PN configuration. Mininet of OVS emulates PN topologies—linear and fat-tree topologies, as shown in Table III. In the linear topology, the number of switches in a forwarding path between a source and destination hosts is as many hops as switches in the network. For the 4-ary fat-tree topology, a path consists of five switches on average.

Flows are generated as TCP connections using iperf3. The created flow does not stop until the last flow is created so that NH receives control messages for flows consecutively. With the emulated PN on a physical machine, two additional machines run NH and SDN controllers of tenants, respectively. Each machine is equipped with two Intel Xeon E5-2690 CPUs (24 cores) and 64 GB of memory. A 10 GbE Ethernet connects three machines. For SDN controllers, we use ONOS. The match fields of flow rules from ONOS include ethernet, IP, and port addresses to process the new flows in the PN with a separate flow rule so that the number of control message processing tasks of NH is the same as the number of flows[3].

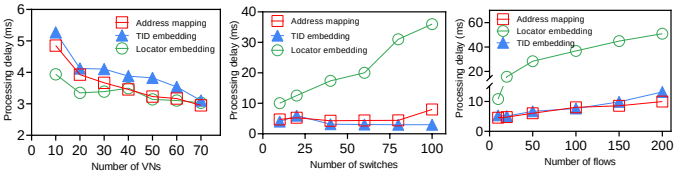
Metrics. We measure two metrics: message processing delay and resource consumption of NH. The message processing delay is presented for each control message type (e.g., `PACKETIN`, `FLOWMOD`, and `PACKETOUT`). For resource consumption, we measure the CPU cycle. CPU cycles are measured while control messages are processed.

[3]If the rules from ONOS match up to IP addresses, multiple flows from the same host pairs can be processed with the single flow rule.



(a) VN (b) switch (c) flow

Fig. 5: Message processing delay (PACKETIN).



(a) VN (b) switch (c) flow

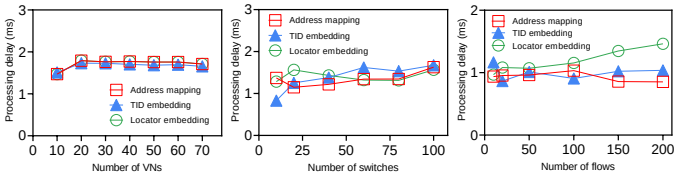
Fig. 6: Message processing delay (FLOWMOD).

Experimental variables. We use three variables in experiments: the number of physical switches, flows, and VNs, as shown in Table III. The range of each variable (e.g., 10–100 for switches, 10–200 for flows, and 10–70 for VNs) is determined under the condition where the CPU, NIC, and memory of the experiment machines do not become bottlenecks. Their ranges are wide and sufficient compared to those of previous studies. As shown in Table III, when each variable is changed, the other variables are fixed at a value. We set the inter-flow generation time to 1 s. To change the number of VNs, we evenly distribute the flows for all VNs. For example, when the number of VNs is seven, each VN processes ten flows. The VN topology is cloned into a PN topology.

Comparative groups. NHs differ from each other based on the address virtualization schemes (§II-B); thus, we compare the above metrics and variables for three SDN-NV comparative groups: 1) address mapping, 2) TID embedding, and 3) locator embedding. These three comparative groups represent the overheads of NH. In SDN-NV, NH is a standalone component between PN and VN control plane. Thus, the metrics, such as message processing delays and CPU cycles, are the overheads additionally added to the tenant's control plane due to the SDN-NV. The comparative groups are run by Libera because it has the implementation of three comparative groups. Note that for the experiments on NH, we do not measure overheads of the existing NV technologies because the NH is the component that only exists in the SDN-NV. Therefore, the results presented in this section are overheads of SDN-NV. There are several applications of SDN without NH (non-virtualized SDN) on existing NV technology to enhance network management [22]. The performance or overheads of the application can be found in previous studies that measured performance in non-virtualized SDN [23], [24], [25], [26].

B. NH Evaluation Results: Message Processing Delay

Figs. 5, 6, and 7 show the message processing delays of PACKETIN, FLOWMOD, and PACKETOUT messages. For each message, we analyze the delay by varying the experimental



(a) VN (b) switch (c) flow

Fig. 7: Message processing delay (PACKETOUT).

variables (x-axis), so the subfigure (a) is the result of changing the number of VNs, (b) switches, and (c) flows.

1) PACKETIN messages: In Fig. 5a, when the number of VNs (x-axis) increases, the message processing delays have little variance among the three comparative groups: 1.84 ms on average. Also, when the number of switches (Fig. 5b) increases, the delays are similar between comparative groups: 4.65 ms on average. When the number of flows increases (Fig. 5c), address mapping and TID embedding show approximately constant delays (1.02 ms on average), but locator embedding shows much longer delays (2× higher than both address mapping and TID embedding).

We analyze further details when the number of flows increases. The delay for PACKETIN depends on the number of host addresses because for each PACKETIN message, NH looks up the objects that represent the hosts: virtual addresses (address mapping), corresponding TID (TID embedding), or LITE values (locator embedding). Such lookup delay increases as the number of objects increases. However, according to Table III, the number of the total hosts is identical, which means that the delay is expected to be constant. That is consistent with what Fig. 5a and Fig. 5b show. However, for locator embedding, the delay increases up to 39% (when comparing 10 and 200 flows).

Regarding the locator embedding, we find that the semantics in matching fields of SDN controller and NH make the difference. The PACKETIN message triggers the flow rule installations for packet forwarding. The locator embedding in NH distinguishes the flow rules from the SDN controller by generating flowID, which is a hashed key for the addresses in the flow rules. However, the locator embedding creates flowID keys without transport layer addresses and is only based on the data link and IP layer addresses. Our SDN controller creates flow rules to match the port addresses of the transport layer. This leads to mismatching flow rule processing between the SDN controller and NH. In other words, flow rules are created from the SDN controller, but they cannot be interpreted correctly in NH. Thus, they are not installed correctly at the physical switches. Accordingly, the PACKETIN messages are generated repeatedly from the physical switches, which results in the high delay in Fig. 5c. We present the optimization results on this problem and show that the processing delay can be significantly reduced to a level similar to those of address mapping and locator embedding (§IV-D).

2) FLOWMOD messages: The FLOWMOD message (Fig. 6) is used for installing or modifying new flow rules. In Fig. 6a, when the number of VNs increases, the processing delay

decreases by up to 39%, 41%, and 23% for 10 and 70 VNs, respectively, for address mapping, TID embedding, and locator embedding[4]. This decrease is somewhat surprising because the bottleneck for NH and message processing delays could increase as the number of VNs. This experiment sets the number of flows to 70 (Table III). Thus, as the number of VNs increases, the number of flows that each VN processes becomes smaller because 70 flows are distributed equally to each VN. Accordingly, the flow rules for each virtual switch are reduced, which contributes to reducing the delay of NH.

In Fig. 6b, when the number of switches increases, address mapping and TID embedding show relatively constant delay—5.2 ms and 3.7 ms on average, respectively. The delays of the two comparative groups do not increase because NH we use (Libera) parallelizes the flow rule processing. NH handles each flow rule with a separate thread. Therefore, as long as CPU is not the bottleneck, flow rule processing is not affected. However, in the case of locator embedding, the delay increases linearly—up to 3.57× when the number of switches grows 10 to 100. This is because locator embedding processes flow rules when a set of rules composing a path is all arrived at NH; thus, the delay increases as the number of switches composing a path increases.

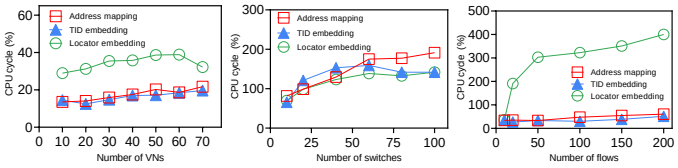
In Fig 6c, as the number of flows increases, the processing delays of all comparative groups increase—from 10 to 200 flows, 2.19×, 2.47×, and 4.69× times, respectively, for experimental variables. The experiments use a single VN with 20 switches (Table III). The reason for the increase is the increased flow rules in the NH with the number of flows, which results in the delay increase.

3) `PACKETOUT` messages: The `PACKETOUT` message is used to inject packets into the network in order to deliver data packets to the destination host. During the flow rule installation, the packet forwarding in data plane can pause when network switches do not have flow rules; so, through `PACKETOUT`, NH tries to reduce such pause in packet forwarding. In Fig. 7, for all varying experimental variables (i.e., VNs, switches, and flows), the delays are quite constant for address mapping and TID embedding. For locator embedding, the delays are constant for the increasing number of VNs and switches. However, when the number of flows increases, locator embedding shows the delay increased up to 1.5 ms (50% higher than that of the 10 flows). The result comes from the semantic difference problem of locator embedding described above.

C. NH Evaluation Results: CPU Cycle

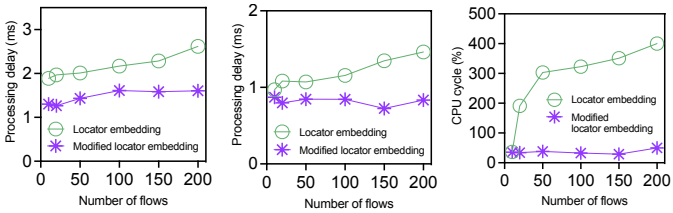
Fig. 8 shows the CPU cycles consumed in NH when the three messages are being processed. The CPU cycles are measured along with the experiments for Figs. 5, 6, and 7. When the number of VNs changes (Fig. 8a), the cycles of address mapping and TID embedding are similar: 17.4% and

[4]Note that locator embedding does not immediately process individual flow rules independently and install them at once after gathering all flow rules constituting a path (§II-B). Thus, the delay for locator embedding includes installing a set of flow rules for composing a path.



(a) VN (b) switch (c) flow

Fig. 8: CPU cycle of NH.



(a) `PACKETIN` (b) `PACKETOUT` (c) CPU cycle

Fig. 9: Optimized locator embedding.

16.26% on average, respectively. Locator embedding shows 34.4% average CPU cycles, much higher than the other two. It is because locator embedding has to track the source and destination hosts and their attached switches to generate LITE values (§II-B).

In Fig. 8b, when the number of switches increases from 10 to 60, all three comparative groups show increasing CPU cycles (2.15×, 2.43×, and 1.98× higher for address mapping, TID embedding, and locator embedding, respectively). However, when the number of switches increases from 80 to 100, the comparative groups show relatively constant CPU cycles—1.81 cores, 1.48 cores, and 1.37 cores, on average, for address mapping, TID embedding, and locator embedding.

The main reason for the constant CPU cycles is the topology discovery where NH regularly updates the topology (e.g., link connections between switches and hosts) through echoing switches. For echoing, NH and switches exchange a small size of TCP packets (e.g., 190 bytes on average) in a very short time (mostly within 50–110 ms). In cloud computing, this kind of network communication is known as "short-lived TCP," and it is well-known that networking stacks of operating systems become bottlenecked with the high number of short-lived TCP packets [27]. In our experiments, 60 switches used in the experiments cause the bottlenecks on the networking stack, so the CPU cycles of NH is dominated by the underlining kernel, not NH itself. Thus, the CPU cycles of NH become constant values from the 60 switches.

In Fig. 8c, the CPU cycles increase with the number of flows. The CPU cycles of 10 and 200 flows increase by 0.8×, 0.5×, and 10×, respectively, for address mapping, TID embedding, and locator embedding. Note that the high increase of locator embedding is due to the flow rule semantic difference.

D. Optimization on Locator Embedding

Herein, we introduce optimization on locator embedding that solves its semantics problem (§IV-B1). We modify the flowID implementation of Libera's locator embedding so that

the new implementation includes port addresses (transport layer) in addition to MAC and IP addresses. The modifications include additional improvements, such as the elimination of redundant table lookup and unnecessary loggings. Except for the flowID creation, other operations in Libera (such as managing ARP and physical flow rule creation) are based on IP addresses without considering port addresses. We do not change any operation of Libera, but only modify the code for handling flowID.

Fig. 9 shows the enhanced results on locator embedding. Fig. 9a, Fig. 9b, and Fig. 9c show `PACKETIN` processing delay, `PACKETOUT` processing delay , and CPU cycles. The line with circles represents the evaluation results with the old locator embedding, and the line with * marks represents the ones with our newly modified locator embedding. In Fig. 9a and Fig. 9b, `PACKETIN` and `PACKETOUT` delays of the new locator embedding show relatively constant values (1.47 ms and 0.82 ms, respectively, on average) while `PACKETIN` and `PACKETOUT` delays of the old locator embedding continuously increases. Specifically, the new locator embedding improves the processing delay of `PACKETIN` and `PACKETOUT` by about 32% and 31% on average, respectively which becomes similar to that of address mapping and TID embedding. In Fig. 9c, the CPU cycle of the new locator embedding shows constant values (36.5% on average), which is 7.3× lower than existing locator embedding on average and similar to address mapping TID embedding.

V. CONCLUSION

The goal of this paper is to investigate the performance and overhead of SDN-NV comprehensively. We run a large number of benchmarks to evaluate SDN-NV with varying numbers of switches, flows, and VNs. Then, we compare the results with existing NV technologies. Contrary to the popular belief that SDN-NV has little practical value due to its overheads, this paper reports that in the data plane performance, SDN-NV is competitive over the existing NV technologies like overlay, and sometimes it is even superior to them. In addition, even when the number of VNs, switches, or flows increases, our results show that the overheads of NHs mostly remain constant. We hope our benchmarking results make a case for SDN-NV to be adopted in cloud datacenters.

REFERENCES

[1] G. Yang, B.-y. Yu, H. Jin, and C. Yoo, "Libera for programmable network virtualization," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 38–44, 2020.

[2] 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, 2015.

[3] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2012.

[4] X. Jin, J. Gossels, J. Rexford, and D. Walker, "Covisor: A compositional hypervisor for software-defined networks," in *12th USENIX Symposium on Networked Systems Design and Implementation*, 2015, pp. 87–101.

[5] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make your virtual SDNs programmable," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 25–30.

[6] G. Yang, B.-y. Yu, W. Jeong, and C. Yoo, "FlowVirt: Flow rule virtualization for dynamic scalability of programmable network virtualization," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 350–358.

[7] Y. Yoo, G. Yang, M. Kang, and C. Yoo, "Adaptive control channel traffic shaping for virtualized SDN in clouds," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 22–24.

[8] G. Yang, B.-Y. Yu, S.-M. Kim, and C. Yoo, "LiteVisor: A network hypervisor to support flow aggregation and seamless network reconfiguration for vm migration in virtualized software-defined networks," *IEEE Access*, vol. 6, pp. 65 945–65 959, 2018.

[9] G. Yang, Y. Yoo, M. Kang, H. Jin, and C. Yoo, "Bandwidth isolation guarantee for SDN virtual networks," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS operating systems review*, vol. 37, no. 5, pp. 164–177, 2003.

[11] K. Suo, Y. Zhao, W. Chen, and J. Rao, "An analysis and empirical study of container networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 189–197.

[12] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: a state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2017.

[13] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar, "Can the production network be the testbed?" in *OSDI*, vol. 10, 2010, pp. 1–6.

[14] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of network and computer applications*, vol. 52, pp. 11–25, 2015.

[15] A. Blenk, A. Basta, W. Kellerer, and S. Schmid, "On the impact of the network hypervisor on virtual network performance," in *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 2019, pp. 1–9.

[16] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "PIAS: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, 2017.

[17] S. Choi, B. Burkov, A. Eckert, T. Fang, S. Kazemkhani, R. Sherwood, Y. Zhang, and H. Zeng, "FBOSS: building switch software at scale," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 342–356.

[18] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM 2011 Conference*, 2011.

[19] Y. Zhao, N. Xia, C. Tian, B. Li, Y. Tang, Y. Wang, G. Zhang, R. Li, and A. X. Liu, "Performance of container networking technologies," in *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*, 2017, pp. 1–6.

[20] D. Zhuo, K. Zhang, Y. Zhu, H. H. Liu, M. Rockett, A. Krishnamurthy, and T. Anderson, "Slim: OS kernel support for a low-overhead container overlay network," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 331–344.

[21] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A comparative analysis of data center network architectures," in *2014 IEEE International Conference on Communications*. IEEE, 2014, pp. 3106–3111.

[22] F. Foresta, W. Cerroni, L. Foschini, G. Davoli, C. Contoli, A. Corradi, and F. Callegati, "Improving openstack networking: Advantages and performance of native sdn integration," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[23] B.-y. Yu, G. Yang, and C. Yoo, "Comprehensive prediction models of control traffic for SDN controllers," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 262–266.

[24] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone, "Evaluating the SDN control traffic in large ISP networks," in *2015 IEEE International Conference on Communications*, 2015, pp. 5248–5253.

[25] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, 2017.

[26] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of ONOS reactive forwarding applications in ISP networks," *Computer Communications*, vol. 102, pp. 130–138, 2017.

[27] K. Yasukata, M. Honda, D. Santry, and L. Eggert, "StackMap: Low-latency networking with the OS stack and dedicated nics," in *2016 USENIX Annual Technical Conference (ATC 16)*, 2016, pp. 43–56.