Accurate and Efficient Monitoring for Virtualized SDN in Clouds

Gyeongsik Yang, *Member, IEEE,* Yeonho Yoo, Minkoo Kang, Heesang Jin, and Chuck Yoo, *Member, IEEE*

Abstract—This paper presents V-Sight, a network monitoring framework for programmable virtual networks in clouds. Network virtualization based on software-defined networking (SDN-NV) in clouds makes it possible to realize programmable virtual networks; consequently, this technology offers many benefits to cloud services for tenants. However, to the best of our knowledge, network monitoring, which is a prerequisite for managing and optimizing virtual networks, has not been investigated in the context of SDN-NV systems. As the first framework for network monitoring in SDN-NV, we identify three challenges: non-isolated and inaccurate statistics, high monitoring delay, and excessive control channel consumption for gathering statistics. To address these challenges, V-Sight introduces three key mechanisms: 1) statistics virtualization for isolated statistics, 2) transmission disaggregation for reduced transmission delay, and 3) pCollector aggregation for efficient control channel consumption. The evaluation results reveal that V-Sight successfully provides accurate and isolated statistics while reducing the monitoring delay and control channel consumption in orders of magnitude. We also show that V-Sight can achieve a data plane throughput close to that of non-virtualized SDN.

Index Terms—Distributed systems, Network management, Network monitoring

1 Introduction

Network virtualization (NV) is a vital technology in datacenters [2]. NV creates virtual networks (VNs) for tenants based on a single physical network infrastructure and isolates network traffic between the VNs. Because tenants require isolated network connections between their computing nodes, such as virtual machines (VMs) and containers, NV is widely deployed in cloud datacenters [3], [4]. To implement NV, overlay networking of TCP/IP network stacks is commonly used. Overlay networking distinguishes the packets of multiple tenants with a tenant identifier (TID) attached as an additional encapsulated header.

However, overlay networking has a critical shortcoming—it does not allow tenants to configure or program their VNs [5] because the underlying network resources (e.g., switches, ports, and links) are solely determined by datacenter operators. Therefore, tenants cannot install their desired network policies (e.g., flow entries for packet forwarding or redirection to a proxy) in an arbitrary VN switch. In addition, tenants cannot create a VN topology between their VMs or containers as required.

This limitation translates into a severe problem because many applications demand in-network optimizations (e.g., OpenFlow [6] and P4 [7]) or their own network architectures (e.g., information-centric networking [8]), which requires programmable networks. Their objective is to enhance the service quality of the applications. However, due to the restricted programmability of overlay networking, such optimizations are hindered in clouds [5]. Consequently, programmable VNs have been identified as a critical missing component for NV [9], [10].

Fortunately, software-defined networking (SDN) provides a new path for NV [5]. SDN is a network system structure that splits the network control and packet forwarding functionalities. SDN centralizes the network control functions into software (SDN controller). Because multiple tenants are present in clouds, each tenant can have its own SDN controller (tenant controller)[1], which leads to SDN-based NV (SDN-NV). One of the SDN-NV architectures utilizes the network hypervisor [5], [11], [12], [13] that sits between the physical network and the tenant controllers.

Network hypervisors support VN abstractions, such as virtual switches, links, ports [11], and addresses [13]. With the abstractions provided, SDN-NV can provide programmability to tenants [13]. In other words, each tenant can have a virtualized SDN so that it can create its own VN topology and program its VN using SDN controllers (e.g., POX [14], ONOS [15], or OpenDayLight [16]). There have been advances in network hypervisor technology that enhance the scalability [17], [18] and flexibility [19].

Nevertheless, to the best of our knowledge, no study has discussed network monitoring for SDN-NV (details in §2.4). Network monitoring is a vital prerequisite for VN management in providing statistics. For example, gathering the processed volume of traffic for each flow entry or port is a basis of link utilization for network management, such as QoS routing, network planning, and anomaly detection

- *Gyeongsik Yang, Yeonho Yoo, Minkoo Kang, and Chuck Yoo are with the Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea, 02841. E-mail: ksyang@os.korea.ac.kr, yhyoo@os.korea.ac.kr, mkkang@os.korea.ac.kr, chuckyoo@os.korea.ac.kr. (Corresponding author: Chuck Yoo.)*
- *Heesang Jin is with the Blockchain Research Section, Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea, 34129. This work was performed when Heesang Jin was a graduate student at Korea University. E-mail: jinhs@etri.re.kr.*
- *A preliminary version of this paper appeared in the proceedings of IEEE INFOCOM 2020 - IEEE Conference on Computer Communications [1].*

Manuscript received XX; revised XX.

1. Typically, SDN controllers (e.g., POX, ONOS, OpenDayLight) are used as tenant controllers. Thus, throughout this paper, we use the terms "SDN controller" and "tenant controller" interchangeably. We use the term "SDN controller" for the context of non-virtualized SDN, and the term "tenant controller" for SDN-based NV

[20], [21], [22], [23], [24]. Specifically, a research paper of Microsoft has reported that despite the significant difficulties associated with VN monitoring, monitoring of VNs is necessary because it is key to handling faults in network infrastructures and providing performant services to tenants by detecting overheads for each tenant [25]. Despite its such conspicuous importance, network monitoring has received relatively little attention in SDN-NV studies.

To address such problems, this paper presents V-Sight, a comprehensive network monitoring framework for SDN-NV. As the first framework for network monitoring, V-Sight faces three main challenges: 1) inaccurate statistics, 2) high monitoring delay, and 3) excessive control channel traffic consumption. First, because tenant controllers attempt to optimize and manage their VNs based on statistics (e.g., the volume of traffic processed by a flow entry for routing), accurate statistics should be provided. In SDN-NV, the statistics collected in the physical network are the aggregate of multiple VNs running on the network, but there is no mechanism that isolates the statistics for each VN.

Second, the SDN-NV system inevitably increases the delay (so-called transmission delay) between the statistics request from a tenant controller and the reply from switches. When the statistics request message arrives at the network hypervisor from the controller, the network hypervisor must send the corresponding network statistics request messages to the physical network (switches) and wait to receive the results, thus increasing the transmission delay. For example, if a tenant controller sends a request for "all flow entries of a virtual switch," the transmission delay can be high because the individual flow entries' statistics are collected sequentially. Our experiment shows that the transmission delay increases by up to 333 times compared with that of a non-virtualized SDN (§2.3.2). The increased delay causes the collected statistics to be out-of-date; thus, a careful design to reduce such delay is required.

Third, the network hypervisor consumes control channel traffic excessively compared with a non-virtualized SDN. In our experiment, control channel consumption increases by up to three times (§2.3.3) when the tenant controller asks for the statistics of all the flow entries per switch. This high consumption is because the network hypervisor has to send multiple messages to switches. Considering that such messages go through the control channel, other traffic is affected [26]. For instance, our experiment finds that the flow entry installation time increases by 4.3 times due to the control channel consumption in retrieving the statistics.

V-Sight addresses the above challenges through three key mechanisms: 1) statistics virtualization to isolate statistics per VN, 2) transmission disaggregation to reduce transmission delay, and 3) pCollector aggregation to reduce control channel consumption. Statistics virtualization (§3.2) isolates the virtual network statistics (vStatistics) per VN from physical network statistics (pStatistics). Transmission disaggregation (§3.3) uses caching of frequently used pStatistics. The caching is performed by a pCollector that retrieves the pStatistics routinely and stores the data in the network hypervisor, which removes the delays for pStatistics transmission. Further, we design pCollector aggregation (§3.4) to reduce the control channel consumption of the pCollector. Instead of collecting the pStatistics from individual pCollectors, pCollector aggregation attempts to merge the pCollectors to ensure that multiple pStatistics are retrieved with a single request message, which reduces the number of messages and thus control channel consumptions.

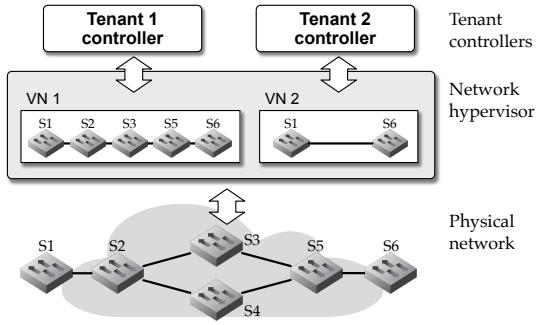


Fig. 1. SDN-based network virtualization.

In short, this paper accomplishes the followings:

- Identification and formulation of three key challenges for network monitoring in SDN-NV systems: statistics isolation, monitoring delay, and control channel consumption for network hypervisors.
- Introduction of the new concepts, namely, statistics virtualization, transmission disaggregation, and pCollector aggregation.
- Full system implementation of the framework as an open-source software.
- Comprehensive experiments that result in 1) improvement in vStatistics accuracy by three orders of magnitude, 2) up to 454 times reduction in transmission delay, 3) up to 1.9 times improvement in control channel consumptions, and 4) 5.5 times variance improvement in TCP throughput in a practical usage scenario.

The remainder of this paper is organized as follows. §2 describes the background and challenges of network monitoring in SDN-NV. §3 provides the fundamental concepts and the complete design of V-Sight, and §4 presents the evaluation results. §2.4 elaborates on related work, and §5 discusses future research directions. Finally, §6 concludes this paper.

2 Background and Motivation

Here, we explain the background of this study: SDN-NV and network monitoring. Then, we identify challenges for the network monitoring framework in SDN-NV systems. In addition, we comprehensively explain the related work and the differences of this study.

2.1 SDN-based Network Virtualization

SDN-NV comprises three layers (Fig. 1), namely, tenant controllers, network hypervisor, and physical network (PN). A tenant refers to a user or a group of users who share the authority for using the given resources provided by a cloud. We denote a physical network connecting the servers of a datacenter as PN. Based on the PN, each tenant provides VN, which stands for virtual networks over PN.

A tenant controller can create its VN's topology with VN resources, such as virtual switches, links, and ports, when

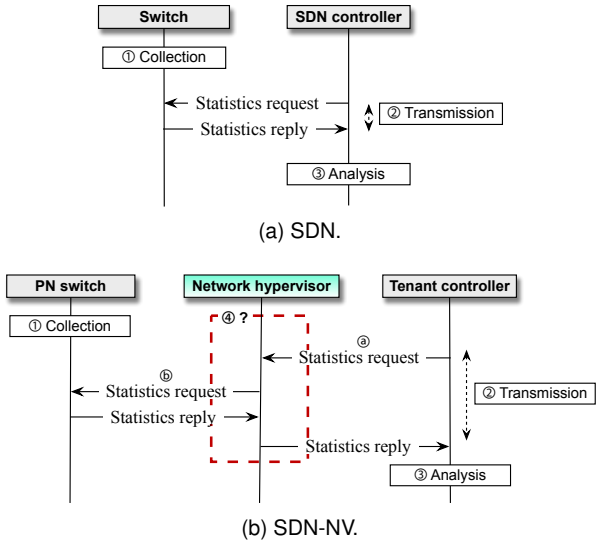


Fig. 2. Steps of network monitoring.

the tenant controller sends a request to the network hypervisor. When the network hypervisor receives the request, it substantiates the VN resources with mappings to the PN resources. For instance, a virtual switch operates based on the mapping of one physical switch or a set of physical switches and links. The virtual port (vp) for each virtual switch is also mapped to the physical port (pp). In addition, a virtual link can be created by connecting two vps.

After the VN topology is created, the network hypervisor emulates the requested VN resources as they are standard SDN switches. The tenant controller then manages the created VN resources without recognizing whether its resources are virtualized or not. The tenant controller connects to the virtual switches through south-bound interfaces (e.g., OpenFlow) and implements flow entries that match packets to ensure that it can process (e.g., forward) the matched packets. These operations are achieved by control messages from the tenant controller, and the messages pass through the control channel.

VN resources and flow entries are mapped to the corresponding resources in the PN, which implies that the PN resources can be mapped to either one or more of the VN resources. Thus, the flow entries from multiple tenant controllers can be mapped to a smaller number of physical flow entries [17], [18]. Throughout this paper, the term V represents a virtualization function, and V′ represents a de-virtualization function—these functions map PN resources to the VN resource, or vice versa. For example, when a physical flow entry (pf) is given, V(pf) provides the virtual flow entries (vfs) mapped to the pf. Similarly, given a virtual switch S, V′(S) generates the list of physical switches and links mapped to S.

2.2 Network Monitoring

2.2.1 Network monitoring in SDN

Network monitoring in SDN involves three steps (Fig. 2a): ① collection, ② transmission, and ③ analysis [27]. The statistics are recorded at the switches, which measure the processed number of packets per flow entry or port (① collection). We denote the statistics of a network resource by the notation S. For example, $S(pf_i)$ and $S(pp_j)$ represent the statistics of pf_i and pp_j. An SDN controller then gathers the statistics from a switch (② transmission). With the collected information, the SDN controller analyzes, manages, and optimizes the networks (③ analysis).

We additionally explain the sizes of statistics request and reply messages. Both messages consist of a packet header, usually an Ethernet, IP, TCP, or OpenFlow header, and payloads. The header sizes of requests and replies are similar ($l(H)$). A request's payload is the network resource to be monitored. For example, in case of flow entry, the IP addresses or actions that the entry performs are included. The reply's payload includes the same network resource and its statistics. The size of the network resource i included in both request and reply payloads is $l(I(i))$, and the size of actual statistics included in the reply payload is $l(S(i))$.

For example, in SDN, an ONOS controller [15] sends statistics requests messages toward flow entries and ports of a switch every 5 s as its default settings. The statistics transmission process is finished when the SDN controller receives the corresponding reply messages from the switches, and the transmission time can be longer or shorter than 5 s. Note that the statistics request sending interval can be changed by a network operator.

2.2.2 Network monitoring in SDN-NV

Fig. 2b shows the network monitoring in SDN-NV. In SDN-NV, the switches in PN collect $S(pf)$ and $S(pp)$ (① in Fig.2b), similar to the collection step of SDN (① in Fig.2a). In SDN-NV, the tenant controllers perform the transmission process (② in Fig.2a). However, the difference is that as the controllers face the virtual switches (which is a network hypervisor), the statistics request is delivered to the network hypervisor instead of PN switches (ⓐ in Fig. 2b. The network hypervisor then appropriately handles the request and generates a reply for the request. For example, the network hypervisor should collect the statistics corresponding to the request from the PN switches ((ⓑ in Fig. 2b, which may be multiple transmissions according to VN and PN mappings) and generates a reply message based on the collected statistics. However, to the best of our knowledge, existing NHs lack an appropriate scheme for handling such requests from tenant controllers (④ in Fig. 2b, details in §2.4.1), resulting in critical challenges (discussed in §2.3).

For further discussions, we formalize several notations for network monitoring in SDN-NV. First, the transmission time for the statistics request and reply messages between the tenant controller and the network hypervisor is denoted as d_v. This time is identical to the round-trip-time (RTT) between the tenant controller and network hypervisor. The transmission time for the statistics request and reply message between the network hypervisor and PN switches is denoted as d_p. In addition, the processing time in the network hypervisor for calculating vStatistics and generating the reply message for tenant controllers is denoted as d_{NH}. Table 1 summarizes the terminologies explained up to now.

2.3 Challenges of Network Monitoring for SDN-NV

Here, we discuss the three network monitoring challenges in SDN-NV systems in detail, which motivates the development of V-Sight. The challenges here represent the

TABLE 1
Terminologies and their description.

Terminology	Description
pStatistics vStatistics	Statistics of physical network resource Statistics of virtual network resource
Tenant	A user or group of users sharing the resources provided by a cloud
PN	Datacenter network connecting physical servers for VMs and containers
VN	An isolated logical network given for a tenant
pf, pp	Physical flow entry and physical port
$V(i)$ $V'(j)$	VN resources mapped to PN resource i PN resources mapped to VN resource j
$S(i)$	Statistics of the network resource i
$l(H)$ $l(I(i)), l(S(i))$	Header length of statistics request/reply Length of information, statistics of i in payload
d_v	Transmission time of vStatistics request or reply message
d_p	Transmission time of pStatistics request or reply message
d_{NH}	Processing time of network hypervisor for vStatistics

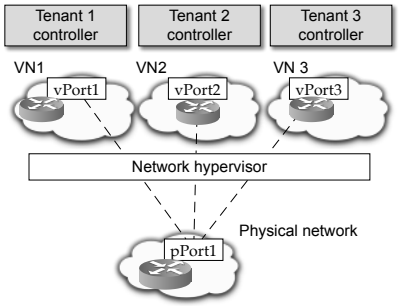


Fig. 3. Non-isolated statistics example.

problems that V-Sight should overcome as the first network monitoring framework for SDN-NV systems.

2.3.1 Non-isolated and inaccurate statistics

In SDN-NV, the PN resources (e.g., switches and ports) are shared among multiple VNs. Thus, the collected statistics from PN resources are not isolated between the VNs. The pStatistics collected in PN switches can be expressed as follows. For the PN resources i (e.g., pf or pp), $S(i) = \sum_{j \in V(i)} S(j)$. Fig. 3 shows an example of three VNs, each comprising one vp. In this scenario, all vps are mapped to the same pp (pPort1). Suppose that the tenant1 controller retrieves the statistics of vPort1. Because pPort1 is unaware of the presence of multiple VNs, $S(vPort1)$ collected in PN is the sum of $S(vPort1) + S(vPort2) + S(vPort3)$. This indicates that pStatistics does not separate the statistics per VN. Thus, the tenant1 controller ends up with aggregated statistics which is inaccurate for VN1.

Statistics are used for various network management operations of tenant controllers, such as cost-based central routing, traffic engineering, and QoS. However, with non-isolated statistics, tenant controllers cannot accomplish their desired management operations. Thus, V-Sight should be capable of isolating statistics in the sense that the statistics provided to each tenant should only contain information regarding a particular VN, not the aggregated statistics.

2.3.2 High transmission delay

Network monitoring is performed repeatedly to track the changing statistics. A reply to a statistics request should arrive as quickly as possible because any transmission delay between the request and reply messages distances the value of the statistics from the request time.

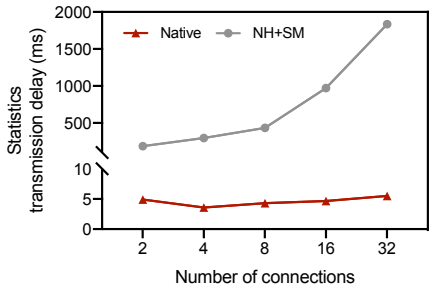


Fig. 4. Statistics transmission delay comparison (ms).

We conduct an experiment to determine the increase in transmission delay. Existing network hypervisors do not support network monitoring; thus, we implement a simple monitoring function on Libera [5], which is an open-source network hypervisor. The implementation receives the statistics requests from tenant controllers and then gathers the corresponding statistics from the PN based on the mappings between the VNs and the PN. The monitoring function replies to the tenant controllers after all pStatistics from the physical switches arrive. We call evaluations performed using this implementation, NH+SM. The experiment is conducted in a 4-ary fat-tree topology with 2, 4, 8, 16, and 32 TCP connections with one VN. The tenant controller issues statistics requests at 5 s intervals for every switch in its network, requesting the statistics of all flow entries of each switch. As described in Fig. 4, the non-virtualized SDN (Native) case exhibits almost constant statistics transmission delays, at 4.6 ms on average, regardless of the number of network connections. In contrast, NH+SM exhibits delays of 187 to 1,836 ms, which are 38 to 333 times higher than that of Native.

We formulate the transmission delay of NH+SM and NH to determine the reason of the increased delay by using the notations introduced in Table 1. For a request message from a tenant controller that retrieves the statistics of all flow entries of a virtual switch, we refer to the number of flow entries of the switch as n. Then, the transmission delay of NH+SM is formulated as $d_v + nd_p + d_{NH}$, which is the sum of the following instances: 1) one d_v vStatistics transmission, 2) n instances of pStatistics transmissions, and 3) one instance of processing in the network hypervisor for a vStatistics calculation and reply message creation (d_{NH}).

On the other hand, the total transmission delay in the Native case is d_c because a single statistics transmission between the PN and the SDN controller can retrieve all existing flow entries of a switch. Note that NH+SM cannot retrieve n numbers of pfs in a single transmission because NH+SM collects only the pfs mapped to the tenant and the PN switches contain the pfs of other tenants at the same time. Consequently, the transmission delay of NH+SM includes the additional time of $nd_p + d_{NH}$, which increases this transmission delay by up to 1.84 s (Fig. 4).

2.3.3 Excessive control channel consumption

Statistics transmission passes through the control channel. In SDN-NV, two types of control channels exist: between the network hypervisor and tenant controllers, and between the

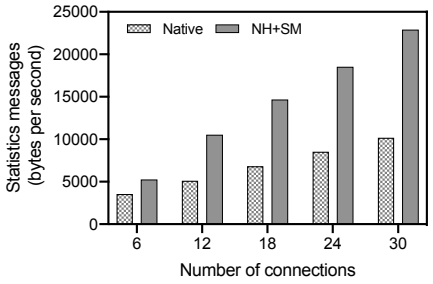


Fig. 5. Control channel consumption comparison (bytes per second).

network hypervisor and physical switches. In this study, we focus on the latter because the traffic between the network hypervisor and physical switches is increased by network virtualization. A network hypervisor emulates ordinary switches with virtual switches so that SDN controllers can be used as tenant controllers without any further modification [5]. Thus, the traffic between the network hypervisor and tenant controllers is similar to the control traffic in non-virtualized SDN. Therefore, this study presents and aims to reduce the control traffic increased by network virtualization, which is the traffic between the hypervisor and physical switches.

The control channel is utilized by tenant controllers for control operations, such as switch connection handshaking, flow entry installation and modification, the topology discovery process, and ARP processing. Thus, when the control channel consumption for statistics increases from 5.11 to 22 KB/s, we find that the flow entry installation suffers a four times higher delay (from 86 to 368 ms). Moreover, because operations such as flow entry installation can reduce the throughput of network connections, the control channel consumption for network monitoring should be reduced.

To be precise, we evaluate the control channel consumption for the network monitoring of NH+SM. We set a linear topology with five switches and three VNs. Each VN consists of two hosts at the edge of the topology with 6, 12, 18, 24, and 30 network connections in PN. We conduct experiments with the same monitoring function as §2.3.2. Fig. 5 shows the control channel consumption for the flow statistics transmission. The results for NH+SM are 1.5 to 2.3 times higher than those of Native.

In NH+SM, a network hypervisor collects the statistics for a request as follows. It first checks the existing vfs in the requested switch. Let us denote a set of vfs in the requested virtual switch as F_{vf}. For each element of j in F_{vf}, the network hypervisor finds the mapped pfs to j and sends the statistics request messages one-by-one. The reply message arrives at the network hypervisor for each request message. The control channel consumption from network monitoring, which is the total of request and reply messages, is thus formulated by the sum of the size of total request messages $\sum_{j \in F_{vf}} \sum_{k \in V'(j)} l(H) + l(I(k))$ and the size of total reply messages $\sum_{j \in F_{vf}} \sum_{k \in V'(j)} l(H) + l(I(k)) + l(S(k))$.

On the other hand, in Native, SDN controllers can collect all pfs existing in a switch through a single request and reply by assigning "all entry" as the payload of the request message. We denote the payload size for "all flow entry" message as $l(*_{vf})$; then, the size of the request message becomes $l(H)+l(*_{vf})$. When a set of pfs in the physical switch is denoted by F_{pf}, the size of the reply message becomes $\sum_{i \in F_{vf}} l(H) + l(I(i)) + l(S(i))$. Thus, control channel consumption, which is the sum of request and reply messages, is quite higher in NH+SM than in Native.

2.4 Related Work

In this section, we first explain the existing studies on network hypervisors and their consideration in network monitoring. Then, we review the existing studies on network monitoring in non-virtualized SDN and summarize the differences of V-Sight compared with them.

2.4.1 Related studies on NH and monitoring

Table 2 presents the descriptions and objectives of existing network hypervisors. FlowVisor [11] introduced the first idea of NV in SDN, and FlowN [12] defined abstractions for virtual networks, such as virtual addresses, based on containers. OpenVirteX [13] defined address virtualization schemes based on mapping between virtual and physical addresses, which can provide full address field accesses to tenants. AutoSlice [28] and AutoVFlow [29] proposed a distributed network hypervisor to improve the platform scalability. Also, CoVisor [31] designed a policy composition framework for a network to be managed using heterogeneous SDN controllers. Libera [5] defined a cloud-service model based on the SDN-NV system.

In addition, FlowVirt [17] proposed flow entry virtualization, which maps multiple physical flow entries to virtual flow entries, thereby reducing the amount of switch memory. LiteVisor [18] proposed a new packet forwarding scheme named LITE, which separates the address, location identifier, and tenant identifier to effectively manage and update information in a datacenter. TeaVisor [32] proposed path virtualization, which ensures provision of the requested bandwidth of each tenant by leveraging multipath routing, bandwidth reservation, and bandwidth limiting.

The above studies improve different aspects of SDN-NV systems to make the system feasible and reliable in cloud computing. Including the content of the survey papers on NV [33], [34], [35], however, we find that studies on network hypervisors do not cover network monitoring for tenants (i.e., non-isolated statistics, transmission delay, and control channel consumption). In addition, Microsoft has mentioned that the monitoring of the virtualized networks in cloud systems has not been investigated [25]. Thus, to the best of our knowledge, no previous study has focused on network monitoring for tenants, and this fact motivates us to develop V-Sight.

2.4.2 Related studies on monitoring in non-virtualized SDN

The SDN controllers, used as tenant controllers, provide APIs or sub-modules for network monitoring (e.g., *fwd* in ONOS or OpenFlow Plugin in OpenDayLight). Such tools are used for creating statistics request messages and processing the reply messages to be received according to the request. To work with such tools and physical networks, we aim to design V-Sight to generate proper statistics reply messages containing the isolated statistics, with reasonable transmission delay and control channel consumptions.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3089225

TABLE 2
Related studies analysis—network hypervisor.

Study	Description	Monitoring challenges in SDN-NV		
		Statistics isolation	High transmission delay	Excessive control channel consumption
FlowVisor [11]	Divide network resources such as address and topology and allocate them to tenants	Not solved	Not solved	Not solved
FlowN [12]	Design an address virtualization scheme (based on FlowN) and container-based tenant controller architecture	Not solved	Not solved	Not solved
OpenVirteX [13]	Design address virtualization that provides an access to entire address fields	Not solved	Not solved	Not solved
AutoSlice [28]	Design network hypervisor as a distributed system (multiple proxies)	Not solved	Not solved	Not solved
AutoVFlow [29]	In addition to AutoSlice, AutoVFlow enables entire address space to tenants	Not solved	Not solved	Not solved
HyperFlex [30]	Disaggregate the functions of an internal network hypervisor to flexibly locate them	Not solved	Not solved	Not solved
CoVisor [31]	Design composition policies for flow entries coming from various SDN controllers	Not solved	Not solved	Not solved
Libera [5]	Define the architecture, APIs, and essential operations of SDN-NV for cloud datacenters, summarized as Libera	Not solved	Not solved	Not solved
FlowVirt [17]	Aggregate multiple flow entries of tenants into a smaller number of physical ones	Not solved	Not solved	Not solved
LiteVisor [18]	Suggest a routing scheme for separating the location, identifier, and tenant distinguisher	Not solved	Not solved	Not solved
TeaVisor [32]	Design path virtualization, a customized multipath routing, bandwidth reservation, and bandwidth limiting, for SDN-NV	Not solved	Not solved	Not solved
V-Sight	Provide isolated statistics and reduce statistics transmission delay and control channel consumption	**Solved** (§3.2)	**Solved** (§3.3)	**Solved** (§3.4)

TABLE 3
Related studies analysis—network monitoring in non-virtualized SDN.

	Designs					Evaluation methodology	
	New switch architecture	New API	Sampling	Adaptive interval	Others	Implementation	Network topology/trace
OpenSketch [36]	✓	✓				Switch: NetFPGA-based HW Controller: C++ SW	CAIDA packet trace or single switch
SDN-Mon [37]	✓	✓				Switch: Lagopus-based SW Controller: Module on Ryu	Single switch
OpenSample [21]			✓(packet)			Module on Floodlight	4-ary fat-tree and four switches
OpenTM [22]			✓(flow)			Module on NOX	Custom linear topology (ten switches)
FlowCover [23]			✓(switch)			Simulator	Erdős–Rényi graph, Waxman graph
OpenNetMon [38]			✓(flow)			Module on POX	Linear topology (four switches)
cFlow [20]			✓(flow)			Simulator	GEANT trace
Tahaei et. al. [39]			✓(switch)	✓(link utilization)		Module on Floodlight	Fat-tree topology
PayLess [24]				✓(link utilization)		Module on Floodlight	Tree topology
IPro [40]				✓(reinforcement learning)		SW based on Ryu API	Custom tree topology (11 switches)
MicroTE [41]					Split controller and monitoring framework	Kernel module on Linux	Tree topology
FlowSense [42]		✓			Embed statistics in other messages	NA Simulator	Linear topology (two switches) EDU1 trace
V-Sight	**Not relevant (can work together)**					Internal scheme in Libera	Linear (five switches) and 4-ary fat-tree topology

Various studies have been proposed to reduce monitoring overheads in non-virtualized SDN. Table 3 summarizes these studies by comparing their key designs and evaluation methodology. The objectives of these studies are mostly to reduce the monitoring overheads between SDN controllers and switches while maintaining a degree of statistics accuracy. We explain these studies comprehensively here.

First, OpenSketch [36] and SDN-Mon [37] introduced new monitoring architectures to reduce monitoring overheads on both the switch and controller sides; thus, they require architecture modification on switches and API modification on SDN controllers. For example, OpenSketch designs a hash-based architecture that collects statistics based on the hash result of each flow. The memory for collection statistics in switches reduces, and the traffic or delay in statistic transmission decreases accordingly. SDN-Mon introduced a switch architecture that separates the flow table for packet routing and statistics collection so that the collection in switches can be performed with different granularities from flow entries. Because statistics collection can be performed in a more coarse-grained manner than the flow entries, the monitoring overhead can be reduced.

Second, OpenSample [21], OpenTM [22], FlowCover [23], OpenNetMon [38], and cFlow [20] perform sampling on monitoring, which means that parts of the statistics are selectively gathered to reduce overhead. OpenSample performs sampling on packets on the network and calculates the flow and port statistics based on the sampled packets. OpenTM monitors only statistics of some flows for calculating link utilization. FlowCover selects network switches using greedy algorithms and heuristics in response to flow changes. OpenNetMon monitors only edge switches for per-flow statistics. In addition, cFlow calculates link utilization using machine learning, and the required flows that have high impact on the prediction accuracy of utilization are prioritized and monitored.

Third, Tahaei et al. [39], PayLess [24], and IPro [40] regulated the monitoring interval. Tahaei et al. introduced a monitoring scheme that frequently measures the statistics that contribute more highly to link utilization than others. They also designed their scheme as sampling because their scheme only monitors the statistics of top-of-rack or edge switches. PayLess regulates the monitoring interval of each flow according to its contribution on link utilization, similar to the scheme of Tahaei et al. IPro regulates the monitoring interval using reinforcement learning.

In addition, MicroTE [41] implemented monitoring and traffic engineering functions on a separate machine from the

SDN controllers; thus, the bottleneck is removed from the controller itself. FlowSense [42] introduced a technique for removing additional statistics request messages from controllers by embedding the statistics values in other OpenFlow messages, such as PacketIn (for flow entry generation) or FlowRemoved (notification for the event that the flow entry has been deleted).

The above-mentioned studies change the schemes of statistics collection or the manner of transmission (e.g., the number or intervals of statistics request messages). On the other hand, the goal of V-Sight is to provide isolated statistics and reduce the transmission delay and control channel consumption of the network hypervisor. Therefore, V-Sight and the studies above are orthogonal and can work together.

One thing to note is that the evaluation methodology in Table 3 can be categorized as follows. First, previous studies [36], [37] proposed a new architecture for monitoring and evaluated the architecture through a hardware prototype. Using the hardware prototype, evaluations are conducted on relatively small testbeds, usually a single switch, or simulations using traces. Second, other studies [20], [23], [42] implemented their solutions as a type of simulator and tested them based on network traces. In addition, several studies [21], [22], [24], [38], [39], [40] implemented a component (module) atop an existing SDN controller (e.g., Floodlight, NOX, POX, or Ryu) and evaluated the component using actual switches (e.g., Open vSwitch).

For evaluations with switches, previous studies used linear and fat-tree topologies that do not contain routing loops. We believe that this topology selection is performed because the existing SDN controllers are not capable of properly handling routing loops in a network topology [43]. Thus, we expect to conduct experiments when SDN controllers become capable of handling routing loops. In the meantime, we fully implement V-Sight in a network hypervisor and evaluate V-Sight with linear and fat-tree topologies (§4).

3 V-Sight Design

In this section, we first introduce the overall architecture of the V-Sight framework and its operations. We then present three mechanisms of V-Sight: 1) statistics virtualization for isolated statistics, 2) transmission disaggregation for improved transmission delay, and 3) pCollector aggregation for reduced control channel consumption.

3.1 V-Sight Framework Architecture

Fig. 6 shows the architecture of the V-Sight framework. The processing sequence of V-Sight is as follows. When a statistics request (e.g., vf or vp) from a tenant controller is sent, the statistics virtualization component (§3.2) of V-Sight receives the message and calculates the requested vStatistics based on the pStatistics. For calculation, V-Sight references the virtualization map that maintains mappings between the VN and PN resources.

The pStatistics required for vStatistics calculation is obtained from the "pStatistics cache." Transmission disaggregation (§3.3) maintains the pStatistics cache, and the cache is filled by the pCollector. Transmission disaggregation enables a pCollector to run before the vStatistics request. A

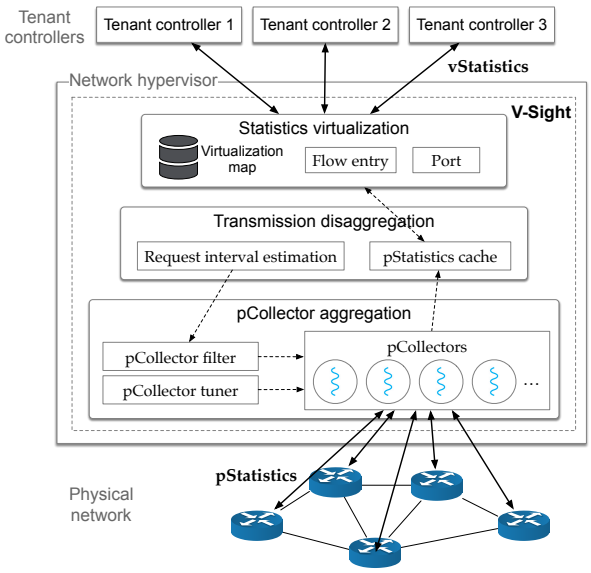


Fig. 6. V-Sight architecture.

key point of transmission disaggregation is to prepare the pStatistics needed for the vStatistics by disaggregating the time the vStatistics comes in and the time at which the pStatistics is ready. In other words, transmission disaggregation ensures that the pStatistics is in the pStatistics cache before the vStatistics request arrives. To achieve this, transmission disaggregation performs the "request interval estimation."

pCollector aggregation (§3.4) consists of two tasks: the "pCollector filter" decides the execution period of each pCollector and checks whether pCollectors can be merged as one pCollector for a specific physical switch; and the "pCollector tuner" decides the starting delay of a pCollector for improved accuracy.

3.2 Statistics Virtualization

Statistics virtualization aims to provide per-VN vStatistics from non-isolated pStatistics. We develop calculation algorithms for vfs (flow entry) and vps (port), which are the most fine-grained resources of network monitoring in SDN networks [6]. Other resources (e.g., flow table, switch, or entire network) can be derived from the per-VN statistics.

3.2.1 Per-VN flow entry statistics

For statistics isolation, V-Sight checks the mapping between vf and pf from the virtualization map (in the statistics virtualization component of Fig. 6), which maintains the relationships between the vf from tenant controllers and pf existing in the PN. The mapping of vf is used in two ways. First, if pf is not shared with the other VNs ($|V(pf)| = 1$), the statistics of pf become the statistics of vf. Second, pf is shared between VNs[2] ($|V(pf)| > 1$) [17], [18], [44]. In this case, because the pf aggregates all the statistics of

2. Multiple VNs can share flow entries when NH merges flow entries in order to reduce the physical memory consumed by the flow entries [17], [18], [44]. The conditions for flow entry merging are: 1) the flow entries are for packet forwarding, 2) the input port and output port of the flow entries are identical or their masked IP addresses are identical, and 3) VN permits the sharing of flow entries.

Algorithm 1: Per-tenant flow entry statistics.

Input: vf: virtual flow entry for which the VN controller requires statistics
Output: $S(vf)$: statistics of the vf
$pf = V'(vf)$
if $|V(pf)| == 1$ **then**
  $S(vf) = S(pf)$
else
  if $|V(pf)| > 1$ **then**
    E_{pf} = Find edge pf of vf
    $S(vf) = S(E_{pf})$
Return $S(vf)$

Algorithm 2: Per-tenant port statistics.

Input: vp: virtual port for which the VN controller requires statistics
vs: virtual switch to which the vp belongs
vf, vf^{in}, vf^{out}: virtual flow entry, input port of the vf, output port of the vf
Output: $S(vp)$: statistics of the vp
$pp = V'(vp)$
if $|V(pp)| == 1$ **then**
  $S(vp) = S(pp)$
else
  for vf_i *belongs to* vs **do**
    if $vf_i^{in} == vp$ **then**
      $S(vp).RX+ = S(vf_i)$;
    else if $vf_i^{out} == vp$ **then**
      $S(vp).TX+ = S(vf_i)$
Return $S(vp)$

vfs mapped to the pf, V-Sight should not return the pf statistics directly to the tenant controller. Instead, V-Sight isolates the pf statistics with the following observation: even though multiple vfs are mapped to one pf, the vfs for edge switches (the first and last switches on the forwarding path) are installed individually per VN. This is because the packets are dealt with separately per VN in the edge switches to ensure isolation in NV [18], [44]. In other words, pf in the edge is allocated per-VN so that the packets at the edge are delivered to the host (or VM). Thus, V-Sight returns the pStatistics of the edge switch pf as the requested vStatistics. Alg. 1 summarizes how to obtain the per-VN flow entry statistics. Because the vf statistics contain the packet number (count) and byte (quantity), the algorithm calculates the count and quantity individually.

3.2.2 Per-VN port statistics

vp statistics include the count and amount of received (RX) and transmitted (TX) packets. Similar to the flow entry, a pp can be shared by one or more VNs. If only one VN utilizes the physical port, the statistics of pp become vp statistics. Meanwhile, if pp is mapped to multiple vps, it receives and transmits the traffic of multiple VNs. In this case, V-Sight uses the vf statistics obtained in Alg. 1 because the vfs process the packets going to and from the vp of a switch. For RX packets, V-Sight accumulates the vStatistics of the vfs that have vp as their input port. To calculate the TX packet statistics, V-Sight sums the vf statistics that send packets out to the vp. This calculation is summarized in Alg. 2.

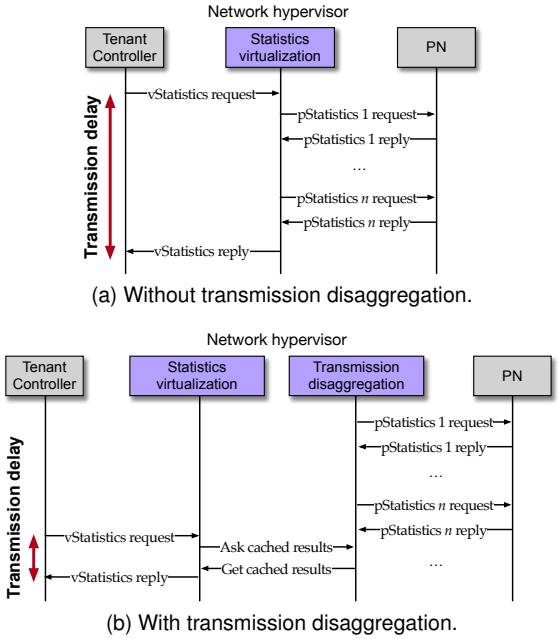


Fig. 7. Transmission delay comparison.

3.3 Transmission Disaggregation

As formulated in §2.3.2, the increased delay from SDN-NV is denoted by $nd_p + d_{NH}$. The time d_{NH} is to perform statistics virtualization (§3.2); thus, minimizing the transmission delay aims to reduce nd_p, which is the time for pStatistics transmissions (Fig. 7b). To reduce nd_p, transmission disaggregation introduces the pStatistics cache and the request interval estimation, to reduce the transmission delay from Fig. 7a to Fig. 7b.

3.3.1 pStatistics cache

The pStatistics cache tracks the time that pStatistics are stored and whether it has already been used per VN. When the pStatistics cache contains pStatistics that are not out-of-date (old), the pStatistics can be directly returned without retrieving pStatistics from any physical switch of the network hypervisor (hit).

The pStatistics cache is considered old when 1) the retrieved time of the pStatistics is longer than the monitoring interval or 2) it has already been used for the requested VN. The reasons are as follows. First, when the tenant controller performs periodic network monitoring, at least the statistics measured within the request interval should be returned because, if the statistics are collected before the interval starts, the value is old with respect to the current request; so, it is not accurate for the request. Second, if the stored pStatistics are used for the requested VN, the tenant controller would have already collected the data at that time; thus, we consider the data old. If pStatistics do not exist in the pStatistics cache (miss) or they are old, the cache retrieves the pStatistics from physical switches. Fig. 7b shows the working of transmission disaggregation.

When the number of pStatistics required for vStatistics is n, and k of pStatistics are "hit" (i.e., $n-k$ accesses to the pStatistics cache are "miss" or "old"), the physical transmissions of $n-k$ times are conducted for vStatistics. Subsequently, the entire transmission delay can be reduced to $(1+n-k)d_p + d_{NH}$. Therefore, increasing the number k is important for improving the transmission delay.

In addition, when pStatistics are updated, the pStatistics cache verifies whether the previously stored value has been used. If the pStatistics stored in the pStatistics cache are not used for a certain time (e.g., 10 times), they are removed from the pStatistics cache. This policy prevents useless transmission disaggregation. Even if the pStatistics are released, they can be re-cached to the pStatistics cache when the vStatistics that require pStatistics for statistics virtualization are requested.

3.3.2 Request interval estimation

The pStatistics cache is filled by pCollectors. A pCollector exists per pf so that a pCollector executes to retrieve the pStatistics of the pf of a physical switch. In particular, we use the term "interval" for the time between two consecutive requests from a tenant controller for a pf and "period" for the time difference between two consecutive executions of a pCollector. For each pCollector, the period of execution should be determined. If the period of the pCollector is much shorter than the request interval, the pCollector will end up executing multiple times before a "hit," which wastes CPU and control channel resources. Conversely, if the pCollector is executed less often than the vStatistics requests, the transmission delay cannot be reduced because the pStatistics are "old." Therefore, determining the execution period is very important, and this is the reason the request interval estimation is used.

The request interval estimation calculates the mean (μ) and variance (σ) per pf that characterize the VN controller's request intervals. For pf_i, the request of VN j is denoted as $pf_{i,j}$, and its distribution is $(\mu_{i,j}, \sigma_{i,j})$. The pStatistics cache contains a pf identifier (pf_i) and VN identifier (j). The k-th interval for $pf_{i,j}$ is denoted as $pf^k_{i,j}$.

Fig. 8 shows the flowchart of the entire request interval estimation. This process is executed every time the pf identifier (pf_i) and VN identifier (j) are received as per each vStatistics request. First, the request interval estimation records the interval between consecutive requests (① in Fig. 8). The request interval estimation calculates $(\mu_{i,j}, \sigma_{i,j})$ (③) once a certain number of intervals is accumulated, which is denoted as "interval window (w)[3]." When the $(w+1)$-th request arrives, the distribution of $pf_{i,j}$ $(\mu_{i,j}, \sigma_{i,j})$ is calculated based on the $pf^1_{i,j}$ to $pf^w_{i,j}$. Next, among the distributions of multiple VNs, V-Sight chooses the interval distribution that has the minimum μ value (⑤). In other words, $(\mu_i, \sigma_i) = (\mu_{i,l}, \sigma_{i,l})$ where $l = \arg\min_j \mu_{i,j}$. Requests that have a higher μ than the selected pf_i will "hit" because the pCollector for pf_i based on (μ_i, σ_i) stores the statistics of pf_i for those requests in a timely manner. The selected distribution is passed to the pCollector aggregation (⑥, §3.4) as a triple (pf_i, μ_i, σ_i). Note that a pCollector is created per pf after the interval window (w number of intervals) is accumulated. Before the interval window, the pStatistics cache generates a "miss" for the required pStatistics of pf_i, which makes V-Sight collect pStatistics from the PN for each request.

Clearly, the request interval of each tenant controller can change. The request estimation interval flushes the w number of past intervals ($pf^1_{i,j}$ to $pf^w_{i,j}$) after sending a new interval distribution (⑦) and accumulates the intervals from 1 to w again. Therefore, for the w number of recorded intervals (②), $(\mu_{i,j}, \sigma_{i,j})$ is updated (③). If the pCollector for pf_i has already been created (④), the request interval estimation checks how much the newly updated $\mu_{i,j}$ has changed from the previous value (⑧). If the changed amount is large (e.g., 25%), this function selects a new distribution for pf_i (⑤) and delivers a new triple (pf_i, μ_i, σ_i) to pCollector aggregation (⑥).

3.4 pCollector Aggregation

The objective of pCollector aggregation is to execute and merge pCollectors. Given a triple (pf_i, μ_i, σ_i) from transmission disaggregation, a pCollector for pf_i is created. The pCollector periodically retrieves the pf_i statistics from a switch. However, if the number of pCollectors increases, the pCollectors can consume too much of the control channel (as discussed in §2.3.3).

There are two types of pCollectors, as shown in Fig. 9. At the top of Fig. 9, three pCollectors retrieve statistics from their own pf. The bottom of Fig. 9 shows one pCollector that collects multiple pf statistics of a switch simultaneously. The latter pCollector consumes less of the control channel than the former because the required message sizes for statistics transmission are smaller. Specifically, for the statistics requests, the former pCollector should contain the specific information of individual pf, i; thus, the request message size is formulated as $l(H) + l(I(i))$. If n numbers of the former pCollectors are running, the entire request message size becomes $n \times l(H) + \sum_k l(I(i_k))$. In the reply message, the statistics for the requested pfs are added, so $n \times l(H) + \sum_k l(I(i_k)) + l(S(i_k))$.

In contrast, the request message of the latter pCollector includes "all flow entry" instead of individual information of pfs. The number of request messages then becomes one, and its size is $l(H)+l(I(*_{pf}))$. The payload of the reply message does not change compared with the former pCollector to contain the statistics of each pf, but the replies are created as a single message corresponding to a single request message, resulting in the creation of a single packet header. Thus, the size of the reply is $l(H) + \sum_k l(I(i_k) + l(S(i_k))$ [4]. Thus, the latter pCollector reduces less control traffic in amount of $(2n-2) \times l(H) + \sum_k l(I(i_k)) - l(I(*_{pf}))$.

We call the pCollector for a single pf (former) as a "tiny pCollector" and the other pCollector as an "aggregated pCollector." An aggregated pCollector is created when multiple tiny pCollectors follow a similar period for pfs in a switch. pCollector aggregation is achieved using two tasks:

3. Explicitly, we find that the value 30 is sufficient to obtain a stable and reliable interval distribution with general SDN controllers.

4. When the size of the reply exceeds the maximum transmission unit size, the reply packet fragments. In this case, the number of $l(H)$ consumed can increase but still be less than the former pCollector, because the payloads are piggybacked as much as possible.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3089225

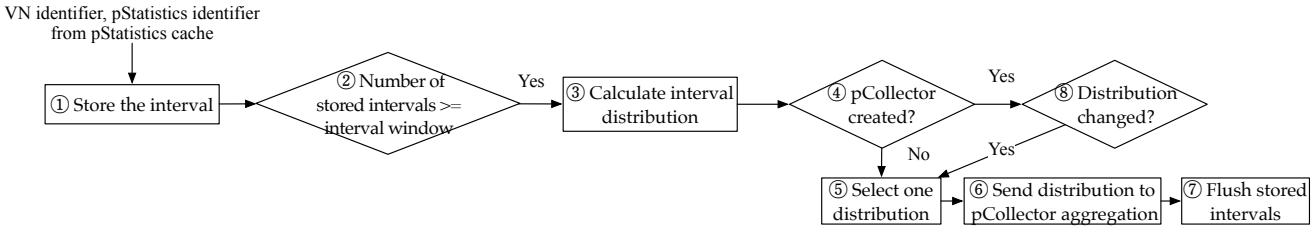


Fig. 8. Flowchart of request interval estimation.

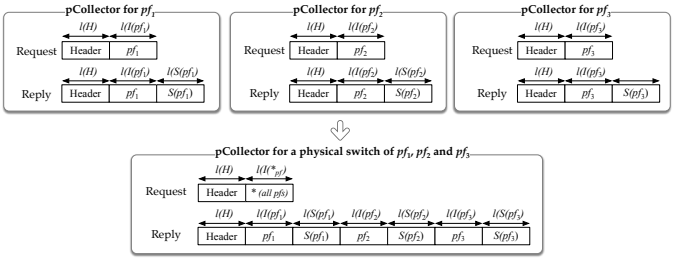


Fig. 9. Control channel consumption for two kinds of pCollectors.

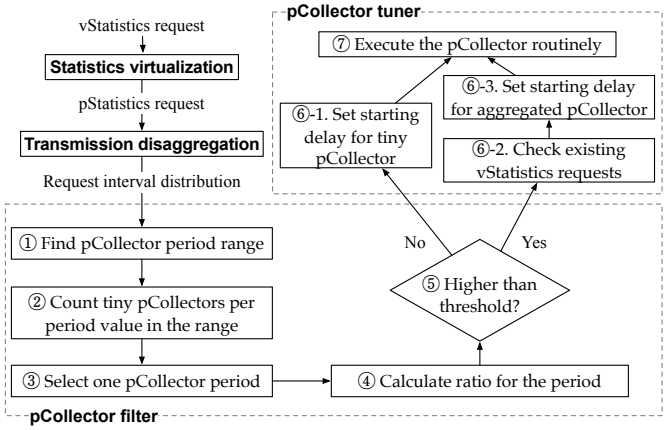


Fig. 10. Flowchart of pCollector aggregation. This routine is executed according to statistics virtualization and transmission disaggregation.

1) a pCollector filter determines the execution period of the tiny pCollectors and aggregated pCollectors and 2) a pCollector tuner improves the accuracy of vStatistics. Fig. 10 explains the operation of the two tasks to be discussed in the following subsections.

3.4.1 pCollector filter

From (pf_i, μ_i, σ_i), the pCollector filter decides on a period of the pCollector for pf_i. For the tiny pCollector, it is simple. However, for the aggregated pCollector, even if tenant controllers issue statistics requests with similar intervals, each μ_i of pf_i can be slightly different (e.g., 4.7, 4.9, and 5.1 s) because the distribution is estimated based on w samples. Thus, it is challenging to decide the period of an aggregated pCollector.

To address this problem, the pCollector filter starts with tiny pCollectors that have a similar period. From the cumulative probability distribution function derived by μ_i and σ_i, the pCollector filter finds a period range that satisfies a specific hit rate, such as 90% to 95% (① in Fig. 10). The requests that have longer intervals than the pCollector's period will "hit"; so, this task can stochastically derive the period range using μ_i and σ_i. Next, for every possible period value within the range, the pCollector filter counts the number of tiny pCollectors with the period for the value (②). The period value with the largest number of tiny pCollectors is selected (③). Once a period is selected, the pCollector filter calculates the ratio of the number of tiny pCollectors that follow a similar period to the number of existing pfs in the switch (④). If the ratio is low, an aggregated pCollector consumes more control traffic than the tiny pCollectors. Subsequently, only when the ratio is high (⑤), for instance, 70%[5], the pCollector tuner merges tiny pCollectors into an aggregated pCollector.

3.4.2 pCollector tuner

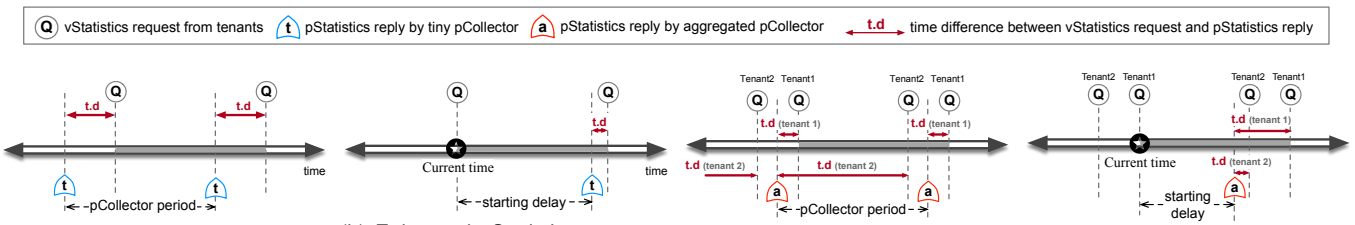
The role of the pCollector tuner is to provide an additional delay to the first execution of each pCollector in order to improve the accuracy of vStatistics. In Fig. 11a, a time difference, which is shown as an arrow with "t.d.," exists between the time the vStatistics requests arrive and the time the pStatistics are gathered through the pCollector. This time difference depends on the time at which the pCollector first runs. If the pCollector is executed slightly before the vStatistics request, the time difference becomes small, as shown in Fig. 11b, which implies that the cached pStatistics are up to date. As the time difference becomes larger, it decreases the accuracy of vStatistics. Therefore, V-Sight introduces a "starting delay" to add the delay to the first execution of the pCollectors.

For tiny pCollectors, the starting delay should be set in order to execute the tiny pCollector immediately before the vStatistics requests (coming after the interval window). In addition, the starting delay should not be too large to prevent the pCollector from being executed after the vStatistics request, as shown in Fig. 11a. Empirically, we set the starting delay at 95% of the pCollector period (⑥-1 in Fig. 10).

Meanwhile, the method of setting the starting delay for tiny pCollectors leads to poor accuracy for aggregated pCollectors. This is because the multiple requests managed by an aggregated pCollector exist at different times in terms of the pCollector period. Fig. 11c shows an example with two vStatistics requests from different tenants (tenant2 followed by tenant1). If the starting delay is set to 95% of the aggregated pCollector, the execution time of the aggregated pCollector is after tenant2 and before tenant1. As shown in Fig. 11c, tenant2 suffers a long delay because the aggregated pCollector executes immediately after tenant2's request.

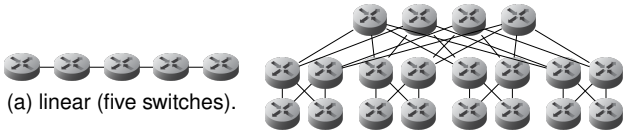
Therefore, the pCollector tuner sets the starting delay for the aggregated pCollector as follows. First, the pCollector

5. In our evaluation, we explicitly find that sufficient improvement is obtained using 70%.



(a) Low vStatistics accuracy (tiny pCollector).
(b) Enhanced vStatistics accuracy with starting delay for tiny pCollectors.
(c) Low vStatistics accuracy (aggregated pCollector).
(d) Enhanced vStatistics accuracy with starting delay for aggregated pCollectors.

Fig. 11. Starting delay for tiny pCollectors and aggregated pCollectors.



(a) linear (five switches).
(b) fat-tree (4-ary).

Fig. 12. Experiment topologies.

TABLE 4
Hardware and software specifications.

Server (hardware) specifications.	
CPU	Intel Xeon E5-2650 (2.30 GHz)
Memory	64 GB
NIC	Intel 82599ES 10GbE NIC
Software specifications.	
PN	Mininet: 2.3.0d6 with Open vSwitch v2.9.5 OS: Ubuntu 18.04
Network hypervisor	Libera: v0.1 OS: Ubuntu 14.04
Tenant controller	ONOS: v2.0.0 OS: Ubuntu 16.04.6

tuner checks the request interval estimation (§3.3.2), which stores the vStatistics request times for each VN (⑥-2 in Fig. 10). Then, the starting delay is set to be immediately before the first vStatistics request among the VN requests that the aggregated pCollector merges, which is tenant2's request in Fig. 11d (⑥-3). In this way, the sum of time differences from the time the aggregated pCollector executes to the time each vStatistics request arrives is minimized. Finally, the pCollector tuner executes the pCollector periodically with the starting delay (⑦).

4 Evaluation

In this section, we present the evaluation results of V-Sight. V-Sight is implemented on Libera network hypervisor with OpenFlow version 1.3 (1.8K LoCs) [5], [13]. The source code of V-Sight is available in the GitHub repository[6]. We measure micro-benchmarks, system overheads, and macro-benchmarks that are explained in detail below. Each experiment is repeated to obtain reliable results.

4.1 Test Setup

4.1.1 Settings

We use three physical servers. Mininet [45], the network hypervisor, and one or more ONOS as tenant controllers run on separate physical servers. Table 4 summarizes the hardware and software specifications used for evaluations. Mininet emulates the PN based on Open vSwitch. We emulate two types of topologies (Fig. 12): 1) a linear topology consisting of five switches and 2) a 4-ary fat-tree topology to evaluate the effects on datacenters. For the linear topology, we create three tenants that clone the PN topology as their VN topologies. For each tenant, the number of TCP connections varies (i.e., 2, 4, 6, 8, 10); thus, in the PN, 6, 12, 18, 24, or 30 connections exist. For the fat-tree topology, we change the number of connections to 2, 4, 8, 16, and 32 with a single tenant, resulting in the same number of TCP connections as in the PN. The TCP connections are generated through the iperf3 [46]. Each VN is managed by an ONOS controller. The ONOS monitors all the flow entries and ports of each switch at 5 s intervals. ONOS controllers run as containers, and no ONOS container suffers performance or resource bottlenecks.

4.1.2 Metrics

We evaluate V-Sight based on the following micro-benchmarking metrics.

- **Statistics virtualization accuracy (§4.2)**: the root mean squared error (RMSE)—caused by the statistics virtualization algorithms—between the statistics calculated from the network hypervisor and the actual value.
- **Transmission delay (§4.3)**: the average interval between the vStatistics request and the reply messages from/to tenant controllers.
- **Control channel consumption (§4.4)**: the average bytes per second of the control channel traffic to obtain pf statistics between the network hypervisor and the physical switches.

Second, we present three metrics for the system overheads of V-Sight as below.

- **Time skew of pStatistics cache (§4.5)**: time skew implies an interval between the vStatistics request time and pStatistics collection time of the pCollectors—the average value with 95% confidence interval. This time skew shows the time difference of transmission disaggregation on the accuracy of the pStatistics required for the vStatistics calculation.
- **CPU and memory usage (§4.6)**: the average CPU cycle and memory consumption of the V-Sight framework during the experiment.

Finally, for macro-benchmarks (§4.7) in practice, we show the effects of V-Sight on tenants by measuring the TCP

6. https://github.com/gsyang33/V-Sight. V-Sight is easily tested through the tutorial from Libera (https://github.com/os-libera/Libera).

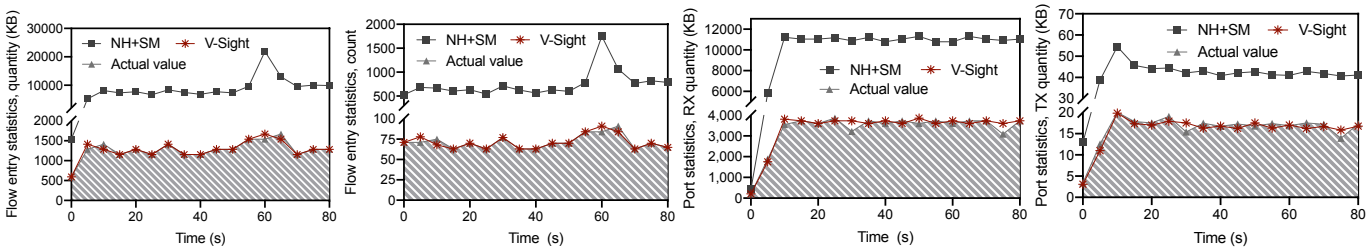
TABLE 5
RMSEs of statistics provided by SDN-NV and V-Sight compared with real values. Quantity is the volume of data processed by each flow entry or port (shown in the unit of MB), and count is the number of packets processed.

	NH+SM		V-Sight	
	Quantity	Count	Quantity	Count
Tenant 1	114.78	10000.83	0.14	10.87
Tenant 2	121.80	10633.79	0.02	8.52
Tenant 3	112.62	9875.42	0.55	13.60

(a) Flow entry statistics.

	NH+SM				V-Sight			
	RX		TX		RX		TX	
	Quantity	Count	Quantity	Count	Quantity	Count	Quantity	Count
Tenant 1	65.83	5871.13	65.77	5871.00	0.15	16.85	0.15	16.85
Tenant 2	53.33	4583.02	53.24	4582.68	0.42	37.98	0.41	38.02
Tenant 3	40.92	4469.03	40.38	4468.36	0.88	61.81	0.89	61.94

(b) Port statistics.



(a) flow entry statistics, quantity. (b) flow entry statistics, count. (c) port statistics, RX quantity. (d) port statistics, TX quantity.

Fig. 13. vStatistics replied by the network hypervisor in comparison with actual values.

throughput, CPU cycle, and control channel consumption of the tenant controller.

4.1.3 Comparisons

The metrics explained in §4.1.2 are measured in the following comparison cases:

- **Native**: non-virtualized SDN in which physical switches are directly connected to ONOS without a network hypervisor.
- **Network hypervisor (NH)**: Libera[7] without V-Sight. It frequently returns no values to the statistics requests from tenant controllers.
- **NH with simple monitoring function (NH+SM)**: Libera with simple monitoring that is used in §2.3.2.
- **V-Sight**: the full implementation of V-Sight.

4.2 Statistics Virtualization Accuracy

Statistics virtualization accuracy is measured on the linear topology with three tenants because accuracy results in the linear and fat-tree topologies are similar. Each tenant generates single TCP traffic at different sending rates. We measure the vStatistics collected from NH+SM and V-Sight and present the RMSEs of the statistics from the two cases and the actual values. The vStatistics from V-Sight could contain errors when the calculation algorithms use the pfs in edge switches and the statistics of the pf or pp contain the statistics of multiple tenants (§3.2). In other words, when the calculation uses statistics from other switches, it can result in errors. In addition, the pStatistics used for the calculation are retrieved in advance, not at the requested time, which can also lead to errors (§3.3). The actual values used for error calculation are measured from hosts that send and receive the traffic within each tenant's VN.

7. To the best of our knowledge, existing network hypervisors (including open-source network hypervisors) do not have a complete network monitoring framework, so we choose Libera, which is up to date and open-source, for comparison with V-Sight.

Statistics virtualization provides isolated statistics from the pStatistics. When pf or pp is not shared between multiple tenants and only mapped to a single tenant, their statistics are those of the single tenant's vf or vp. Therefore, for this evaluation, we set pf and pp to be mapped to multiple tenants based on the concepts of previous studies [11], [12], [17], [18].

Table 5 shows the RMSEs of vStatistics provided by NH+SM and V-Sight per tenant. For both flow entry and port statistics, two types of vStatistics, namely, quantity and count (number of packets), are provided. In terms of the flow entry in Table 5a, the RMSEs for quantity in V-Sight are less than 1, whereas those of NH+SM are more than 110. For the count, the average RMSE of V-Sight is 10.99, whereas that of NH+SM is 10170.01—an improvement by three orders of magnitude.

In terms of port (Table 5b), vStatistics are categorized into RX and TX for incoming and outgoing traffic, respectively. NH+SM exhibits an average RMSE of 53.25 for the quantity, whereas V-Sight exhibits an average of 0.48. Similarly, for the count, NH+SM and V-Sight exhibit the average values of 4974.20 and 38.91, respectively. In summary, V-Sight exhibits much better accuracy (by two orders of magnitude) than NH+SM because NH+SM does not have any mechanism to provide isolated statistics. This result implies that the statistics virtualization algorithms of V-Sight successfully provide isolated statistics from pStatistics.

In addition, Fig. 13 shows the vStatistics over time, which the tenant controller receives in response to its requests from NH+SM and V-Sight. The monitoring results of tenant 2's vf and vp installed in the center switch of the linear topology are shown as representative results. Each point represents the number of quantities or counts processed since the previous statistics reply. The hatched area in the graphs depicts the actual values.

Figs. 13a and 13b show the vf statistics of quantity and count, respectively, and Figs. 13c and 13d show the

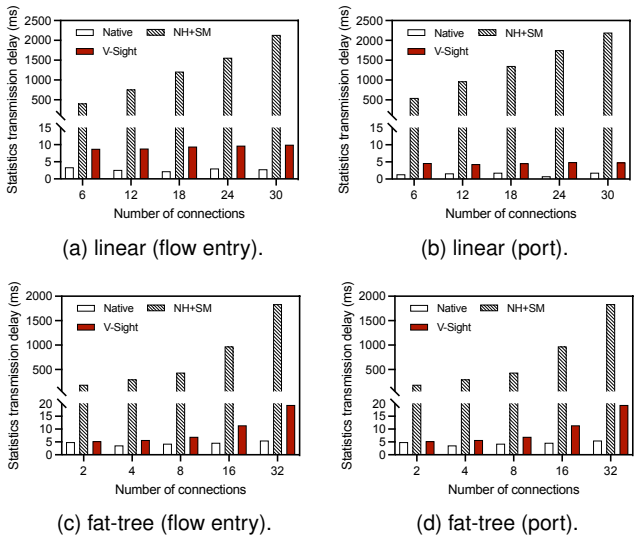


Fig. 14. Average statistics transmission delay (ms).

vp statistics for RX quantity and TX quantity, respectively. We omit the graphs for RX and TX counts for vp statistics because they look similar to the quantity results of the vp statistics. In short, the vStatistics provided in NH+SM are aggregated values of tenants that are significantly different from the actual values. For example, the mean absolute errors (MAEs) of the flow entry statistics for the count (Fig. 13b) of NH+SM and V-Sight are 612% and 2%, respectively, and other results show similar tendencies on MAEs. By using V-Sight, the tenant controllers can receive statistics very close to the actual values, thus enhancing the accuracy of the network management and optimization tasks of the tenant controllers.

4.3 Transmission Delay

Transmission delay is measured on both the linear and fat-tree topologies. The results are analyzed in regard to two criteria: 1) V-Sight performance improvement—a comparison between NH+SM and V-Sight, and 2) SDN-NV overheads—a comparison between Native and V-Sight. To measure the transmission delay, we modify the ONOS controller to report the sending time of the statistics request message and the receiving time of the statistics reply. Except for this timestamping, the ONOS is not modified in any other way.

Because the PN, network hypervisors, and tenant controllers execute on separate physical servers, we measure the RTT between two servers. The average and 95% tail RTT are 0.15 ms and 0.18 ms, respectively, for all pairs of servers. Further, we confirm that no bottleneck occurs on network connections between servers.

4.3.1 V-Sight performance improvement

In the linear topology, V-Sight consumes 9.35 and 4.68 ms, on average, for flow entry and port statistics transmission, respectively (Figs. 14a and 14b). The delays in V-Sight improve by 46 times (flow entry statistics, 6 connections) to 454 times (port statistics, 30 connections) compared with those in NH+SM. For the fat-tree topology, V-Sight exhibits 9.75 and 7.29 ms of transmission delay for flow entry and port, respectively (Figs. 14c and 14d). The delay in V-Sight improves by 14 times (port statistics, 2 connections) to 269 times (port statistics, 32 connections).

In detail, the transmission delay in NH+SM increases in proportion to the number of TCP connections (even by over 2 s) because the number of pStatistics required for vStatistics increases as the number of connections increases. Subsequently, the vStatistics are returned to the tenant controller only after the corresponding pStatistics are collected. Conversely, V-Sight disaggregates the pStatistics transmission routines from the statistics virtualization, thereby reducing this delay.

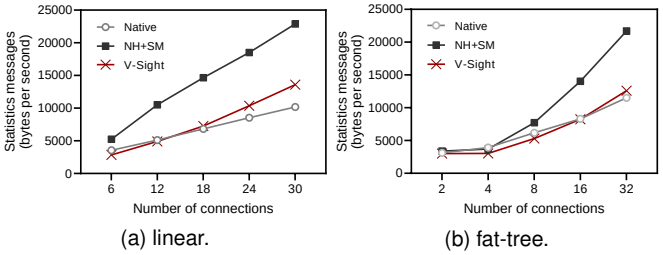


Fig. 15. Control channel traffic consumption (bytes per second).

4.3.2 SDN-NV overheads

We compare the transmission delay between V-Sight and Native to examine the SDN-NV overheads. In the linear topology, Native exhibits 2.8 and 1.5 ms for flow entry and port statistics transmission delay, respectively (Figs. 14a and 14b). The delays in V-Sight are 3.4 times higher, on average, than those of Native. Further, for the fat-tree topology, Native exhibits 4.6 and 2.37 ms delays for flow entry and port statistics transmission (Figs. 14c and 14d), respectively. The results of V-Sight are 1.09 times (flow statistics, 2 connections) to 6.69 times (port statistics, 2 connections) higher than those of Native.

Although the delays of V-Sight are higher than those of Native, note that all the values are lower than 20 ms. In comparison, the default monitoring intervals of the ONOS, Floodlight, and OpenDayLight are 5, 10, and 15 s, respectively. Therefore, we believe that the transmission delay of V-Sight, which is 19.36 ms at maximum, is acceptable.

4.4 Control Channel Consumption

Similar to the transmission delay, the control channel consumption for statistics is also evaluated under the linear and fat-tree topologies. Two criteria are used: V-Sight performance improvement and SDN-NV overheads as in §4.3.

4.4.1 V-Sight performance improvement

Fig. 15 shows the control channel consumption for statistics transmission in both topologies. The consumption increases in proportion to the number of connections because the number of pfs to be monitored increases accordingly. In the linear topology (Fig. 15a), V-Sight improves the control channel consumption by approximately 1.9 times on average. In the fat-tree topology (Fig. 15b), the average consumption of V-Sight is 1.44 times less than that of NH+SM. This improvement is due to the benefit of the aggregated pCollector that merges the individual statistics messages (§3.4).

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3089225

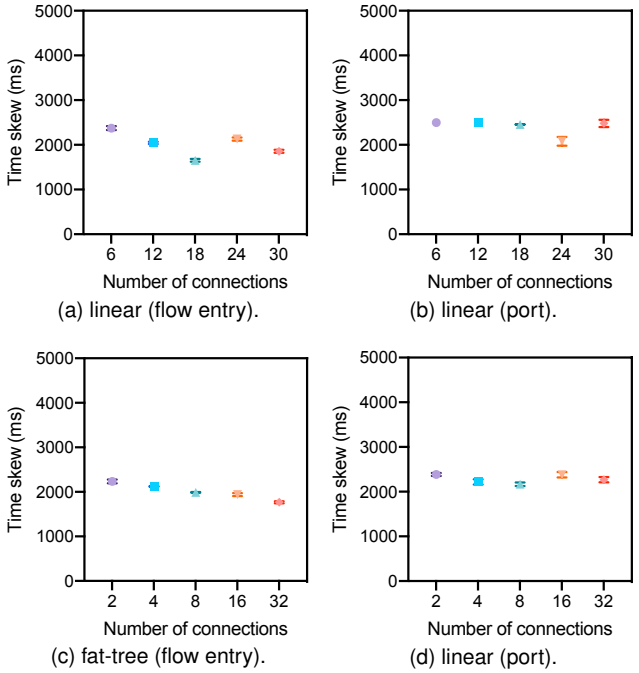


Fig. 16. Time skews of pStatistics cache (ms).

4.4.2 SDN-NV overheads

Comparing V-Sight with Native, V-Sight consumes 107% and 93% of Native's control channel traffic in the linear and fat-tree topologies, respectively, which implies that the consumption of V-Sight is comparable to that of Native. Furthermore, in the fat-tree topology with few network connections, V-Sight is even better than Native. This is because V-Sight only monitors the switches that have the pStatistics required for vStatistics. The fat-tree topology has 20 switches (Fig. 12b), and multiple paths are available between every host pair. In this topology, when the number of connections is small, not all switches are used for packet forwarding and, consequently, not for vStatistics.

In Native, however, the tenant controller monitors all the switches in the PN. Therefore, request and reply messages are generated for all switches regularly. In V-Sight, transmission disaggregation controls the creation of pCollectors toward the required pfs. Thus, pCollectors are created only for the required pfs, and the statistics request/reply messages are not created for switches that are not used.

4.5 Time Skew of pStatistics Cache

The experiment results of previous sections (§4.3 and §4.4) show that V-Sight successfully improves the transmission delay and control channel consumption. The improvements come with a time skew, which is the interval between the times when a vStatistics request arrives and when the pStatistics required for the vStatistics request are stored in the cache. When this timing in pStatistics cache becomes longer, the pStatistics required for the vStatistics are collected from the PN much in advance; thus, the calculated vStatistics could be out-of-date.

Fig. 16 shows the time skews for the number of network connections in the linear and fat-tree topologies, plotted with average and 95% confidence intervals. The results

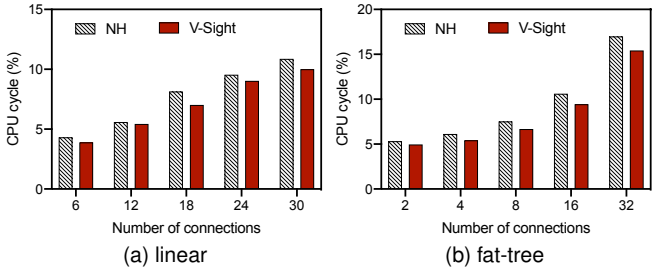


Fig. 17. CPU cycle usage (%).

indicate that the average time skews in all cases of network connections are equal to or less than 2500 ms in both topologies. This implies that V-Sight replies to the tenant controller within 2.5 s, which is half of the request interval of the tenant controllers. Consequently, the tenant controller, at least, does not receive statistics from the previous statistics request; therefore, the accuracy overhead does not jeopardize the accuracy of vStatistics itself.

4.6 CPU and Memory Usage

V-Sight inevitably incurs additional computational resource consumption, especially CPU and memory resources in the network hypervisor (e.g., pStatistics cache and pCollectors). The CPU and memory usage are measured with similar settings for the transmission delay and control channel consumption evaluations. We compare the results of NH and V-Sight.

4.6.1 CPU usage

Fig. 17 shows the average CPU cycle usage of NH and V-Sight during the evaluation based on the network topology and number of connections. Regardless of the topologies (Figs. 17a and 17b), the CPU cycles are proportional to the number of connections for both NH and V-Sight. Comparing the two, V-Sight is expected to use more CPU cycles than NH because V-Sight runs additional threads, such as pCollectors. However, surprisingly, V-Sight consumes, on average, 0.6% and 0.9% fewer CPU cycles than NH in the linear and fat-tree topologies, respectively.

The reason for the CPU usage results is because V-Sight prevents the unnecessary operations of tenant controllers. Specifically, we find that the tenant controller in the experiments periodically collects flow entry and port statistics to confirm whether the network policies, such as the installed flow entries or the configured settings on the network devices, are consistent between the controller and the network.

However, NH, without a network monitoring scheme, does not reply to such statistics requests (i.e., mostly answering that there is nothing in the switches). Thus, the tenant controller considers this situation as inconsistent. The controller then removes or re-installs the flow entries, which leads to the repeated installation of flow entries already existing in the PN. Thus, NH repeatedly processes the re-installation messages from the tenant controllers. On the other hand, V-Sight eliminates these repetitions because it provides accurate and timely statistics to tenant controllers. For this reason, even if V-Sight adds additional design and

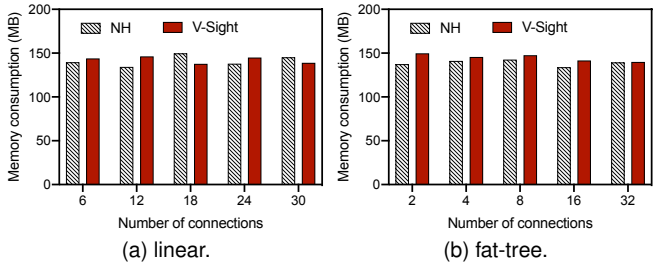
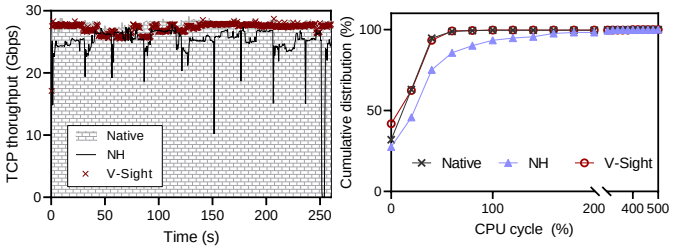


Fig. 18. Memory consumption (MB).



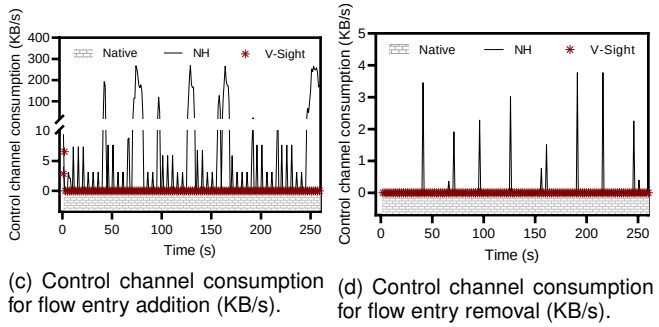


Fig. 19. Effects of V-Sight on tenants.

implementation overheads to the NH, the CPU cycle is improved.

4.6.2 Memory usage

Fig. 18 depicts the memory consumption of V-Sight and NH. In the results, two cases exhibit memory consumption of between 130 and 150 MB for both topologies. On average, V-Sight consumes 0.99 and 6.06 MB more memory than NH in the linear and fat-tree topologies, respectively. This consumption comes from the additional structures in V-Sight such as the pStatistics cache.

However, at some points (e.g., 18 and 30 connections in the linear topology), V-Sight consumes lesser memory than NH. This result also comes from the same reasons why the CPU consumption is improved (§4.6.1). NH and V-Sight store the messages from the tenant controllers to distinguish the flow entries and policies between the tenants for VN isolation. Therefore, when the messages for setting up flow entries and policies are generated repeatedly, NH sometimes consumes more memory than V-Sight.

4.7 Effects of V-Sight on Tenants

We investigate the effects of V-Sight on tenants in practice. Because network monitoring is used for routing (flow entry installation), we measure the TCP throughput. In addition, the CPU cycles of tenant controllers and control channel consumption (between tenant controllers and network hypervisor) are measured. The Native, NH, and V-Sight cases are compared in the linear topology of one tenant having a single TCP connection.

TABLE 6
Total amount of control channel consumptions of VN controller (KB).

	Native	NH	V-Sight
Flow entry addition	4.10	5510.54	9.43
Flow entry removal	0	23.16	0

Fig. 19a shows the TCP throughput in the data plane. We omit the first 5 s, which is typically the period for congestion window convergence. First, the results indicate that the throughput of NH fluctuates more than that of Native. The 90% tail throughput values of Native, NH, and V-Sight are 25.9, 18.7, and 25.6 Gbps, respectively. Even after approximately 250 s, the throughput of NH decreases to zero, while Native and V-Sight show relatively constant throughput values throughout the experiment. In addition, the ranges between the lowest and highest TCP throughput of V-Sight and Native are 3.5 and 3.2 Gbps, respectively, while that of NH is 17.6 Gbps, which implies that 1) the performance of V-Sight is similar to Native and 2) the improvement in variance over that of NH is about 5.5 times.

The reason for this TCP throughput improvement in V-Sight is the same as the reason discussed in Section4.6.1. Because NH does not provide correct statistics to the tenant controller, the flow entries used for transmitting packets are re-installed. When they are removed or re-installed in the middle of packet processing, the TCP throughput shows high variation and a value of even zero in NH. On the other hand, V-Sight provides timely and correct statistics similar to Native, and thus, its TCP throughput also becomes similar to that of Native.

Next, Fig. 19b shows the cumulative distribution function of the CPU cycles of the tenant controller. The results show that the Native and V-Sight cases exhibit similar distributions. The average CPU cycle consumptions of Native, NH, and V-Sight are 23.94%, 47.25%, and 22.09%, respectively. V-Sight improves the CPU utilization of the tenant controller about 2.14 times over that of NH by removing the flow entry inconsistency situations.

Lastly, Table 6 summarizes the total amount of control channel consumptions, which are categorized into flow entry addition and flow entry removal. Figs. 19c and 19d show the control channel consumptions according to time. For flow entry addition, NH consumes significantly higher volumes of control channel traffic, i.e., 1344.03 and 584.36 times over Native and V-Sight, respectively (Table 2), because of the repetitive flow entry installation, which can be confirmed from NH's repetitive spike patterns in Fig. 19c. On the other hand, Native and V-Sight consume control channel traffic only at the beginning. In terms of flow entry removal (Table 6 and Fig. 19d), Native and V-Sight do not consume any traffic, while NH consumes 23.16 KB.

In comparison to Native, V-Sight consumes 2.3 times more bandwidth for flow entry installation because V-Sight is a network hypervisor and it inherently has longer control

channel messages [13], [17]; but, note that the network hypervisor provides address virtualization and topology virtualization for isolation between tenants while Native does not.

5 Discussion

5.1 Consideration of In-band Network Telemetry

Unlike existing monitoring approaches, in-band network telemetry (INT) provides custom packet-level network monitoring abilities by allowing the collection and reporting of network states according to user-defined operations in network switches [47], [48]. For example, a network user who wants to know certain states inside a network (e.g., the queue length of each switch) sends an instruction to INT-capable devices to notify them of the types of states to be collected. The devices then embed the requested states in packets, and the hosts can receive the in-network states [49].

However, INT-based monitoring requires INT-capable hardware devices such as a field programmable gate array (FPGA) or P4 programmable switch, while V-Sight exists as a software component that operates on SDN-compatible switches that are common nowadays. In addition, INT cannot deliver per-tenant isolated statistics itself. In terms of NH, existing NHs do not support P4 or FPGA to enable INT. However, several approaches for making P4 and other devices on NV exist [50], [51], [52], [53]. So, if INT could be available on NV, it also could be a part of V-Sight framework as a means to collect custom network statistics. Then, V-Sight designs for isolated statistics and resource-efficient monitoring schemes could work for INT as well.

5.2 Isolated Statistics using P4

Using P4, network operators can implement custom operations on hardware switches. This could open the possibility for isolating statistics from the switch-side, not from the network hypervisor. To enable this, switches should be able to distinguish which tenant each packet belongs to, for counting packets per tenants. A challenge is that the method of classifying tenants at the packet level differs for each proposed network hypervisor (e.g., VLAN [12], address rewriting [13], MPLS [18], and TID embedding [44]). We leave this topic as our future work, which can provide an accurate and flexible monitoring scheme in SDN-NV.

5.3 Machine Learning for Monitoring on SDN-NV

Recently, various studies attempted to use machine learning to predict network traffic in the data plane. For example, various studies [54], [55] used neural networks to predict the existing network traffic or load on paths for better network management, such as routing. Currently, V-Sight calculates vStatistics based on non-isolated pStatistics. If prediction on the tenant statistics is possible, the accuracy of vStatistics could be enhanced. Thus, it is an open question of future research to investigate.

5.4 Integration with Cloud Orchestration Platforms

One of the interesting future research is integrating V-Sight with cloud orchestration platforms for practically deploying V-Sight. Several cloud orchestration platforms (e.g., OpenStack and Kubernetes) provide plug-in type solutions for SDN systems [56] to manage the underlying network infrastructures and control the network connections between the containers of cloud infrastructures. Since V-Sight is atop of SDN-based solutions, it can be readily integrated with such solutions.

5.5 Consolidated Monitoring Framework for In-network Computing

Another promising research direction is about building a consolidated and unified monitoring framework for in-network computing. For upcoming beyond 5G and 6G systems, the network resources are expected to perform both network functions (e.g., packet forwarding) and computing functions offloaded from hosts [57]. For example, several studies proposed offloaded training on deep learning models that accelerates the speeds [58], [59]. In that system, the monitoring framework should consider both the network side and the computational resources of the network devices (such as CPUs, memories, and ASICs). Such consolidated monitoring can lead to highly optimized decision on network devices (e.g., routing considering both network and computational resource capacities).

6 Conclusion

We present V-Sight, the first comprehensive network monitoring framework in SDN-NV. V-Sight makes it possible to isolate statistics between VNs, reduce statistics transmission delays, and scale control channel consumption. To this end, V-Sight introduces statistics virtualization, transmission disaggregation, and pCollector aggregation. We implement V-Sight and evaluate its key performance characteristics in terms of statistics virtualization accuracy, transmission delay, and control channel consumption. Furthermore, we present the time skew of pStatistics cache, CPU and memory usage, and effects of V-Sight on tenants, which implies that V-Sight attains a level comparable to network monitoring in a non-virtualized SDN.

In future research, we plan to investigate the reliability and performance of VNs through traffic engineering, working on the isolated and timely statistics from V-Sight.

Acknowledgments

We thank Gi Jun Moon listed in [1] who made substantial contributions to earlier work. This research was supported in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (MSIT) (NRF-2019H1D8A2105513) and also by Basic Science Research Program through the NRF funded by the Ministry of Education (NRF-2021R1A6A1A13044830). This work was also partially supported by Institute of Information & communications Technology Planning & Evaluation grant funded by the Korea government (MSIT) (2015-0-00280, (SW Starlab) Next generation cloud infra-software toward the guarantee of performance and security SLA), and a Korea University Grant.

REFERENCES

[1] G. Yang, H. Jin, M. Kang, G. J. Moon, and C. Yoo, "Network monitoring for SDN virtual networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1261–1270.

[2] B. Wang, Z. Qi, R. Ma, H. Guan, and A. V. Vasilakos, "A survey on data center networking for cloud computing," *Computer Networks*, vol. 91, pp. 528–547, 2015.

[3] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson *et al.*, "Network virtualization in multi-tenant datacenters," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, 2014, pp. 203–216.

[4] D. Firestone, "VFP: A virtual switch platform for host SDN in the public cloud," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, 2017, pp. 315–328.

[5] 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.

[6] Z. B. Pfaff, B. Lantz, B. Heller, C. Barker, D. Cohn, D. Talayco, D. Erickson, E. Crabbe, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Pettit, K. Yap, L. Poutievski, M. Casado, M. Takahashi, M. Kobayashi, and N. McKeown, "Openflow switch specification," *Open Networking Foundation*, vol. 28, pp. 1–56, 2011.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[8] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[9] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[10] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NaaS: Network-as-a-service in the cloud," in *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*. USENIX Association, 2012.

[11] 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 *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, vol. 10. USENIX Association, 2010, pp. 365–378.

[12] 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.

[13] 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*. Association for Computing Machinery, 2014, pp. 25–30.

[14] "POX controller." Accessed: Nov. 17, 2020. [Online]. Available: https://noxrepo.github.io/pox-doc/html/

[15] 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*. Association for Computing Machinery, 2014, pp. 1–6.

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

[17] 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.

[18] 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.

[19] B.-y. Yu, G. Yang, H. Jin, and C. Yoo, "WhiteVisor: Support of white-box switch in SDN-based network hypervisor," in *2019 International Conference on Information Networking (ICOIN)*. IEEE, 2019, pp. 242–247.

[20] M. Li, C. Chen, C. Hua, and X. Guan, "CFlow: A learning-based compressive flow statistics collection scheme for SDNs," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[21] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," in *2014 IEEE 34th International Conference on Distributed Computing Systems*. IEEE, 2014, pp. 228–237.

[22] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for openflow networks," in *Passive and Active Measurement*, A. Krishnamurthy and B. Plattner, Eds. Springer Berlin Heidelberg, 2010, pp. 201–210.

[23] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *2014 IEEE Global Communications Conference*, 2014, pp. 1956–1961.

[24] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[25] A. Roy, D. Bansal, D. Brumley, H. K. Chandrappa, P. Sharma, R. Tewari, B. Arzani, and A. C. Snoeren, "Cloud datacenter SDN monitoring: Experiences and challenges," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. Association for Computing Machinery, 2018, pp. 464–470.

[26] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, 2011.

[27] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3958–3969, 2018.

[28] Z. Bozakov and P. Papadimitriou, "AutoSlice: Automated and scalable slicing for software-defined networks," in *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, ser. CoNEXT Student '12. Association for Computing Machinery, 2012, pp. 3–4.

[29] H. Yamanaka, E. Kawai, S. Ishii, and S. Shimojo, "AutoVFlow: Autonomous virtualization for wide-area openflow networks," in *2014 Third European Workshop on Software Defined Networks*, 2014, pp. 67–72.

[30] A. Blenk, A. Basta, and W. Kellerer, "HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 397–405.

[31] 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 (NSDI 15)*. USENIX Association, 2015, pp. 87–101.

[32] 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*. IEEE, 2021.

[33] 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.

[34] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[35] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, and Y. Wang, "A survey of network virtualization techniques for internet of things using SDN and NFV," *ACM Comput. Surv.*, vol. 53, no. 2, Apr. 2020.

[36] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, 2013, pp. 29–42.

[37] X. T. Phan and K. Fukuda, "SDN-Mon: Fine-grained traffic monitoring framework in software-defined networks," *Journal of Information Processing*, vol. 25, pp. 182–190, 2017.

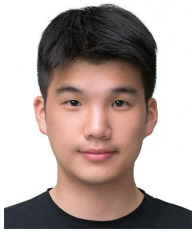
[38] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–8.

[39] H. Tahaei, R. Salleh, S. Khan, R. Izard, K.-K. R. Choo, and N. B. Anuar, "A multi-objective software defined network traffic measurement," *Measurement*, vol. 95, pp. 317–327, 2017.

[40] E. F. Castillo, O. M. C. Rendon, A. Ordonez, and L. Zambenedetti Granville, "IPro: An approach for intelligent SDN monitoring," *Computer Networks*, vol. 170, p. 107108, 2020.
[41] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*. Association for Computing Machinery, 2011.
[42] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *International Conference on Passive and Active Network Measurement*. Springer, 2013, pp. 31–41.
[43] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient loop-free rerouting of multiple SDN flows," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 948–961, 2018.
[44] B.-y. Yu, G. Yang, K. Lee, and C. Yoo, "AggFlow: Scalable and efficient network address virtualization on software defined networking," in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*. Association for Computing Machinery, 2016, pp. 1–6.
[45] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Association for Computing Machinery, 2010, pp. 1–6.
[46] V. Gueant, "iperf-the tcp, udp and sctp network bandwidth measurement tool," *Iperf. fr.*, 2017, Accessed: Aug. 06, 2020. [Online]. Available: https://iperf.fr/
[47] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM Demo*, 2015.
[48] "Cisco nexus 9000 series nx-os programmability guide, release 9.2(x) - inband network telemetry [cisco nexus 9000 series switches]," 2020, Accessed: Nov. 02, 2020. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/92x/programmability/guide/b-cisco-nexus-9000-series-nx-os-programmability-guide-92x/b-cisco-nexus-9000-series-nx-os-programmability-guide-92x_chapter_0100001.html
[49] T. P. W. Group, "In-band network telemetry (int) dataplane specification," 2020, Accessed: Oct. 24, 2020. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf
[50] S. Han, S. Jang, H. Choi, H. Lee, and S. Pack, "Virtualization in programmable data plane: A survey and open challenges," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 527–534, 2020.
[51] C. Zhang, J. Bi, Y. Zhou, and J. Wu, "HyperVDP: High-performance virtualization of the programmable data plane," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 556–569, 2019.
[52] M. Saquetti, G. Bueno, W. Cordeiro, and J. R. Azambuja, "P4VBox: Enabling P4-based switch virtualization," *IEEE Communications Letters*, vol. 24, no. 1, pp. 146–149, 2019.
[53] P. Zheng, T. Benson, and C. Hu, "P4visor: Lightweight virtualization and composition primitives for building and testing modular programs," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. Association for Computing Machinery, 2018, pp. 98–111.
[54] R. Alvizu, S. Troia, G. Maier, and A. Pattavina, "Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks," *Journal of Optical Communications and Networking*, vol. 9, no. 9, pp. D19–D30, Sep 2017.
[55] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 393–430, 2019.
[56] Z. Tao, Q. Xia, Z. Hao, C. Li, L. Ma, S. Yi, and Q. Li, "A survey of virtual machine management in edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1482–1499, 2019.
[57] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*. Association for Computing Machinery, 2019.
[58] M. Kang, G. Yang, Y. Yoo, and C. Yoo, "TensorExpress: In-network communication scheduling for distributed deep learning," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 25–27.
[59] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with in-network aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 2021, pp. 785–808.

Gyeongsik Yang [M] received his B.S., M.S., and Ph.D. degrees in computer science from Korea University, Seoul, Republic of Korea, in 2015, 2017, and 2019, respectively.

He worked as a research intern at Microsoft Research Asia in 2018. He is currently a research professor at Korea University. His research interests include network virtualization, distributed deep learning, data center systems, and SDN.



Yeonho Yoo received his B.S. degree in computer science from Kookmin University, Seoul, Republic of Korea, in 2017.

He is currently pursuing his M.S. degree with Korea University, Seoul, Republic of Korea. His current research interests include network virtualization, distributed deep learning, and SDN.

Minkoo Kang received his B.S. and M.S. degrees in computer science from Korea University in 2019 and 2021, respectively.

He is currently a researcher at KIST. His research interests include programmable dataplane, data center networking, and distributed deep learning.

Heesang Jin received his B.S. and M.S. degrees in computer science from Kookmin University, Seoul, Republic of Korea, in 2018 and Korea University, Seoul, Republic of Korea, in 2020, respectively.

He is currently a researcher at ETRI. His research interests include network virtualization, traffic engineering, blockchain, and SDN.

Chuck Yoo [M] received his 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.