

# Network Monitoring for SDN Virtual Networks

Gyeongsik Yang, Heesang Jin, Minkoo Kang, Gi Jun Moon, Chuck Yoo

*Department of Computer Science and Engineering*

*Korea University*

Seoul, Republic of Korea

ksyang@os.korea.ac.kr, hsjin@os.korea.ac.kr, mkkang@os.korea.ac.kr, shangmoon@korea.ac.kr, chuckyoo@os.korea.ac.kr

**Abstract**—This paper proposes V-Sight, a network monitoring framework for software-defined networking (SDN)-based virtual networks. Network virtualization with SDN (SDN-NV) makes it possible to realize programmable virtual networks; so, the technology can be beneficial to cloud services for tenants. However, to the best of our knowledge, although network monitoring is a vital prerequisite for managing and optimizing virtual networks, it has not been investigated in the context of SDN-NV. Thus, virtual networks suffer from non-isolated statistics between virtual networks, high monitoring delays, and excessive control channel consumption for gathering statistics, which critically hinders the benefits of SDN-NV. To solve these problems, V-Sight presents 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. V-Sight is implemented on top of OpenVirtX, and the evaluation results demonstrate that V-Sight successfully reduces monitoring delay and control channel consumption up to 454 times.

**Index Terms**—Network monitoring, network virtualization, Software-defined networking

## I. INTRODUCTION

Network virtualization (NV) based on software-defined networking (SDN) allows network operators to compose and manage virtual networks (VN) in their preferences. Network hypervisor (NH) [1]–[3] is an enabling technology that supports VN abstractions such as virtual switches, links, ports [1], and addresses [3]. NH sits between switches and VN controllers. With NH, VN users can create their own VN topology and control it using their VN controllers (e.g., POX [4], ONOS [5], OpenDayLight [6]).

Until now, NHs have evolved to be more scalable [7]–[9] and flexible [10]. However, to the best of our knowledge, none of the NHs provides network traffic monitoring, which is a vital prerequisite for network management. Thus, network traffic monitoring in SDN-NV causes three main issues: 1) inaccurate statistics, 2) high monitoring delay, and 3) excessive control channel traffic consumption. First, because VN controllers attempt to optimize and manage their VNs based on statistics, it is essential to have accurate statistics. However, in SDN-NV, the statistics collected in physical network are the aggregate of multiple VNs running on the network, and yet there is no mechanism to retrieve the statistics and isolate them for each VN.

Second, SDN-NV architecture inevitably increases the delay (so-called transmission delay) between the statistics request from the controller and the reply from switches. When the

statistics request message arrives from the controller to the NH, the NH must send the corresponding network statistics request messages to the physical network (switches) and wait to receive the results. Therefore, the transmission delay increases. As an example, if a VN controller sends a request for “all flow entries of a virtual switch,” the transmission delay can be exceedingly high since the statistics of individual flow entries are collected sequentially before being sent back to the VN controller. Our experiment shows that the transmission delay increases up to 333 times when compared to non-virtualized SDN (§II-C2). The increased delay means that the collected statistics can be out-of-date.

Third, the NH excessively consumes control channel traffic compared with non-virtualized SDN. In our experiment, control channel consumption increases up to three times (§II-C3) when the VN controller asks for the statistics of all the flow entries per switch. This high consumption is because the NH has to send multiple messages to switches in order to respond to a request from the controller. Since such messages go through the control channel, the control channel consumption increases, and other traffic gets affected [11]. For instance, our experiment finds that flow rule installation time increases 4.3 times because of the control channel consumption for retrieving the statistics.

To sum up, the problems caused by the lack of network monitoring preclude network optimization and essential network controls. To solve these problems, we design V-Sight, a comprehensive network monitoring framework for SDN-NV. V-Sight provides three mechanisms: 1) statistics virtualization to isolate statistics per VN, 2) transmission disaggregation to reduce the transmission delay, and 3) pCollector aggregation to reduce the control channel consumption.

Statistics virtualization calculates virtual network statistics (vStatistics) per VN from physical network statistics (pStatistics) (§III-B). Transmission disaggregation (§III-C) uses caching of the frequently used pStatistics. The caching is done with pCollector that retrieves the pStatistics routinely and stores the data in NH, which removes the delays for pStatistics transmission. We design pCollector aggregation (§III-D) to reduce the control channel consumption of pCollector. Rather than collecting pStatistics from individual pCollectors, pCollector aggregation attempts to merge pCollectors so that multiple pStatistics are retrieved with a single request message, which reduces the number of messages and thus control channel consumption.

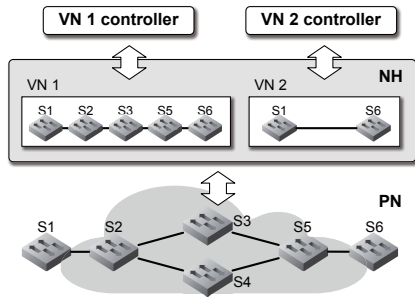


Fig. 1: SDN-based network virtualization.

V-Sight accomplishes the following contributions:

- We address the problems of network monitoring for SDN-NV: statistics isolation, improved monitoring delay, and enhanced control channel consumption for NHs.
- We design and develop V-Sight, the first network monitoring framework that leads to accurate statistics and low-overhead monitoring for virtualized SDN.
- We implement V-Sight in an open-source network hypervisor, OpenVirteX (OVX), and evaluate the framework rigorously.

The remainder of this paper is organized as follows. §II describes the background, motivation, and particular problems of network monitoring in SDN-NV. The fundamental concepts and the complete design of V-Sight are given in §III, and §IV presents the evaluation results. §V elaborates on the related work. Finally, §VI concludes this paper.

## II. BACKGROUND AND MOTIVATION

Here, we explain the background of this study: SDN-NV and network monitoring. Then, we identify issues for network monitoring on SDN-NV.

### A. SDN-based Network Virtualization

SDN-NV is composed of three layers, as shown in Fig. 1: the VN controllers, NH, and physical network (PN). A VN controller can create its VN topology with VN resources, such as virtual switches, links, and ports. This is done when the VN controller sends a request to the NH. When the NH receives the request, it substantiates the VN resources with the mappings to the PN switches or ports. 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*.

Once the VN topology is created, the VN controller<sup>1</sup> runs to manage the created virtual resources. The VN controller connects with the virtual switches through south-bound interfaces (e.g., OpenFlow) and implements flow entries that match packets so that it processes (e.g., forward) the matched packets. These operations are done by control messages from the VN

<sup>1</sup>Typically, SDN controllers (e.g., POX, ONOS, OpenDayLight) are used as VN controllers. So, throughout this paper, we use the term ‘SDN controller,’ and ‘VN controller’ interchangeably. For non-virtualized SDN context, we use the term ‘SDN controller,’ and ‘VN controller’ is used for SDN-NV.

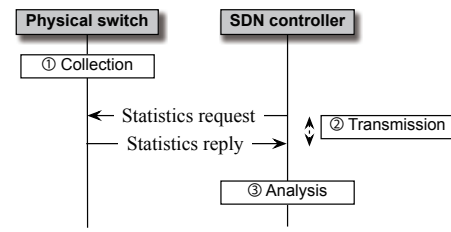


Fig. 2: Three steps of network monitoring in SDN.

controller, and the control messages go through the control channel.

VN resources and flow entries are mapped to the corresponding resources in the PN. This means that the PN resources can be mapped to either one or more of the VN resources. Thus, the flow entries from multiple VN controllers can be mapped to the smaller number of physical flow entries [7], [8]. Throughout this paper, we use the term  $V$  to represent a virtualization function and  $V'$  to represent a de-virtualization function that gets the mapped resources to the physical resource, or vice versa. For example, when a physical flow entry ( $pf$ ) is given,  $V(pf)$  gives the virtual flow entries ( $vfs$ ) mapped to the  $pf$ . Also, given a virtual switch  $S$ ,  $V(S)$  generates the list of physical switches and links mapped to  $S$ .

### B. Network Monitoring in SDN

Network monitoring in SDN goes through three steps as shown in Fig. 2 [12]: ① collection, ② transmission, and ③ analysis. The statistics are recorded at the switches, which measure the processed amount of packets per flow entry or port (① collection). Then, an SDN controller gathers the statistics from a switch (② transmission). With the collected information, the SDN controller analyses, manages, and optimizes the networks (③ analysis).

For example, ONOS [5] collects the statistics of flow entries and ports at 5 s intervals. Also, ONOS checks the consistency of the implemented flow entries, which needs to collect a large number of flow entry statistics. Therefore, it is well-known that such network monitoring causes enormous overheads to SDN controllers [13]. The overheads are considered as a critical problem for SDN controllers; therefore, many studies have tried to reduce the monitoring overheads. Typically, their solutions use a trade-off relationship between statistics accuracy and monitoring overhead, so they lower the accuracy in order to reduce the monitoring overhead. For example, transmission sampling [14]–[16], statistics hashing [17], and statistics prediction [18] compromise the accuracy by omitting the collection of some statistics values. Therefore, directly applying these solutions to SDN-NV makes the monitoring overhead worse. It is because multiple VNs increase the monitoring overhead which can easily void the validity of monitoring itself.

### C. Issues of Network Monitoring for SDN-based Network Virtualization

Here, we discuss three issues caused by the lack of network monitoring in SDN-NV in detail, which motivates V-Sight.

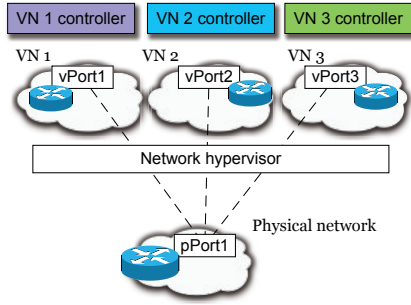


Fig. 3: Non-isolated statistics example.

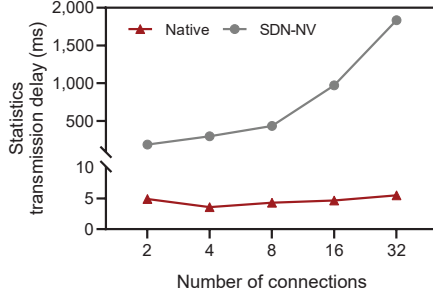


Fig. 4: Statistics transmission delay comparison (ms).

1) *Non-isolated statistics*: In SDN-NV, the PN resources (e.g., switch and port) are shared between multiple VNs. So, the collected statistics from physical resources are not isolated between the VNs. Figure 3 shows an example of three VNs, each having one *vp*. In this scenario, all *vps* are mapped to the same *pp* (pPort1). Suppose that the VN1 controller retrieves the statistics of vPort1. As pPort1 does not know about the presence of multiple VNs, it does not separately collect the statistics per VN. Thus, the VN1 controller ends up with the aggregated statistics, not its own. Statistics are used for various network management operations of VN controllers, such as cost-based central routing, traffic engineering features, and quality-of-service. However, with the non-isolated statistics, VN controllers cannot fulfill their desired goals. Thus, V-Sight should isolate statistics in the sense that the statistics provided to each VN controller should only contain information regarding the VN, not the others.

2) *High transmission delay*: Network monitoring is repeatedly performed to track changing statistics. The reply to the statistics request should arrive as soon as possible because the transmission delay between the request and reply messages distances the value of the statistics from the request time.

We experiment to see how much the transmission delay increases in SDN-NV. Because existing NHs do not support network monitoring, we implement a simple monitoring function on OVX. The implementation receives the statistics requests from VN controllers and then gathers the corresponding statistics from PN based on the mappings between VNs and PN. The monitoring function replies to VN controllers after the arrival of all pStatistics from the physical switches. The experiment is conducted in a 4-ary fat-tree topology with 2, 4, 8, 16, and 32 TCP connections in a VN. The VN controller

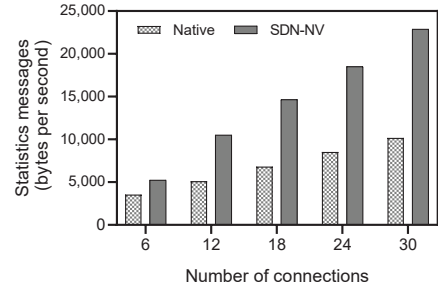


Fig. 5: Control channel consumption comparison (bytes per second).

issues statistics requests at a 5 s interval for every switch in its network, asking for the statistics of all flow entries of each switch.

As described in Fig. 4, non-virtualized SDN (Native) exhibits almost constant statistics transmission delay, at 4.6 ms on average, regardless of the number of network connections. In contrast, SDN-NV shows a delay of 187 ms to 1,836 ms, which is 38 to 333 times higher than that of Native. The reason for the increased delay is that, for a request for statistics of all flow entries in the switch, the implementation performs statistics transmission as many as the number of flow entries in the switch. After the statistics of each flow entry are collected, the VN controller receives the reply. As a result, the transmission delay in SDN-NV increases up to 1.84 s (Fig. 4).

3) *Excessive control channel consumption*: Statistics transmission is realized through the control channel. The control channel is also utilized by VN controllers for operations like switch connection handshaking, flow entry installation and modification, topology discovery process, and ARP processing. So, when the control channel consumption for statistics increases from 5.11 KB/s to 22 KB/s, we find that flow entry installation suffers four times higher delay (from 86 ms to 368 ms). Since operations such as flow entry installation can affect the throughput of network connections, the consumption of the control channel for network monitoring should be reduced.

To be precise, we evaluate the control channel consumption for the network monitoring of SDN-NV. We set a linear topology with five switches and three VNs. Each VN has 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 on the NH and statistics requests as §II-C2. Figure 5 shows the control channel consumption for the flow statistics transmission. The results are 1.5 to 2.3 times higher than Native. In Native, most SDN controllers collect all *pf* statistics with a single request message for a switch. On the contrary, the NH collects the statistics for each *pf* one-by-one. If 30 *pfs* exist in a switch and only three *pfs* are used for the VN, collecting all *pfs* for three *pfs* is very inefficient.

### III. V-SIGHT DESIGN

In this section, we first introduce the overall architecture of 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

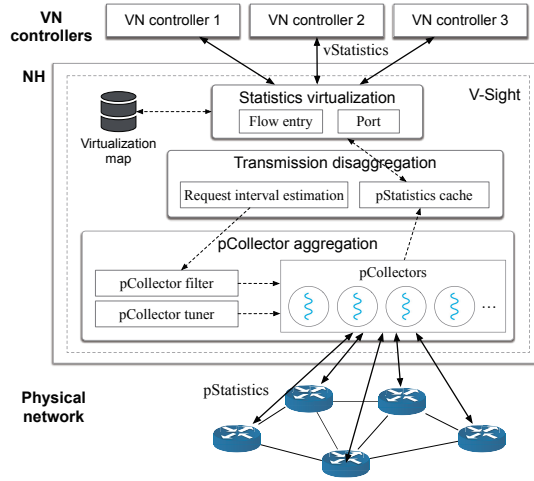


Fig. 6: V-Sight architecture.

transmission delay, and 3) pCollector aggregation for reduced control channel consumption.

#### A. V-Sight Framework Architecture

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

The  $pStatistics$  needed for  $vStatistics$  calculation is obtained from  $pStatistics$  cache. Transmission disaggregation (§III-C) maintains the ‘ $pStatistics$  cache,’ and the cache is filled by  $pCollector$ . What transmission disaggregation does is to run a  $pCollector$  before the  $vStatistics$  request. A key point of the transmission disaggregation is to prepare the  $pStatistics$  needed for the  $vStatistics$  requested with disaggregating the time the  $vStatistics$  comes in and the time the  $pStatistics$  are ready. In other words, transmission disaggregation allows the  $pStatistics$  in the  $pStatistics$  cache before the  $vStatistics$  request arrives. To achieve this, transmission disaggregation performs ‘request interval estimation.’

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

#### B. Statistics Virtualization

Statistics virtualization aims to provide per-VN  $vStatistics$  from non-isolated  $pStatistics$ . We develop calculation algorithms for  $vf$ s (flow entry) and  $vp$ s (port), which are the most fine-grained resources of network monitoring in SDN networks [19]. Other resources (e.g., flow table, switch, or entire network) could be derived from the per-VN statistics.

1) *Per-VN flow entry statistics*:  $vf$  statistics contain the packet count, byte count, and duration of the installed entry. For statistics isolation, V-Sight checks the mapping between

---

#### 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**

$E_{pf} = \text{Find edge } pf \text{ of } vf$

$S(vf) = S(E_{pf})$

**Return**  $S(vf)$

---

$vf$  and  $pf$  from the virtualization map (Fig. 6). 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 ( $|V(pf)| > 1$ ) [8], [11]. In this case, because the  $pf$  aggregates all the statistics of  $vfs$  mapped to the  $pf$ , V-Sight should not return the  $pf$  statistics directly to the VN 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 (first and last switches on the forwarding path) are installed individually per VN like the mapping case  $|V(pf)| = 1$ . This is because, in the edge switches, the packets are dealt with separately per VN to ensure isolation in NV [8], [20]. That is,  $pf$  in the edge is allocated per-VN so that the packets at the edge are delivered to the host (or virtual machine). Thus, V-Sight returns the  $pStatistics$  of the edge switch  $pf$  as the requested  $vStatistics$ . Algorithm 1 summarizes the steps for getting the per-VN flow entry statistics.

---

#### 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

$vf, v_i^{fin}, v_i^{fout}$ : 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**  $v_i^{fin} == vp$  **then**

$S(vp).RX += S(vf_i);$

**else if**  $v_i^{fout} == vp$  **then**

$S(vp).TX += S(vf_i)$

**Return**  $S(vp)$

---

2) *Per-VN port statistics*:  $vp$  statistics include the amounts of received (RX) and transmitted (TX) packets. Similar to the flow entry,  $pp$  can be shared by one or more VNs. If only a



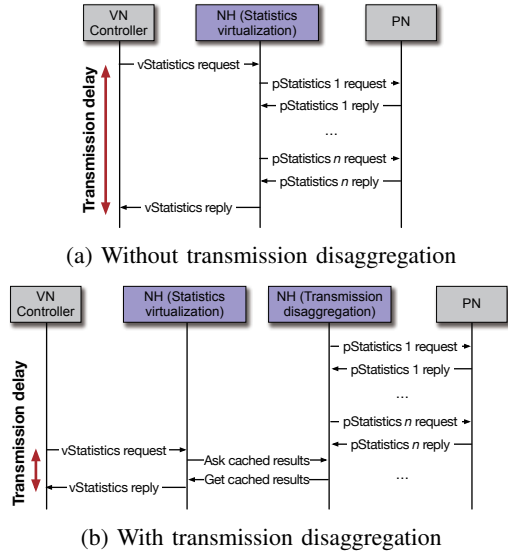


Fig. 7: Transmission delay comparison.

single VN utilizes the physical port, the statistics of  $pp$  become  $vp$  statistics. On the other hand, if  $pp$  is mapped to multiple  $vps$ , it receives and transmits the traffic of multiple VNs. In this case, V-Sight uses  $vf$  statistics obtained in Alg. 1 since  $vf$ s process the packets going to and from the  $vp$  of a switch. For RX, V-Sight accumulates the  $vStatistics$  of  $vfs$  that have  $vp$  as its input port. To calculate the TX, V-Sight adds up the  $vStatistics$  of the  $vfs$  that send packets out to the  $vp$ . This calculation is summarized in Alg. 2.

### C. Transmission Disaggregation

The transmission delay for  $vStatistics$  (Fig. 7a) is composed of the following parts: 1)  $vStatistics$  request and reply message transmission time ( $d_v$ ) between the VN controller and NH, 2)  $pStatistics$  request and reply message transmission time ( $d_p$ ) between the NH and physical switches, which may occur multiple times depending on the  $vStatistics$  request, and 3) processing time in the NH for statistics calculation ( $d_{NH}$ ). Then, if the number of  $pStatistics$  needed for  $vStatistics$  is  $n$ , the total transmission delay is formulated as  $d_v + nd_p + d_{NH}$ .

For comparison, the total transmission delay in non-virtualized SDN is  $d_c$  since single statistics transmission between PN and the SDN controller is enough to get statistics.  $d_v$  is equivalent to  $d_c$  because both are for transmitting messages through the control channel. So, the transmission delay of SDN-NV has the additional time of  $nd_p + d_{NH}$ . The time  $d_{NH}$  is to perform statistics virtualization (§III-B), and so the focus to reduce the transmission delay is on reducing  $nd_p$ , which is the time for  $pStatistics$  transmissions (Fig. 7b). To reduce  $nd_p$ , transmission disaggregation introduces the  $pStatistics$  cache and request interval estimation.

1) *pStatistics cache*: The  $pStatistics$  cache tracks the time when  $pStatistics$  is stored and whether  $pStatistics$  has already been used per VN. If the  $pStatistics$  cache contains  $pStatistics$  not used for the requesting VN before (hit), the  $pStatistics$  can be directly returned without retrieving  $pStatistics$  from any physical switches. On the other hand, if  $pStatistics$  does not

exist in the  $pStatistics$  cache (miss) or they are out-of-date because they have been previously used for the requesting VN (old), the cache retrieves  $pStatistics$  from physical switches. Figure 7b shows how transmission disaggregation works.

When the number of  $pStatistics$  needed for  $vStatistics$  is  $n$ , and  $k$  of  $pStatistics$  are hit (meaning  $n - k$  accesses to the  $pStatistics$  cache are miss or old), physical transmissions of  $n - k$  times are conducted for  $vStatistics$ . Then, 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.

2) *Request interval estimation*: The  $pStatistics$  cache is filled by  $pCollectors$ .  $pCollector$  exists per- $pf$  which means that a  $pCollector$  executes to retrieve  $pStatistics$  of a  $pf$  of a physical switch. To clarify, we use the term ‘interval’ for the time between two consecutive requests from a VN 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 the hit, which wastes the 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 what request interval estimation does.

Request interval estimation calculates the mean ( $\mu$ ) and variance ( $\sigma$ ) 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  $pf$  identifier ( $pf_i$ ) and VN identifier ( $j$ ). The  $k$ th interval for  $pf_{i,j}$  is denoted as  $pf_{i,j}^k$ .

Figure 8 shows the flowchart of the entire request interval estimation. This process is executed every time the  $pf$  identifier ( $pf_i$ ) and the VN identifier ( $j$ ) is received. This process is executed whenever the  $pf$  identifier ( $pf_i$ ) and the VN identifier ( $j$ ) is received as per each  $vStatistics$  request. First, request interval estimation records the interval between consecutive requests (① in Fig. 8). Once enough intervals are accumulated (②), request interval estimation calculates  $(\mu_{i,j}, \sigma_{i,j})$  (③). The number of intervals used for the calculation is denoted as ‘interval window ( $w$ )’<sup>2</sup>. When the  $(w + 1)$ th request comes, the distribution of  $pf_{i,j}$  ( $\mu_{i,j}, \sigma_{i,j}$ ) is calculated based on the  $pf_{i,j}^1$  to  $pf_{i,j}^w$ . Then, among the distributions of multiple VNs, V-Sight chooses the interval distribution that has the minimum  $\mu$  value (⑤). In other words,  $(\mu_i, \sigma_i) = (\mu_{i,l}, \sigma_{i,l})$  where  $l = \arg \min_j \mu_{i,j}$ . The requests that have higher  $\mu$  than the selected  $pf_i$  will hit because the  $pCollector$  for  $pf_i$  based on  $(\mu_i, \sigma_i)$  stores the statistics of  $pf_i$  for those requests in a timely way. The selected distribution is passed to  $pCollector$  aggregation (⑥, §III-D) as a triple  $(pf_i, \mu_i, \sigma_i)$ . Note that a  $pCollector$  is created after the  $w$  number of intervals are

<sup>2</sup>Owing to the space limit of this paper, we do not include the experiments for changing the  $w$  value. However, for the default monitoring operation of ONOS, 30 is enough to obtain a stable and reliable interval distribution.

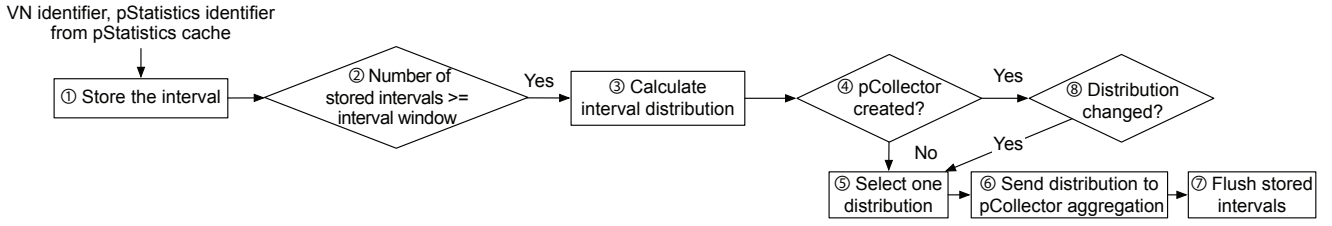


Fig. 8: Flowchart of request interval estimation.

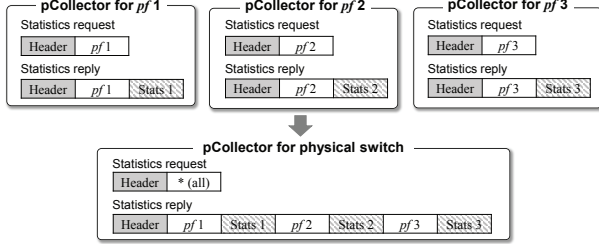


Fig. 9: Control channel consumption for two kinds of pCollectors.

accumulated. Before the interval window, pStatistics cache generates ‘miss’ for the required pStatistics of  $pf_i$  which makes V-Sight collect pStatistics from the PN for each request.

Obviously, the request interval of each VN controller can change. The request estimation interval flushes the  $w$  number of the past intervals ( $pf_{i,j}^1$  to  $pf_{i,j}^w$ ) 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  $pf_i$  is already created (④), request interval estimation checks how much the newly updated  $\mu_{i,j}$  is changed from the previous value (⑧). If the changed amount is large (say, 25%), this function selects a new distribution for  $pf_i$  (⑤) and delivers a new triple  $(pf_i, \mu_i, \sigma_i)$  to pCollector aggregation (⑥).

#### D. pCollector Aggregation

The objective of pCollector aggregation is to execute and merge pCollectors. Given a triple  $(pf_i, \mu_i, \sigma_i)$  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 §II-C3).

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 shows one pCollector that collects multiple statistics at once. For the former, the request message should contain the match fields and actions of  $pf$ . In contrast, for the latter, the request message contains “all” as its request. Obviously, the latter pCollector consumes less control channel than the former.

We call the pCollector for a single  $pf$  (former) as ‘tiny pCollector’ and the other pCollector as ‘aggregated pCollector.’ An aggregated pCollector is created when many tiny pCollectors follow a similar period for  $pfs$  in a switch.

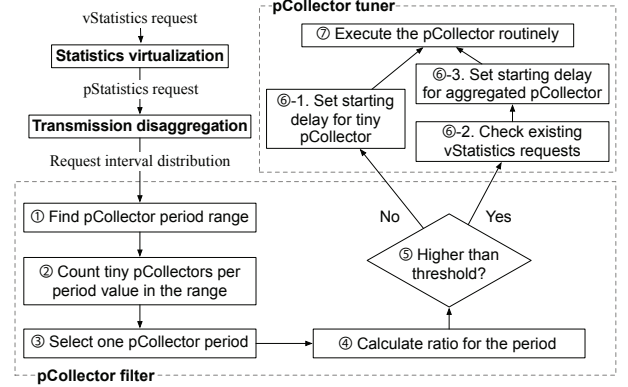


Fig. 10: Flowchart of pCollector aggregation. This routine is executed according to statistics virtualization and transmission disaggregation.

pCollector aggregation is done by two tasks: 1) pCollector filter to determine the execution period of tiny and aggregated pCollectors and 2) pCollector tuner for improving the accuracy of vStatistics using pCollector’s results. Figure 10 explains the operation of the two tasks to be discussed in the following subsections.

1) *pCollector filter*: From  $(pf_i, \mu_i, \sigma_i)$ , pCollector filter decides a period of the pCollector for  $pf_i$ . For the tiny pCollector, this is simple. However, for the aggregated pCollector, even if VN controllers issue statistics requests with a similar interval, each  $\mu_i$  of  $pf_i$  can be slightly different (e.g., 4.7 s, 4.9 s, and 5.1 s) since the distribution is estimated based on  $w$  samples. So, it is challenging to decide the period of aggregated pCollector.

To solve this problem, pCollector filter starts with tiny pCollectors to have a similar period. From the cumulative probability distribution function derived by  $\mu_i$  and  $\sigma_i$ , pCollector filter finds a period range that satisfies a certain 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  $\mu_i$  and  $\sigma_i$ . Then, for every possible period value within the range, pCollector filter counts the number of tiny pCollectors that have the period for the value (②), and the period value with the largest number of tiny pCollectors is selected (③). Once a period is selected, 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

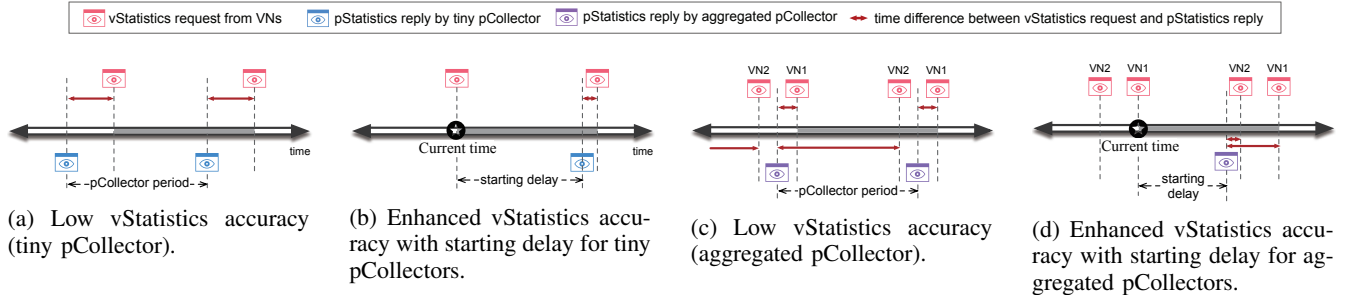


Fig. 11: Starting delay for tiny and aggregated pCollectors.

tiny pCollectors. So, only when the ratio is high, for instance, 70%<sup>3</sup>, (⑤), pCollector tuner merges existing tiny pCollectors into an aggregated pCollector.

2) *pCollector tuner*: The role of pCollector tuner is to give additional delay to the first execution of each pCollector in order to improve the accuracy of vStatistics. In Fig. 11a, a time difference exists between the time when vStatistics requests arrive and the time when pStatistics are gathered through pCollector. This time difference depends on the time when the pCollector first runs. If the pCollector is executed slightly before the vStatistics request, the time difference will become small as shown in Fig. 11b, which means that the cached pStatistics are up-to-date. As the time difference becomes bigger, it hurts the accuracy of vStatistics. Therefore, V-Sight introduces ‘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 just before the vStatistics requests (coming after interval window). Also, the starting delay should not be too large to avoid the pCollector to be executed after the vStatistics request as Fig. 11a. Empirically, we set the starting delay at 95% of the pCollector period (⑥-1 in Fig. 10).

On the other hand, for aggregated pCollectors, the way of setting the starting delay for tiny pCollectors rather leads to poor accuracy. This is because the multiple requests that are handled by an aggregated pCollector exist at different times in terms of the pCollector period. Figure 11c illustrates an example with two vStatistics requests from different VNs (VN2 followed by VN1). If the starting delay is set to 95% of the aggregated pCollector, the execution time of the aggregated pCollector can be after VN2 and before VN1. As Fig. 11c shows, VN2 suffers a long delay because the aggregated pCollector executes right after VN2’s request.

Therefore, pCollector tuner sets the starting delay for the aggregated pCollector as follows. First, pCollector tuner checks request interval estimation (§III-C2) which stores the vStatistics request times for each VN (⑥-2 in Fig. 10). Then, the starting delay is set to be right before the first vStatistics request among the VN requests that the aggregated pCollector merged, which is the VN2’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

<sup>3</sup>In our evaluation, we run different ratio values and find that the improvement comes from 70%.

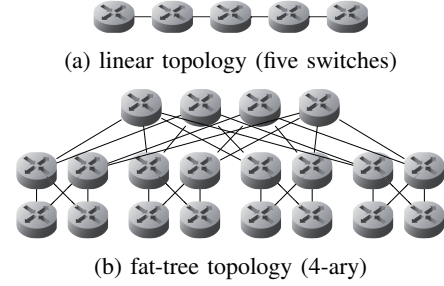


Fig. 12: Experiment topologies.

is minimized. Finally, pCollector tuner executes the pCollector periodically with the starting delay (⑦).

#### IV. EVALUATION

In this section, we show the evaluation results of V-Sight. V-Sight is implemented on OVX v0.1 as independent modules (1.8K LoCs). We measure transmission delay, control channel consumption, and accuracy overheads that are explained in detail below. Each experiment is repeated for more than 40 measured results.

##### A. Test Setup

We use three physical servers with Intel Xeon E5-2600 and 64 G memory. Each server runs Mininet [21], OVX with V-Sight implementation, and ONOS as VN controllers, respectively. Mininet emulates PN based on Open vSwitch, and we emulate two kinds of topologies (Fig. 12): 1) linear topology consisting of five switches and 2) 4-ary fat-tree topology to evaluate the effects for datacenters. We create three VNs and one VN for fat-tree and linear topologies, respectively. Each VN is managed by an ONOS controller. ONOS monitors all the flow entries and ports of each switch at 5 s interval (default settings). We generate TCP connections through the iperf3 [22], and stress V-Sight by varying the number of TCP connections in PN (e.g., 6 to 30 for linear and 2 to 32 for fat-tree topologies).

We evaluate V-Sight performance on the following metrics:

- **Transmission delay**: Average interval between vStatistics request and reply messages from/to VN controllers.
- **Control channel consumption**: Average bytes per second of control channel traffic to get *pf* statistics between the NH and the physical switches.

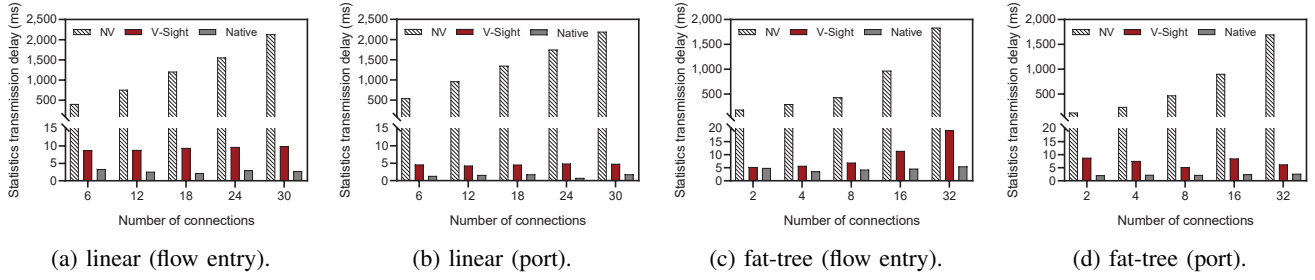


Fig. 13: Average statistics transmission delay (ms).

- **Accuracy overheads:** Time difference between vStatistics request time and pStatistics collection time of pCollectors – Average value with 95% confidence interval.

The above metrics are measured in the following cases:

- **NV:** the basis of experiments with OVX similar to the implementation in §II-C2.
- **V-Sight:** OVX with the full implementation of V-Sight.
- **Native:** Non-virtualized SDN in which physical switches are directly connected to ONOS without NH.

## B. Transmission Delay

1) *V-Sight improvement:* Figure 13 shows that transmission delay in NV increases significantly in proportion to the number of TCP connections (even over 2 s). However, V-Sight disaggregates pStatistics transmission routines from vStatistics virtualization, thereby reducing this delay. In linear topology, V-Sight consumes 9.35 ms and 4.68 ms, on average, for flow entry and port statistics transmission, respectively (Fig. 13a and Fig. 13b). The delays in V-Sight improve 46 times (flow entry statistics, six connections) to 454 times (port statistics, 30 connections) compared to those in NV. For fat-tree topology, V-Sight takes 9.75 ms and 7.29 ms of transmission delay for flow entry and port, respectively (Fig. 13c and Fig. 13d). V-Sight's delay improves 14 times (port statistics, two connections) to 269 times (port statistics, 32 connections).

In detail, the transmission delay in NV increases in proportion to the number of TCP connections, since the number of pStatistics needed for vStatistics increases as the connection numbers increases. Then, vStatistics are returned to the VN controller only after the corresponding pStatistics are collected. In contrast, V-Sight disaggregates pStatistics transmission routines from vStatistics virtualization, thereby reducing this delay.

2) *Virtualization overheads:* We compare the transmission delay between V-Sight and Native to see the virtualization overheads. In linear topology, Native takes 2.8 ms and 1.5 ms for flow entry and port statistics transmission delay, respectively (Fig. 13a and Fig. 13b). The delays in V-Sight are 3.4 times higher, on average, than those of Native. Also, for the fat-tree topology, Native shows 4.6 ms and 2.37 ms delays for flow entry and port statistics transmission (Fig. 13c and Fig. 13d), respectively. The results of V-Sight are 1.09 times (flow

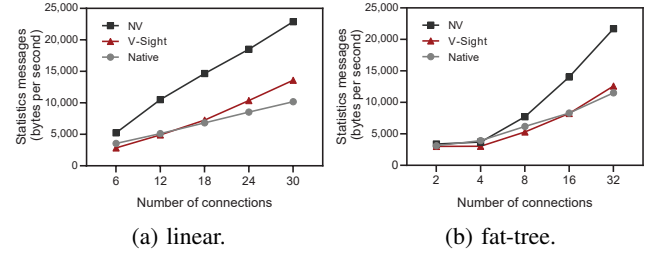


Fig. 14: Control channel traffic usage (bytes/second).

statistics, two connections) to 6.69 times (port statistics, two connections) higher than Native.

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

## C. Control Channel Consumption

1) *V-Sight improvement:* Figure 14 shows control channel consumption for both topologies. The consumption increases in proportion to the number of connections because the monitoring of *pfs* increases with the number of connections. In linear topology (Fig. 14a), V-Sight improves the control channel consumption about 1.9 times, on average. In fat-tree topology (Fig. 14b), the average consumption of V-Sight is 1.44 times less than NV. This improvement is due to the benefit of aggregated pCollector that merges the multiple packet headers (§III-D).

2) *Virtualization overheads:* Comparing V-Sight with Native, V-Sight consumes 107% and 93% of control channel traffic in linear and fat-tree topologies, respectively, which means that the consumption of V-Sight is quite comparable to that of Native. Furthermore, in fat-tree topology with few network connections, V-Sight is even better than Native. The reason is that V-Sight only monitors switches that have the pStatistics needed for vStatistics. Fat-tree topology has 20 switches (Fig. 12b), and multiple paths between each host pair are available. In this topology, when the number of connections is small, not all switches are used for packet forwarding and so not for vStatistics.

In Native, however, the VN controller monitors all the switches in PN. So, request and reply messages are regularly



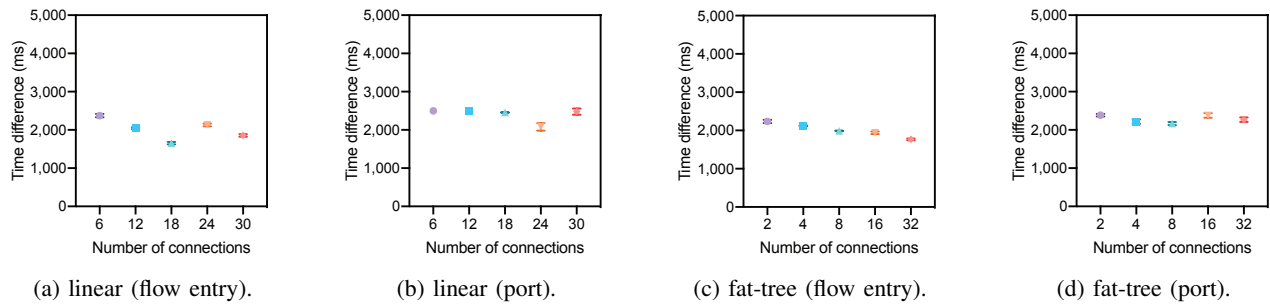


Fig. 15: Time difference between the vStatistics and pStatistics (ms).

generated for all switches. In V-Sight, transmission disaggregation controls the creation of pCollectors toward required *pfs*. Thus, pCollectors are created only for the required *pfs* and, thus, statistics request/reply messages are not created for switches not used.

#### D. Accuracy Overheads

V-Sight successfully improves transmission delay and control channel consumption. The improvements come with the accuracy overhead inevitably because disaggregated transmission and pCollector aggregation incur time difference between when vStatistics requests come and when pStatistics become available by pCollectors. Figure 15 shows the time differences for the number of network connections in linear and fat-tree topologies, plotted with average and 95% confidence interval. The results indicate that the average time differences in all cases of network connections are equal to or less than 2,500 ms in both topologies.

This means that V-Sight replies to the VN controller within 2.5 s which is half of the request interval of VN controllers. Thus, the VN controller, at least, does not receive statistics from the previous statistics request, and therefore the accuracy overhead does not jeopardize the accuracy of vStatistics itself.

### V. RELATED WORK

#### A. Monitoring in non-virtualized SDN

Various studies have been proposed for reducing monitoring overheads in SDN. We mention the notable ones due to this paper's space limitation. FlowSense and PayLess [23], [24] introduced a technique for reducing the number of statistics transmissions using the control for flow removal of OpenFlow. MicroTE [25] implemented monitoring and traffic engineering functions on a separate machine from SDN controllers, so the bottleneck is removed from the controller itself. OpenSketch [17] introduced a hash-based switch measurement architecture to reduce the number of monitoring trials with a multistage hash-based switch measurement architecture. Zhang [26] introduced a prediction-based technique that balances the monitoring aggregation and accuracy.

All the summarized studies investigated non-virtualized SDN. Also, the ways of reducing overheads and retaining statistics accuracy are not suitable with SDN-NV.

#### B. Monitoring in SDN-NV

FlowVisor [1] introduced the first idea of NV in SDN, and FlowN [2] provided more scalable NH structure based on containers. OVX [3] defined the address virtualization schemes, and CoVisor [27] designed a policy composition framework for a network to be managed by heterogeneous SDN controllers. OnVisor [9] developed OVX functionalities on ONOS, enabling the deployment of NH into a physically distributed architecture.

Although monitoring is a well-investigated topic in SDN, to the best of our knowledge, there has not been any paper on network monitoring for SDN-NV. Without monitoring, operation and optimization for VN management are severely hampered.

### VI. 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 the control channel consumption. To this end, V-Sight introduces statistics virtualization, transmission disaggregation, and pCollector aggregation. We fully implement V-Sight and evaluate its key performance characteristics in terms of transmission delay and control channel consumption. Furthermore, we evaluate the accuracy overhead to validate that V-Sight does not compromise the accuracy of network monitoring. The results show that V-Sight attains a level comparable to network monitoring in non-virtualized SDN.

As future research, we plan to extend V-Sight to cover P4's in-band telemetry. Also, based on the isolated and timely statistics from V-Sight, we will research the reliability and performance of traffic engineering.

### ACKNOWLEDGMENT

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

## REFERENCES

- [1] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar, "Can the production network be the testbed?" in *OSDI*, vol. 10, 2010, pp. 1–6.
- [2] D. Drutskey, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2012.
- [3] 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*. ACM, 2014, pp. 25–30.
- [4] "POX Controller." [Online]. Available: <https://noxrepo.github.io/pox-doc/html/>
- [5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.
- [6] 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.
- [7] 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.
- [8] G. Yang, B.-Y. Yu, S.-M. Kim, and C. Yoo, "LiteVisor: A network hypervisor to support flow aggregation and seamless network reconfiguration for VM migration in virtualized software-defined networks," *IEEE Access*, vol. 6, pp. 65 945–65 959, 2018.
- [9] Y. Han, T. Vachuska, A. Al-Shabibi, J. Li, H. Huang, W. Snow, and J. W.-K. Hong, "ONVisor: Towards a scalable and flexible SDN-based network virtualization platform on ONOS," *International Journal of Network Management*, vol. 28, no. 2, p. e2012, 2018.
- [10] 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.
- [11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.
- [12] 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.
- [13] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 2, pp. 42–50, 2015.
- [14] S. Shirali-Shahreza and Y. Ganjali, "Empowering software defined network controller with packet-level information," in *2013 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2013, pp. 1335–1339.
- [15] 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.
- [16] 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.
- [17] 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)*, 2013, pp. 29–42.
- [18] C. Liu, A. Malboubi, and C.-N. Chuah, "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 47–52.
- [19] OPEN NETWORKING FOUNDATION, "OpenFlow Switch Specification," 2012. [Online]. Available: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>
- [20] 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*. ACM, 2016, pp. 1–6.
- [21] 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*. ACM, 2010, p. 19.
- [22] V. GUEANT, "iPerf." [Online]. Available: <https://iperf.fr/>
- [23] 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.
- [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] 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*. ACM, 2011, p. 8.
- [26] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 25–30.
- [27] 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)*, 2015, pp. 87–101.