IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 25, 2019, accepted July 9, 2019, date of publication July 19, 2019, date of current version August 6, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2929958

WhiteRabbit: Scalable Software-Defined Network Data-Plane Verification Method Through Time Scheduling

TAKAHIRO SHIMIZU[1], **(Student Member, IEEE), NAOYA KITAGAWA**[2], **(Member, IEEE), KOHTA OHSHIMA**[3], **(Member, IEEE), AND NARIYOSHI YAMAI**[2], **(Member, IEEE)**

[1]Department of Computer and Information Sciences, Graduate School of Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan
[2]Division of Advanced Information Technology and Computer Science, Department of Institute of Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan
[3]Department of Marine Electronics and Mechanical Engineering, Tokyo University of Marine Science and Technology, Tokyo 135-8533, Japan

Corresponding author: Takahiro Shimizu (tshimizu@net.cs.tuat.ac.jp)

This work was supported by JSPS KAKENHI Grant Number JP19K20252.

ABSTRACT Software-defined networks are vulnerable to attacks by compromised switches because commonly used programmable software switches are risky than traditional hardware ones. Although several countermeasures have been proposed to address compromised switches, the accuracy of detecting a malicious behavior depends on the performance of network statistics gathering by a controller. In this paper, we propose that WhiteRabbit is an approach to verify the consistency of the forwarding state by gathering real-time network statistics gathering from switches with accurate time scheduling. WhiteRabbit can detect attacks by compromised switches without being influenced by the performance of statistics gathering of a controller. Given that the proposed utilizes moving average, it mitigates the effect on the verification accuracy from the impact of the switch performance, such as scheduling error. In our previous work, we demonstrated the feasibility of WhiteRabbit using a prototype system. However, we could not evaluate the impact of the difference between the scheduled and actual execution times in our previous work, because we performed the experiment in a minimal setup using Mininet. Thus, we measured the scheduling error and time required to gather statistics in a large-scale environment. We also confirmed that the scheduling error is lower than the time required to gather statistics. Additionally, considering that WhiteRabbit only depends on the scheduling error, we verified that the accuracy of WhiteRabbit is higher than prior arts on the tree topology constructed with 15 switches.

INDEX TERMS Data-plane verification, software-defined network (SDN), scheduled bundle, statistics gathering, precision time protocol (PTP).

I. INTRODUCTION

A. BACKGROUND

Network attackers may compromise switches by abusing software or hardware vulnerability, and hence, networks are prone to attacks owing to these compromised switches. Cisco reported that routers in 318 models have the vulnerability to possibly encounter compromised switches by invoking a simple command [2]. The possible signs of a compromised switch are drop, delay, and deviation of packets.

In particular, several papers indicate that a software-defined network (SDN) often uses programmable software switches and has higher compromise probability than traditional hardware switches [3]–[6]. For instance, CVE-2016-2074 [7] reported that attackers can execute an arbitrary code by abusing the vulnerability of the buffer overflow in Open vSwitch. Dhawan *et al.* [4] reported that popular SDN controller applications cannot detect a malicious network behavior. Thus, protection of the data plane in SDN is more important than that in traditional networks.

Byte consistency check, which was proposed in SPHINX [4], is a countermeasure against suspicious behaviors of compromised switches such as packet drop and injection.

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaofei Wang.

However, it has a problem because verification accuracy depends on the statistics gathering performance of a controller. Moreover, an alternative solution based on trajectory sampling, referred to as WedgeTail [5], has been proposed recently. Although WedgeTail can be verified with higher accuracy than SPHINX, additional resources are required to perform its verification.

In the flow updating field of SDN, Time4 [8] was proposed as a method to update the flow using Scheduled Bundle. Scheduled Bundle is a method used to schedule the time of executing OpenFlow messages at the switches. Time4 demonstrated that simultaneous updating of flow by Scheduled Bundle can suppress packet loss and without the performance degradation at *Flow Swapping* scenario. Furthermore, given that Scheduled Bundle supports all OpenFlow messages, we considered that utilizing this method can gather statistics without relying on the controller performance through time triggered OpenFlow message execution.

In this study, we propose WhiteRabbit, which is a novel approach to verify forwarding state consistency. Our approach uses the statistics gathered from the switches by Scheduled Bundle simultaneously. Although WhiteRabbit schedules the task to gather transfer statistics in each switch, it is not affected by the controller performance and network size (Section III). Thus, the proposed method can detect attacks by compromised switches without relying on the performance of controller statistics gathering.

In our previous study, we reported a prototype system was implemented in our method in Mininet [1]. However, because the experimental environment in Mininet was constructed in a single machine, our previous work has three issues. First, we could not perform an experiment in other environments aside from a small linear topology setup. Second, the previous work assumed that the time of each emulated node is synchronized using the same hardware clock. Hence, the experiment could not evaluate the effect of time synchronization in a real-world environment. Third, the previous experiment lacked the analysis of performance attributes, such as scheduling error and time required to send the messages. Therefore, we need further experiments in a large scale environment to evaluate the impact of performance attributes, overhead of WhiteRabbit, and accuracy of the validation.

Considering that the controller cannot gather statistics from switches simultaneously, the switches send the transfer statistics immediately within the scheduled time. Hence, time here is regarded as the scheduling error. The scheduling error is affected by two factors: clock accuracy and execution accuracy. Although the clock accuracy can typically achieve one microsecond order, the execution accuracy strongly depends on the implementation of the switches. We showed that the execution accuracy achieves one millisecond order in the software-based switch of our experimental environment (Section IV-A.4). Then, we confirmed that the scheduling error is lower than the time required to send messages to all switches. Although WhiteRabbit depends on the scheduling error, the error is lower than the time required to send messages when the untimed approach is used, such as SPHINX. Thus, WhiteRabbit has the scalability advantage.

Moreover, we evaluated the verification accuracy of WhiteRabbit in a 25-node environment implemented on the DeterLab testbed. Because WhiteRabbit is not affected by the network scale and controller performance through scheduling to gather statistics of each switch, we confirmed that it has a lower false-positive rate than SPHINX (Section IV-B.1). Thus, our experiments showed that WhiteRabbit can efficiently detect attacks without depending on controller performance when compared with SPHINX.

This paper makes the following contributions.

- We present the sequence of the statistics gathering with time scheduling utilizing Scheduled Bundle in WhiteRabbit. By gathering the transfer statistics using Scheduled Bundle simultaneously, WhiteRabbit can gather the statistics without depending on controller performance.
- We present the verification algorithm for using the statistics gathered in scheduled time based on the byte consistency check in SPHINX.
- We examine the elapsed time for sending messages to all switches, and the scheduling error. We also report the scheduling error is lower than the time required in sending the messages to all switches.
- We evaluate WhiteRabbit to show that it can achieve lower false positives than SPHINX, and it does not affect packet transferring.

The rest of this paper is organized as follows. Section II summarizes the related work. Section III depicts our system overview, threat model, and workflow of WhiteRabbit in detail. Section IV evaluates WhiteRabbit by comparing it with the existing one. Section V, discusses the limitations, impact of the scheduling accuracy, and future work. Finally, Section VI provides the conclusion of the entire study.

II. RELATED WORK

A. SOFTWARE DEFINED NETWORK

An SDN has network control and packet forwarding function that enables flexible network control using a centralized control plane. Network intelligence is logically centralized in the trusted software-based controller that maintains the global view of the entire network, and the packet forwarding function comprises hardware and software switches, which are dumb forwarding device.

OpenFlow [9] is a protocol used to realize SDN using controllers and switches.

The OpenFlow messages relevant to this study include `FLOW_MOD`, `STATS_REQUEST`, and `STATS_REPLY`.[1] The `FLOW_MOD` messages create flow entries in the switches,

[1] In OpenFlow 1.3 later, `STATS_REQUEST` and `STATS_REPLY` messages are renamed to `MULTIPART_REQUEST` and `MULTIPART_REPLY`, respectively. However, because these messages are called `STATS_REQUEST` and `STATS_REPLY` messages in [4], we renamed these messages `STATS_REQUEST` and `STATS_REPLY` messages, respectively, in this paper.

and the STATS_REQUEST messages request the statistics to the switches from a controller. As a response to the STATS_REQUEST messages, the STATS_REPLY messages report the network statistics in the switches related to flow, table, and switch port, such as the number of packets and bytes sent or received.

B. SDN AND TIME

The Scheduled Bundle proposed in Time4 [8], a method to schedule the execution timing of some OpenFlow messages execution in switches, is used without depending on the controller performance. Scheduled Bundle is a flexible method that is compatible with all types of OpenFlow messages. The specification of OpenFlow 1.5 [10] also includes Scheduled Bundle.

To execute scheduled messages simultaneously, Time4 utilizes high precision time synchronization, such as Precision Time Protocol (PTP) to be standardized by IEEE 1588 [11]. PTP can synchronize nanosecond order time synchronization using a hardware-timestamping enabled NIC module. Time4 shows that simultaneous updating of flow can suppress packet loss and eliminate performance degradation at the *Flow Swapping* scenario. Mizrahi and Moses [8] reported that nine out of the 13 SDN capable switch silicones listed in the Open Networking Foundation SDN Product Directory have native IEEE 1588 support.

C. TRADITIONAL NETWORK DATA PLANE AND SECURITY

Many researchers reported that detecting compromised switches is an important issue in a traditional network [12]–[15]. Thus, they proposed solutions to detect these compromised switches. The objective of these solutions is to perform the verification in the switches themselves. For example, OPT [14] uses a cryptography approach with key exchange between switches. Therefore, they are not suitable for SDNs, which are maintained by the centralized controller, because these solutions work on the switches.

D. SDN SECURITY AND NETWORK VERIFICATION

The SDNs' paradigm leads to software flexibility to the networks. However, SDN is as vulnerable to compromised switches as traditional networks. Several studies showed that SDN is more vulnerable to compromised switches than traditional networks and confirmed that attackers can even have control of an entire network with compromised switches [3]–[6]. Hence, countermeasures against compromised switches in SDN are required.

To verify network configuration, tools such as VeriFlow [16] and NetPlumber [17] are proposed by several studies. However, these studies only focus on the detection of network bugs, such as loops, but infringed switches are out of scope. These network verification procedures also depends on the traffic information from the switches, which is not regarded as a security issue. Thus, these solutions are not suitable for detecting compromised switches.

SPHINX [4] is the most relevant research on this paper. It is one of the countermeasures against compromised switches and assumes trusted controller and reliable majority switches. SPHINX provides a global view of networks, referred to as flow graph, by collecting FLOW_MOD messages from the trusted controller and then verifies its consistency and constraint. It can also verify the legitimacy of the SDN data plane via byte consistency check with the flow statistics gathered from the switches. Byte consistency check uses Similarity Index (Σ), which is the moving average of byte statistics value regarding flow. Σ must have a similar value to a particular flow of each switch when the networks do not suffer through attacks from compromised switches, such as dropping or injecting packets.

Byte consistency check is a practical countermeasure for attacks by compromised switches, but it has a scalability issue. Given that it influenced by the gap of statistic gathering time between switches, the accuracy of SPHINX's byte consistency check is dependent on the controller performance. In particular, several studies reported that controller performance depends on the number of connected switches [8] and the hardware specification [18]. Furthermore, Curtis *et al.* [19] showed that threshold and sampling-based statistics gathering methods can collect statistics without relying on the controller performance. However, when these methods are applied for data plane verification, they are confirmed to be totally dependent on the timing of the statistics report from untrusted switches. Similarly, Flowmon [20] used the moving average of byte statistics, which also has a scalability issue. Therefore, to apply the byte consistency check to a production environment, operators should improve these solutions independent of the on network scales.

WedgeTail [5] has higher accuracy than SPHINX but requires many resources for gathering packet hash and verifying behaviors of the switches with comparing between except packet behaviors and actual packet behaviors.

Thus, current solutions have issues relating to scalability; for example, the accuracy of verification may depend on the controller performance.

III. SYSTEM DESIGN

A. OVERVIEW

As mentioned in Section II, the existing solutions have scalability issues; for example, the time required to gather statistics depends on the controller performance, and many resources are required. Hence, if the number of switches increases and the controller specification is insufficient, then the verification accuracy may degrade.

In this paper, we propose a data plane verification method that uses the statistics concurrently gathered among all switches through accurate time scheduling. WhiteRabbit can collect statistics without depending on the controller performance, and the controller can simultaneously handle statistics gathering.

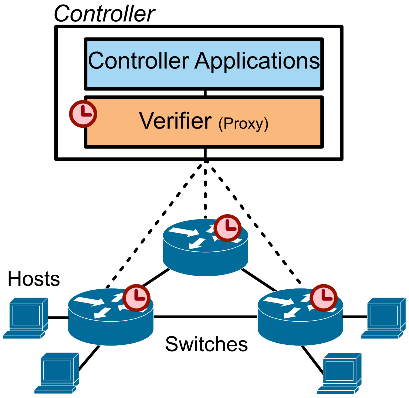


FIGURE 1. System overview.

FIGURE 1 presents the schematic architecture of the system proposed in this study. The part implemented using the proposed method is indicated by Verifier in this figure, and it intercepts the OpenFlow messages between the switches and controller applications. Furthermore, we assume that all switches and controllers are accurately time synchronized.

B. THREAT MODEL

We focus on the data plane security that attacks can detect from packet transfer statistics analysis because the control plane security is well investigated [21]. We assume that the packets of a compromised switch may drop, inject, or delay, and the compromised switch does not accurately handle the packets according to the rules specified by the controller. The cause of these behaviors is probably misconfiguration or switch failure. WhiteRabbit can be utilized not only to detect compromised switch behaviors but also to discover network defects. Regarding the assumption of SPHINX, we assume that the controller applications are trusted and majority of the switches are legitimate. Thus, the messages from the controller are reliable, whereas those messages from any switches may be forged by compromised switches. To focus the analysis on only OpenFlow control messages, we consider that the verifier knows the reliable physical topology information and assume a closed SDN system.

Additionally, we assume that the times of *all* switches and controllers are synchronized accurately through the time synchronization protocol, such as PTP. Although compromised switches synchronized time similar to legitimate switches and disguised the byte transfer statistics, many other switches report legitimate statistics. The difference of byte transfer statistics is consequently higher than that with the compromised and many legitimate switches. Thus, WhiteRabbit can detect attacks by compromised switches (see Section III-C.3). Additionally, we can utilize prior arts, such as [22], as a countermeasure of attacks to the PTP protocol. Therefore, the security of time synchronization is outside the scope of this paper.

C. SEQUENCE OF VALIDATION

FIGURE 2 shows the workflow of WhiteRabbit, which involves three sequences. WhiteRabbit validates whether the packet transmissions are correctly performed on the path as assumed by the trusted controller. Additionally, considering that WhiteRabbit requires us to schedule statistics gathering in real time, we use ReversePTP [23] to collect the time offset between the controller and switches. The validation sequence to a specific traffic flow using WhiteRabbit is as following:

1) Calculate the path that the controller supposes, using physical topology information and `FLOW_MOD` messages sent from the controller application.
2) Obtain the actual transfer statistics of all switches using Scheduled Bundle simultaneously.
3) Validate the transfer state consistency using the expected path of the controller and the difference of statistics among the neighboring switches.

As described in Section III-A, WhiteRabbit intercepts the OpenFlow message, such as `FLOW_MOD` and `STATS_REPLY`, and then relays it to the destination. It also relays OpenFlow messages that are irrelevant to the verification.

From Section III-C.1 through Section III-C.3, we present these mechanisms in detail.

1) CURRENT PATH CULCULATION

WhiteRabbit requires a flow graph, which is a graph theoretic perspective of the network assumed by a trusted controller, to obtain a current path similar to that of SPHINX. The flow graph is constructed only using the `FLOW_MOD` messages issued by the trusted controller. It includes match field and instruction, comprising an src/dst MAC address, src/dst IP address, and in/out port information of the switches. The flow graph does not suffer from untrusted switches because the untrusted `STATS_REPLY` messages are not used in constructing this graph.

The current path assumed by the trusted controller can be obtain by combining information of `FLOW_MOD` messages and physical topology. The current path is used for identifying the switches through which a specific traffic passes when during the verification execution.

2) STATISTICS GATHERING WITH SCHEDULED BUNDLE

WhiteRabbit periodically gathers transfer statistics in real time from the switches. When gathering statistics, a controller generally sends `STATS_REQUEST` messages simultaneously, and the time required to send these messages depends on the controller performance. In particular, WhiteRabbit uses `STATS_REQUEST` wrapped with Scheduled Bundle because it gathers statistics in real time between switches. In addition, it can gather statistics without relying on the controller performance.

WhiteRabbit calculates and uses the scheduling time T_S with the time offset t_i between the switch and controller to obtain transfer statistics on all the switches concurrently. To achieve this objective, we define the scheduling time T_S, which is the time for gathering the transfer statistics from switches. We also define T_{Si} as the actual scheduling time

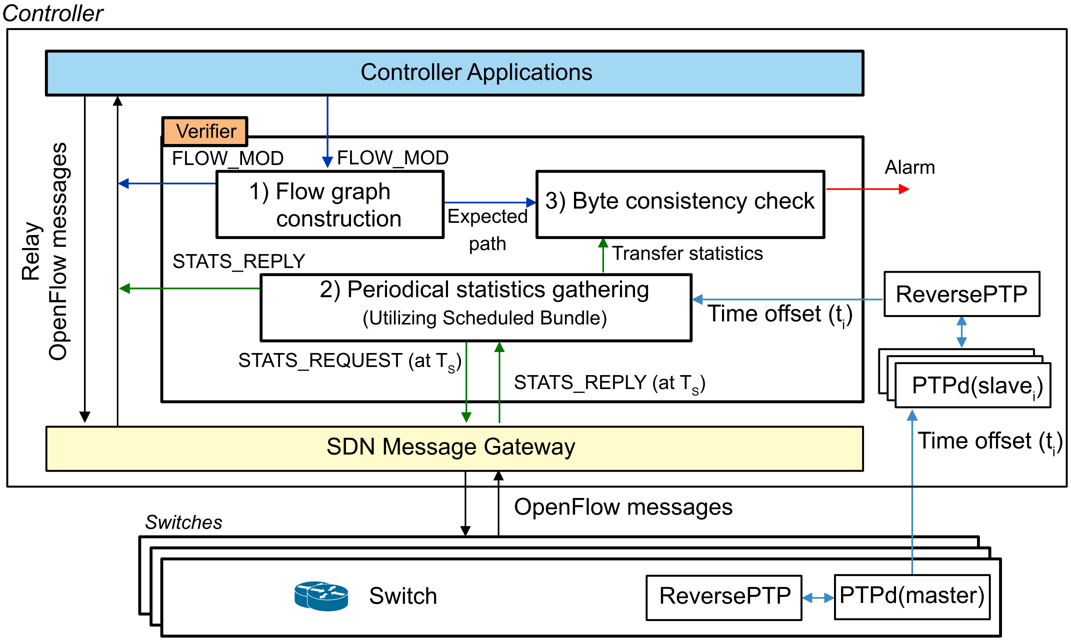


FIGURE 2. **Workflow of WhiteRabbit.**

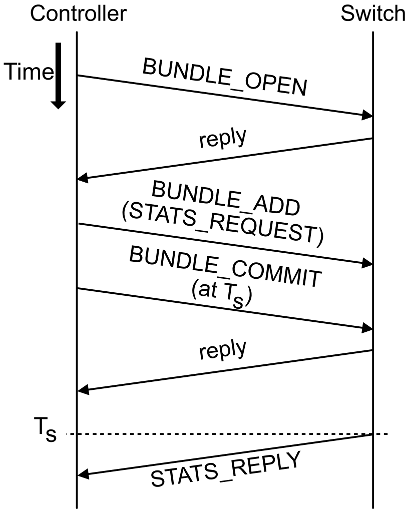


FIGURE 3. **Sequence of the statistics gathering with Scheduled Bundle.**

at the switch$_i$. T_{Si} includes the time offset between the controller and switch i (t_i). In particular, $T_{Si} = T_S + t_i$, and T_{Si} are determined to ensure that they look the same as T_S to the controller. To determine the time offset between the switch$_i$ and controller (t_i), we use ReversePTP [23], which can deliver the time offset between the switches and controller from the switches to the controller. ReversePTP utilizes PTP to deliver the time offset per switch, but cannot sync the time between the switches and controller. It is used only to deliver the time offset between the switches and controller. Therefore, to obtain the time offset t_i and calculate T_{Si} of each switch, WhiteRabbit should schedule when to gather transfer statistics simultaneously among switches.

FIGURE 3 illustrates the flow of gathering transfer statistics at timestamp T_{Si} with Scheduled Bundle. First, the controller sends the `BUNDLE_OPEN` message to the switch, followed by the `BUNDLE_ADD` message that encapsulates the `STATS_REQUEST` message to collect the statistics for all flow entries. Then, the controller sends the `BUNDLE_COMMIT` message with the timestamp of the schedule execution timing T_{Si}.

To verify the byte consistency, WhiteRabbit primarily uses *byte_cnt* and match field information contained in `STATS_REPLY` messages. WhiteRabbit associates the flow and statistics according to match field and calculates the *byte_cnt* difference from the already collected `STATS_REPLY` information.

The statistics difference between the switches may occur owing to the timing of `FLOW_MOD` message sent via the controller application depending on the mechanism of the routing control. Additionally, the statistics of real time gathering rely on the performance of the switches, such as flow table size and schedule execution accuracy. Thus, WhiteRabbit uses moving averages of the difference of the last four statistics report (i.e., use *byte_cnt* difference) simultaneously between switches, referred to as *ByteDiff*. Because this interval is sufficient to eliminate the effects of scheduling errors and traffic bursts, our mechanism can avoid false alarms. Unlike with Σ of SPHINX, given that scheduling mechanism gathers the statistics in real time, *ByteDiff* of WhiteRabbit does not depend on the controller performance.

3) ALGORITHM

Algorithm 1 describes the steps of executing the consistency check using the given flow graph and simultaneously gathered statistics between the switches. The algorithm requires the flow graph as an input, and the flow graph includes a current path relevant to traffic flow F. WhiteRabbit verifies

```
Algorithm 1 Proposed Algorithm
Input: $F$:traffic flow, $\tau$:threshold
Output: $\mathbb{O}$:violation switches of traffic flow$F$
  function Verify($F$,$\tau$)
    Initialize:
      $FG$ := Get_FlowGraph($F$)
      $CurrP$ := Get_CurrentPath($FG$)
      $\mathbb{O} := \emptyset$
      $PrevByte := \infty$
    for all $S \in CurrP$ do
      $FE$ := Get_FlowEntry($S$,$F$)
      $ByteDiff$ := Get_ByteStatsDifference($FE$)
      if True == (($PrevByte == \infty$) $\vee$
         ($PrevByte/\tau < ByteDiff < PrevByte \cdot \tau$)) then
        $PrevByte := ByteDiff$
      else
        $\mathbb{O} := \mathbb{O} \cup S$
    for all $S \in FG \wedge S \notin CurrP$ do
      $FE$ := Get_FlowEntry($S$,$F$)
      $ByteDiff$ := Get_ByteStatsDifference($FE$)
      if $ByteDiff \neq 0$ then
        $\mathbb{O} := \mathbb{O} \cup S$
```

whether the compromised switches attack the network from the same two points as SPHINX.

First, this algorithm validates the statistics of the switches over a current path of the traffic flow F from the nearest switch from a source host. Given that the algorithm uses the concurrent gathered of statistics, all switches that passed through must report the same value of the statistics between the switches. Although a compromised switch disguises that the statistics are similar to a reliable switch, it can be detected using a reliable downstream switch.

Hence, the algorithm needs should consider the difference of the statistics values that occurs owing to propagation delay, scheduling error, and the difference of the flow table sizes maintained by the switches. For that reason, we compare the statistics of neighboring switches based on the validation data already passing through the switches (referred to as *PrevByte*) using a threshold τ. The algorithm reports a violation when it observes a remarkable difference from the concurrent gathered statistics of the neighboring switch over the current path.

Second, the algorithm verifies whether the statistics of the switches that are not included in the current path associated with the traffic flow F are zero. Thus, it can confirm that no traffic has been injected and dropped by the switches that are out of the current path.

The algorithm requires the threshold (τ) as an input, which is used as the margin of the statistics value similarity. Considering that *ByteDiff* varies with communication situation, the algorithm calculates the maximum/minimum *ByteDiff* by multiplying *PrevByte* with the threshold. Additionally, because the performance of statistics gathering depends on the switch performance, which occurs from the schedule execution accuracy, flow table size [19], and switch implementation [24], τ must be determined by considering the switch performance. If the value of τ in this algorithm is significantly large, then false negatives may occur and a genuine alarm may not be outputted. By contrast, if the value of τ is remarkably small, then the algorithm may result in false positives. Therefore, the administrator should thoroughly determine the value of τ.

IV. EVALUATION

A. EXPERIMENTAL SETUP

1) IMPLEMENTATION

We are supported to integrate WhiteRabbit into an application of the controller. However, considering the processing effect for this experiment, we implemented WhiteRabbit as a proxy between the controller application and switches, separate from the controller application. Hence, we executed it in Stopcock [25], which is an implementation of the OpenFlow proxy, and then we used a custom Loxigen [26] script to achieve compatibility with Scheduled Bundle. We utilized the OpenFlow switch ofsoftswitch13_EXT-340 [27], which is compatible with Scheduled Bundle. Moreover, we employed of ReversePTP [28], which is an application used to obtain the time offset between the switches and controller. During implementation, the verifier reads the time offset data asynchronously from ReversePTP and then uses these data to calculate the scheduling time. Because our method schedules the timing of the statistics gathering in the switches, the verification accuracy is not degraded even if integrated into the controller application.

In our experiments, we set that the interval of the statistics gathering is three seconds and the duration of schedule execution is more than one second when the first `BUNDLE_OPEN` message was sent in WhiteRabbit. To compare the results of experiments, we set the interval of the statistics gathering is three seconds in SPHINX as well. Considering that ofsoftswitch13_EXT-340 does not have multiple scheduling, these intervals are sufficient to remove the duplicated schedule.

2) EXPERIMENTAL ENVIRONMENT

To evaluate WhiteRabbit, we adopted a testbed with 25 nodes on DeterLab [29]. FIGURE 4 shows the topology setup in our experiment. Our experiments used a custom tree topology setup that reduces the number of host nodes. This layout is based on tree topology (depth=3, fanout=2). All nodes used are a machine type called MicroCloud in DeterLab. Because both WhiteRabiit and SPHINX use only the statistics gathered from the switches, the verification accuracy of either method is not affected by the number of hosts. Each node has the following roles: switch, host, SDN controller application, verifier component, and SDN message gateway. We also utilized Floodlight v1.1 [30] as an SDN controller application.

The experimental networks in DeterLab were constructed using shared VLAN links. Thus, our experimental environ-

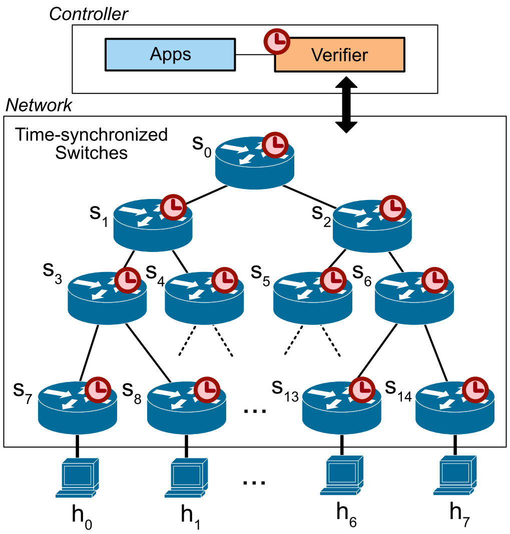


FIGURE 4. Experimental topology.

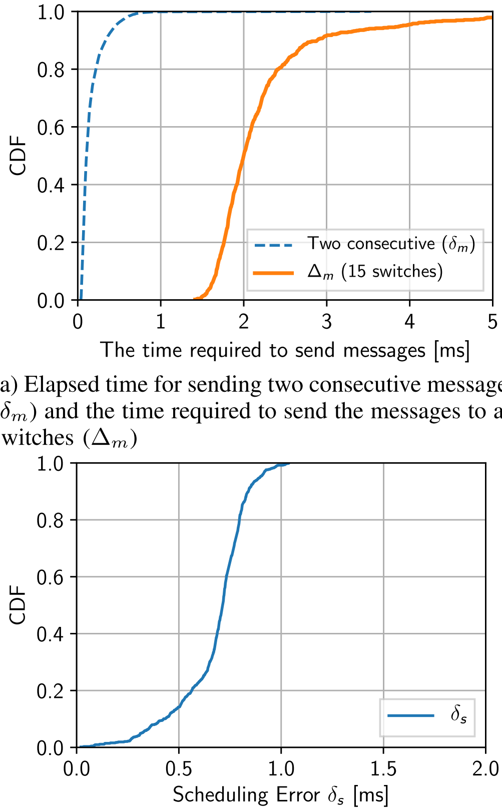
ment has a restriction on PTP synchronization accuracy. (We describe this restriction in Section IV-A.3.) Time synchronization with excellent precision is impossible to achieve because of the experimental environment restriction. Hence, we should evaluate whether WhiteRabbit can be applied to the environment that has synchronized the time with millisecond order as real-world environment.

In particular, this experimental environment is larger than that used in our previous work. In our previous work, we constructed the environment using Mininet on a single machine, which coexisted with the verification component. To determine the impact of resource contention, we used a minimal linear topology and then separated the controller application host and Mininet host including the verification program. By contrast, the experimental node of the environment in this present study is independent of each other. Thus, this experimental environment does not rely with the other nodes, but we must accurately synchronize the time or calculate the time offset between the switches and controller as a real-world environment.

3) PERFORMANCE ATTRIBUTES: δ_M, δ_M AND δ_S

Given that the verification accuracy is influenced by the gap of transfer statistics between switches, influencing the verification accuracy in SPHINX and WhiteRabbit vary. The following three performance attributes serve as key factors in our evaluation: δ_m is the time difference required to send two consecutive messages to two switches, Δ_m is the time required to simultaneously send the messages to all switches in the environment, and δ_s is the difference between the actual and scheduled times when executing statistics gathering.

In an untimed approach, such as SPHINX, depending on the gap that is gathered through the transfer statistics timing simultaneously between switches, δ_m and Δ_m determine the verification accuracy. δ_m indicates the performance of the controller: The SDN controller should handle tens of thousands to millions of packets per second [18], [31], and its performance substantially depends on the controller machine and implementation. Meanwhile, Δ_m denotes the performance of the controller on the supervised network: when sending messages to all switches, such as simultaneous statistics gathered from each switch, the time required to send the messages is dependent on the network size. Thus, the transfer statistics may increase the difference between switches, despite a request to send the messages to gather transfer statistics simultaneously when δ_m and Δ_m increase.



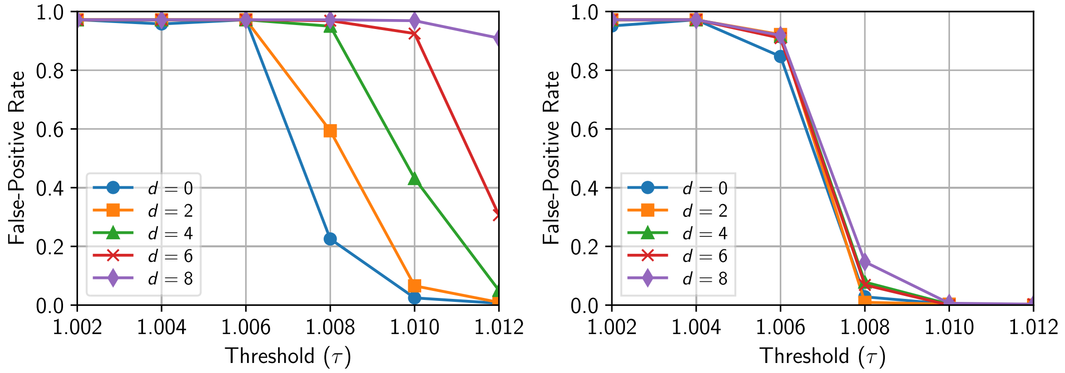
(a) Elapsed time for sending two consecutive messages (δ_m) and the time required to send the messages to all switches (Δ_m)

(b) Scheduling error in our experimental environment (δ_s).

FIGURE 5. Measurement of performance attributes.

Meanwhile, considering that WhiteRabbit supposes to gather concurrent transfer statistics, the scheduling error determines the accuracy. When the transfer statistics gathering is scheduled at time T_0, the execution is basically executed at time $T_0 + \delta_m$. Therefore, the difference in the transfer statistics between switches may occur in the time approach.

The scheduling error is primarily affected by the following two factors mainly: the devices' clock accuracy and execution accuracy. First, the devices' clock accuracy is the clock offset in each node. Clock accuracy depends on the network size, shape of network topology, and synchronization method used. For example, the clock accuracy can realize an order of one microsecond with PTP [11]. Second, the execution accuracy, denoted byte Δ_s, is the measurement of how the device can accurately perform a scheduled task depending on the device specs and the implementation to execute the scheduled task.



(a) SPHINX's false-positive rate with variation in τ and d.

(b) WhiteRabbit's false-positive rate with variation in τ and d.

FIGURE 6. **False-positive experimental results.**

4) MEASUREMENT OF PERFORMANCE ATTRIBUTES

We measured the following performance attributes: δ_m, Δ_m and δ_s. Figure 5a depicts the time required to send two consecutive messages from the controller (i.e., δ_m)m denoted by the blue line, and the time required to send these messages to all switches (i.e., 15) from the controller (i.e., Δ_m), denoted by the orange line. In this measurement, we calculated the time required to continuously send messages from the controller by sending 1K `STATS_REQUEST` messages from the verification program to each switch. Figure 5b shows the results of the experiments measuring the accuracy of scheduling (i.e., δ_s) in this environment. Here, we obtained the difference between the actual and scheduled execution times by sending 1K Scheduled Bundle from the verification program every three seconds to each switch.

We observed two performance attributes, namely, δ_m and δ_s, which are less than one millisecond in 90 percentile. Unlike δ_m in our environment, δ_s is slightly larger than δ_m. However, when the controller needs to send messages to all switches, we must consider Δ_m. We observed Δ_m is larger than δ_s when we are required to send messages to 15 switches. Considering that the scheduling error (δ_s) is independent of each switch, δ_s is not affected by the topology size and workload of the controller when compared with Δ_m. In addition, the authors reported that the scheduling errors in all machines on their testbed were nearly one millisecond. This result is similar to the environment used in Time4 [8].

Our environment uses the software switch wherein the CPU handles the data plane traffic and communication between the controller and switch. Hence, δ_s is affected by the traffic through the switch. Further, WhiteRabbit can mitigate scheduling errors using a moving average of transfer statistics. Therefore, this environment is sufficient for evaluating WhiteRabbit.

B. ACCURACY OF VERIFICATION

We evaluated the verifications with WhiteRabbit and SPHINX and then compared the verification accuracy by measuring the false alarm generated under a benign condition and the lack of real alarms. Because the controller application of our experiment uses Floodlight default simple settings, the workload of the controller application is light and stable. Hence, we simulated the variation of the controller performance by delay (d), which was inserted in the verification program before sending the `STATS_REQUEST` messages. Consequently, d impacts Δ_m. Tootoonchian *et al.* [18] reported that, as the increasing number of connected switches cause I/O handling overhead and resource contention on the task, the latency of the controller response also increased.

Given that the verification algorithm in WhiteRabbit is based on SPHINX, WhiteRabbit can achieve the true positive rate and true negative rate comparable to SPHINX. Hence, we compared false alarms with the absence of actual alarms in our experiments.

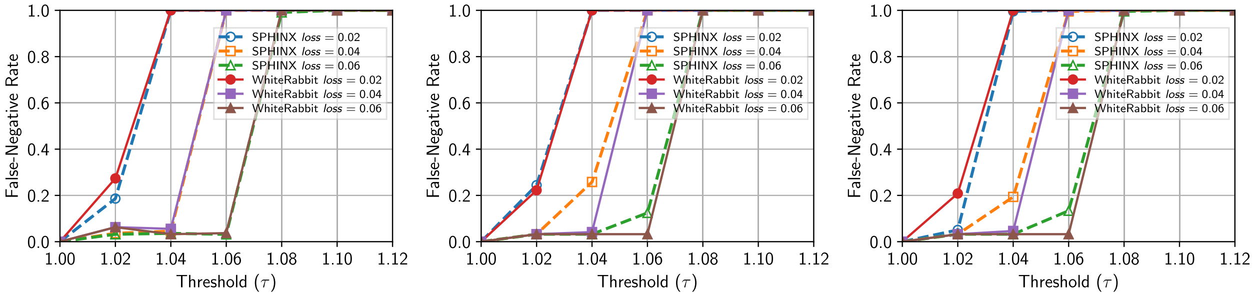
1) FALSE ALARM

We analyzed the probability of false alarm caused by the effect of controller performance degradation using the 3-hop path TCP flows with iperf. In this experiment, raising an alarm for verification is not preferable because all switches are legitimate.

Figure 6a and Figure 6b illustrate the false-positive rate of WhiteRabbit and SPHINX. Noticeably, SPHINX has high false alarm probability evidently when the controller performance degrades (i.e., d increases) as it depends on the controller performance. Meanwhile, we observed that, even if d increases, the false-positive rate of WhiteRabbit is stable, which is similar to SPHINX at $d = 0$. Because WhiteRabbit only depends on δ_s, its false-positive rate does not increase even if d increases, thus affecting Δ_m. Given that δ_s is slightly lower than δ_m, we observed that WhiteRabbit's false-positive rate is slightly lower than SPHINX at $d = 0$.

2) LACK OF REAL ALARM

We evaluated the probability of the lack of an actual alarm given varying controller performance under the same conditions and then compared each method. We performed a packet drop on a 6-hop path link using the given link loss rates,



(a) Comparsion of false-negative rate at $d = 0$ vs τ and link loss rate.

(b) Comparsion of false-negative rate at $d = 5$ vs τ and link loss rate.

(c) Comparsion of false-negative rate at $d = 10$ vs τ and link loss rate.

FIGURE 7. False-negative experimental results.

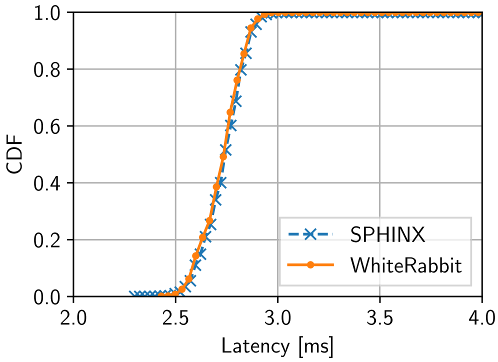


FIGURE 8. Comparison of the ping latencies between WhiteRabbit and SPHINX.

that is, 2%, 4%, and 6%, to emulate malicious behaviors by the compromised switch or link. In this experiment, although alarms generated in all verifications are preferred, false negatives occur when τ increases.

Figure 7a through Figure 7c illustrate the false-negative rates of each method among the three types of controller performance value d ($d = 0, d = 5, d = 10$). We observed that the false-negative rate of WhiteRabbit is slightly higher than that of SPHINX when d increases. However, despite the increase of d, the false-negative rate of WhiteRabbit is nearly constant. The impact of the controller performance in WhiteRabbit is smaller than that of SPHINX. Hence, WhiteRabbit can set smaller τ than SPHINX (see Section IV-B.1) and can achieve equal performance of the false-negative rate to SPHINX by tuning τ.

C. OVERHEADS IN DATA PLANE

We evaluated the performance impact of the packet processing that is affected by statistics gathering from the switches when using WhiteRabbit and SPHINX. Then, we measured the ping latencies in a 6-hop path while gathering statistics every three seconds. FIGURE 8 shows the results of this experiment. As observed in this figure, the ping latencies of WhiteRabbit are similar to those of SPHINX. This result indicates that the ping latencies are not affected even if statistics gathering is performed simultaneously.

In our previous work, the ping latency of WhiteRabbit is partially higher than that of SPHINX for two reasons. First, the implementation of Scheduled Bundle using ofsoftswitch13_EXT-340 affects packet processing. Second, given that the experimental environment of our previous work was constructed in a single machine with Mininet, resource contention had occurred in the experimental environment. By contrast, because the experimental environment used in this study is built independently using multiple machines, the ping latencies in this environment are similar to those of SPHINX.

V. DISCUSSION

A. LIMITATIONS

WhiteRabbit has the following limitations that are similar to SPHINX.

- WhiteRabbit cannot identify whether an ingress or egress switch is compromised or not because the verification accuracy of WhiteRabbit depends on `STATS_REPLY` messages from untrusted switches. Thus, if the edge switch is compromised, then WhiteRabbit cannot detect the attack even if the switch that passes through afterward is legitimate. Therefore, WhiteRabbit cannot be applied to end-to-end verification presently. However, detecting attacks from compromised switches is possible by managing all hosts, as well as the switches, using the verifier, enabling the gathering of statistics from the hosts at a scheduled time.
- WhiteRabbit may omit some transient attacks because the verification span depends on the verification program. To address this limitation, shortening the statistics gathering interval or changing it to a highly accurate network monitoring method, such as WedgeTail [5], is necessary.
- WhiteRabbit cannot verify traffic integrity. However, cryptographic mechanisms can support it to overcome this issue.

B. SCHEDULING ACCURACY

The timed approach, such as WhiteRabbit, remarkably depends on the scheduling accuracy determined from the performance of each switch. Typically, time synchronization can achieve 1-microsecond order precision using PTP. Recently, Kannan *et al.* [32] showed that

Data-Plane Time-synchronization Protocol (DPTP) can achieve 100 nanosecond order time synchronization using a programmable switch ASIC. However, even if the time synchronization accuracy becomes precise, the execution accuracy (described in Section IV-A.3) depends on switch implementation.

In a software-based environment, such as our experimental environment, the scheduling error is 1-millisecond order. However, TimeFlip [33] has shown that the scheduling error in a flow update task can achieve a 1-microsecond order in a hardware-based switch by identifying ways on how to use TCAM. Thus, WhiteRabbit can also be applied to a hardware-based environment using TCAM effectively for scheduled statistics gathering.

C. FUTURE WORK

We intend to reduce the statistics gathering overhead of the Scheduled Bundle by periodic scheduling because our present prototype sends Scheduled Bundle during statistics gathering.

Additionally, the compatibility of WhiteRabbit with distributed controller environment, such as ONOS [34], required further investigation. Although these sites are managed separately, synchronizing the time of all switches and controllers with sufficiently high precision is possible by utilizing a time synchronization method, such as PTP. Thus, we believe that our improved method is suitable for a distributed controller environment. However, because the distributed controller environment may have link delays caused by the distances among the sites, we intend to improve WhiteRabbit by considering the specific delay of a link.

VI. CONCLUSION

SDN is attracting remarkable attention as a future networking paradigm. However, even if the SDN security, such as control plane, is thoroughly investigated, the attacks on the data plane by compromised switches can be a relatively critical threat. Nevertheless, existing solutions have several issues relevant to scalability such as increasing false alarm probability depending on controller performance. In this study, we presented the countermeasure against compromised switches by utilizing the gathered statistics with time scheduling. By gathering statistics simultaneously from the switches through scheduling, WhiteRabbit can detect attacks by compromised switches without depending on the controller performance. Moreover, we demonstrated that the timed approach can mitigate the impact of the time required in sending messages when executing statistics gathering through measurement. Given that WhiteRabbit can mitigate the impact of time required in sending the messages to all switches, we confirmed that the false-positive rate of WhiteRabbit is lower than that of SPHINX when the performance of the controller degrades.

ACKNOWLEDGMENT

This paper was presented at the Proceedings of the 15th APAN Research Workshop 2018 [1].

REFERENCES

[1] T. Shimizu, N. Kitagawa, K. Ohshima, and N. Yamai, "Detecting suspicious behavior of SDN switches by statistics gathering with time," in *Proc. 15th APAN Res. Workshop (APAN-RW)*, Aug. 2018, pp. 32–39.

[2] (2017). *Cisco IOS and IOS XE Software Cluster Management Protocol Remote Code Execution Vulnerability*. Accessed: Jun. 24, 2019. [Online]. Available: https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20170317-cmp/

[3] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2013, pp. 55–60.

[4] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. 22nd Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*. Reston, VA, USA: Internet Society, 2015, pp. 8–11.

[5] A. Shaghaghi, M. A. Kaafar, and S. Jha, "Wedgetail: An intrusion prevention system for the data plane of software defined networks," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (ASIA CCS)*, 2017, pp. 849–861.

[6] A. Feldmann, P. Heyder, M. Kreutzer, S. Schmid, J.-P. Seifert, H. Shulman, K. Thimmaraju, M. Waidner, and J. Sieberg, "NetCo: Reliable routing with unreliable routers," in *Proc. IEEE/IFIP 46th Annu. Int. Conf. Dependable Syst. Netw. Workshop (DSN-W)*, Jun./Jul. 2016, pp. 128–135.

[7] *CVE-2016-2074*. Accessed: Jun. 24, 2019. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2016-2074/

[8] T. Mizrahi and Y. Moses, "Time4: Time for SDN," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 433–446, Sep. 2016.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, 2008.

[10] Open Networking Foundation. (2014). *OpenFlow Switch Specification Version 1.5.0*. Accessed: Jun. 24, 2019. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[11] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Standard 1588-2008, 2008, pp. 1–300.

[12] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Detecting and isolating malicious routers," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 3, pp. 230–244, Jul. 2006.

[13] A. T. Mizrak, S. Savage, and K. Marzullo, "Detecting malicious packet losses," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 191–206, Feb. 2009.

[14] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *Proc. ACM Conf. SIGCOMM (SIGCOMM)*, 2014, pp. 271–282.

[15] S. Lee, T. Wong, and H. S. Kim, "Secure split assignment trajectory sampling: A malicious router detection system," in *Proc. Int. Conf. Dependable Syst. Netw.*, Jun. 2006, pp. 333–342.

[16] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement (NSDI)*, Sep. 2013, vol. 42, no. 4, p. 467.

[17] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement (NSDI)*, 2013, pp. 99–112.

[18] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Workshop Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services (Hot-ICE)*, 2012. [Online]. Available: https:// www. usenix.org/conference/hot-ice12/workshop-program/presentation/tootoonchian

[19] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 254–265.

[20] C. Fung and C. Fung, "FlowMon: Detecting malicious switches in software-defined networks," in *Proc. Workshop Automated Decis. Making Act. Cyber Defense (SafeConfig)*, 2015, pp. 39–45.

[21] S. Khan, A. Gani, A. A. Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 303–324, 1st Quart., 2017.

[22] T. Mizrahi, "Time synchronization security using IPsec and MACsec," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control Commun. (ISPCS)*, Sep. 2011, pp. 38–43.

[23] T. Mizrahi and Y. Moses, "ReversePTP: A clock synchronization scheme for software-defined networks," *Int. J. Netw. Manage.*, vol. 26, no. 5, pp. 355–372, Sep. 2016.

[24] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Passive and Active Measurement* (Lecture Notes in Computer Science), vol. 7192. Berlin, Germany: Springer, 2012, pp. 85–95.

[25] P. Wood. (2014). *Stopcock*. Accessed: Jun. 24, 2019. [Online]. Available: https://github.com/tignetworking/stopcock/

[26] P. Floodlight. (2018). *Loxigen*. Accessed: Jun. 24, 2019. [Online]. Available: https://github.com/floodlight/loxigen/

[27] TimedSDN. (2015). *Ofsoftswitch13_EXT-340*. Accessed: Jun. 24, 2019. [Online]. Available: https://github.com/TimedSDN/ofsoftswitch13_EXT-340/

[28] Tal Mizrahi. (2015). *ReversePTP Implementation*. Accessed: Jun. 24, 2019. [Online]. Available: https://github.com/TimedSDN/ReversePTP

[29] D Project. (2017). *DeterLab*. Accessed: Jun. 24, 2019. [Online]. Available: https://www.isi.deterlab.net/index.php3

[30] (2018). *Floodlight*. Accessed: Jun. 24, 2019. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[31] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," in *Proc. 8th ACM Workshop Hot Topics Netw. (HotNets-VIII)*, 2009, pp. 1–6.

[32] P. G. Kannan, R. Joshi, and M. C. Chan, "Precise time-synchronization in the data-plane using programmable switching ASICs," in *Proc. ACM Symp. SDN Res.*, no. 2, 2019, pp. 8–20.

[33] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Using timestamp-based TCAM ranges to accurately schedule network updates," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 849–863, Apr. 2017.

[34] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*. 2014, pp. 1–6.

TAKAHIRO SHIMIZU received the B.E. degree in computer and information science from the Tokyo University of Agriculture and Technology, in 2018, where he has been a Graduate Student with the Graduate School of Engineering, since April 2018. His research interests include software-defined network (SDN) security and monitoring.

NAOYA KITAGAWA received the B.Sc. and M.Sc. degrees in information science from Chukyo University, Toyota, Japan, in 2009 and 2011, respectively, and the Ph.D. degree in information science from Nagoya University, Nagoya, Japan, in 2014.

In April 2014, he joined the Information Technology Center, Nagoya University, as a Postdoctoral Fellow. Since October 2014, he has been an Assistant Professor with the Institute of Engineering, Tokyo University of Agriculture and Technology. His research interests include the Internet, network security, and distributed systems. He is a member of IPSJ.

KOHTA OHSHIMA received the B.E. and M.E. degrees in electronics and information engineering and the Ph.D. degree from the Tokyo University of Agriculture and Technology, in 2001, 2003, and 2006, respectively.

In 2006, he joined the Faculty of Engineering, Tokyo University of Agriculture and Technology, as a Research Associate. From 2013 to 2015, he was a Senior Lecturer with the Saitama University of Technology, where he was an Associate Professor, in 2016. Since 2016, he has been an Associate Professor with the Faculty of Marine Technology, Tokyo University of Marine Science and Technology. His research interests include mobile computing, marine communication, network science, and network architecture. He is a member of IEICE, IPSJ, and JIN.

NARIYOSHI YAMAI received the B.E. and M.E. degrees in electronic engineering and the Ph.D. degree in information and computer science from Osaka University, Osaka, Japan, in 1984, 1986, and 1993, respectively.

In April 1988, he joined the Department of Information Engineering, Nara National College of Technology, as a Research Associate, where he was an Assistant Professor, from April 1990 to March 1994. In April 1994, he joined the Education Center for Information Processing, Osaka University, as a Research Associate. In April 1995, he joined the Computation Center, Osaka University, as an Assistant Professor. From November 1997 to March 2006, he joined the Computer Center, Okayama University, as an Associate Professor. From April 2006 to March 2014, he was a Professor with the Information Technology Center (at present, the Center for Information Technology and Management), Okayama University. Since April 2014, he has been a Professor with the Institute of Engineering, Tokyo University of Agriculture and Technology. His research interests include distributed systems, network architecture, and Internet. He is a member of IEICE and IPSJ.