Proceedings of the APAN – Research Workshop 2018
ISBN 978-4-9905448-8-1

Detecting Suspicious Behavior of SDN Switches by Statistics Gathering with Time

Takahiro Shimizu, Naoya Kitagawa, Kohta Ohshima, and Nariyoshi Yamai

***Abstract*—In Software Defined Network (SDN), the networks are vulnerable to attacks by compromised switches, since it often used programmable software switches are vulnerable than traditional hardware switches. Although several countermeasures against compromised switches have been proposed, the accuracy of detecting malicious behavior depends on the performance of network statistics gathering by a controller. In this paper, we propose an approach to verify the consistency of forwarding state using simultaneously network statistics gathering from the switch by accurate time scheduling. Our method enables to detect attacks by compromised switches without being influenced by the performance of statistics gathering by the controller. Our method utilizes moving average thus our method mitigates the effect on the verification accuracy from the impact of switches performance such as the error of scheduling. In addition, we implemented the proposed method with Mininet, and we confirmed that our method is able to verify without depending on the performance of statistic-gathering by the controller.**

***Index Terms*—SDN (Software Defined Network), Scheduled Bundle, Statistics Gathering, PTP (Precision Time Protocol).**

I. Introduction

SINCE attackers may compromise switches by abusing software or hardware vulnerabilities, networks are vulnerable to attacks by compromised switches. In particular, several papers indicate that Software Defined Network (SDN) is often used programmable software switches, and increase the probability of compromise than traditional hardware switches [1], [2], [3]. For instance, CVE-2016-2074[4] reports that attacker can execute arbitrary code with abusing the vulnerability of buffer overflow in Open vSwitch. Thus, protection of SDN data plane is more important than traditional networks.

Byte consistency check, which was proposed in SPHINX[2], is a countermeasure against suspicious behaviors by compromised switches such as packet dropping and injection. However, it has an issue that the verification accuracy depends on the performance of statistics gathering of a controller. Moreover, an alternative solution based on trajectory sampling called WedgeTail [3] has proposed recently. Although WedgeTail can be verified with higher accuracy than SPHINX, it requires more resources to perform verification.

In the field of flow updating in SDN, Time 4 [5] was proposed as a method to update the flow simultaneously using Scheduled Bundle. Scheduled Bundle is a method to schedule the timing of executing OpenFlow messages at switches. Time4 showed that updating flow at the same time with Scheduled Bundle can suppress packet loss and without the performance degradation at Flow Swapping scenario. Furthermore, since Scheduled Bundle supports all OpenFlow messages, we considered that utilizing Scheduled Bundle can gather statistics without relying on the performance of controller by time triggered OpenFlow messages execution.

In this paper, we propose a novel approach to verify forwarding state consistency which uses the statistics gathered from switches at the same time by Scheduled Bundle. Our method can detect attacks by compromised switches without being influenced by the performance of controller statistics gathering. In addition, we evaluated this method by micro-benchmarking in an emulated minimal SDN environment implemented in Mininet. Our micro-benchmarking shows that our method can efficiently detect the attacks without depending on controller performance than SPHINX.

II. Related Work

A. Software Defined Network

Software Defined Network (SDN) decouples network

Manuscript submitted May 21, 2018.

Takahiro Shimizu is with Department of Computer and Information Sciences, Graduate School of Engineering, Tokyo University of Agriculture and Technology, Koganei, Tokyo 184-8588 Japan (e-mail: tshimizu@net.cs.tuat.ac.jp).

Naoya Kitagawa is with Division of Advanced Information Technology & Computer Science, Department of Institute of Engineering, Tokyo University of Agriculture and Technology, Koganei, Tokyo 184-8588 Japan (e-mail: nakit@cc.tuat.ac.jp).

Kohta Ohshima is with Department of Marine Electronics and Mechanical Engineering, Tokyo University of Marine Science and Technology, Koto-ku, Tokyo 135-8533, Japan (e-mail: kxoh@kaiyodai.ac.jp).

Nariyoshi Yamai is with Division of Advanced Information Technology & Computer Science, Department of Institute of Engineering, Tokyo University of Agriculture and Technology, Koganei, Tokyo 184-8588 Japan (email: nyamai@cc.tuat.ac.jp).

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

OpenFlow [6] is a protocol to realize SDN using controllers and switches. OpenFlow messages relevant to this paper include FLOW_MOD, STATS_REPLY, and STATS_REQUEST. FLOW_MOD messages create flow entries in switches. STATS_REQUEST messages request the statistics to switches from a controller. As a response to STATS_REQUEST messages, STATS_REPLY messages report network statistics in switches about flow, table, and switch port, such as the number of packets and bytes sent or received.

B. *SDN and Time*

Scheduled Bundle proposed in Time4 [5] is a method to schedule the timing of some OpenFlow messages execution in switches and achieves without depending on the controller performance. Scheduled Bundle is a flexible method to be compatible with all types of OpenFlow messages. Furthermore, OpenFlow 1.5 specification [7] 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 [8]. PTP can synchronize nanosecond order time synchronization by using hardware-timestamping enabled NIC module. Time4 shows that updating flow at the same time can suppress packet loss and without depend on the performance degradation at Flow Swapping scenario. Additionally, T. Mizrahi et al. [5] reported 9 out of the 13 SDN capable switch silicones listed in the Open Networking Foundation (ONF) SDN Product Directory have native IEEE 1588 support.

C. *SDN Security and Network Verification*

Several studies showed that SDN is more vulnerable to compromised switches than traditional networks [1], [2], [3]. To verify network configuration, several studies proposed such as VeriFlow [9] and NetPlumber [10]. However, these studies focus on the detection of network bugs such as loops but infringed switches are out of scope. Hence, countermeasures against compromised switches in SDN are required.

SPHINX [2] is one of the countermeasures against compromised switches, and it assumes trusted controller and honest majority switches. SPHINX creates a global view of networks, called flow graph, by collecting FLOW_MOD messages from the trusted controller, and verifies its consistency and constraint.

Moreover, SPHINX can verify the legitimacy of SDN data plane by byte consistency check with flow statistics gathered

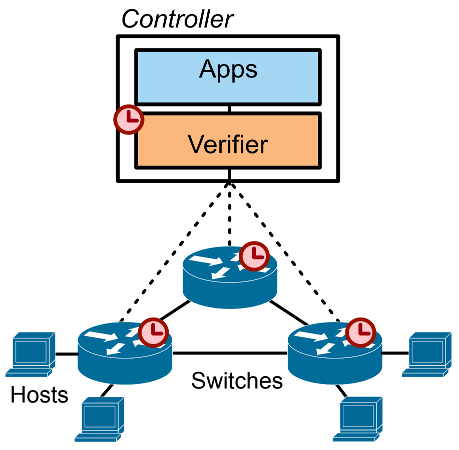


Fig. 1: System Overview.

from switches. Byte consistency check uses Similarity Index (Σ), which is the moving average of byte statistics value regarding flow. Σ must be similar value as to a particular flow each switch when the networks do not suffer from attacks from compromised switches such as dropping or injecting packets.

Byte consistency check is a practical approach to the countermeasure for attacks by compromised switches, but it has a scalability issue. Since it influenced by the gap of statistic gathering timing between each switch, the accuracy of SPHINX's byte consistency check is controller performance dependent. In fact, several studies reported that controller performance depends on the number of connected switches [11] and the hardware specification [5]. Furthermore, A. Curtis et al. [12] showed that threshold-based and sampling-based statistics gathering method can gather statistics without depending on the controller performance. However, unfortunately, when these methods apply for data plane verification, it has an issue totally dependent on the timing of statistics report from untrusted switches.

WedgeTail [3] has higher accuracy than SPHINX but also needs many resources due to gather packet hash and to verify behaviors of switches with comparing except and actual packet behaviors.

Thus, current solutions have issues relating scalability such as the accuracy of verification may depend on controller performance.

III. System Design

A. *Overview*

As mentioned in Section II, the existing solutions have scalability issues such as depends on controller performance, needs many resources. Hence, if the number of switches increase and a controller specification is insufficient, the accuracy of verification may decrease.

In this paper, we propose a data plane verification method which uses the statistics gathered at the same time between all switches by accurate time scheduling. Our method enables collecting statistics without depending on the controller performance, and the controller can handle statistics gathering simultaneously.

Fig. 1 presents a schematic architecture of the system

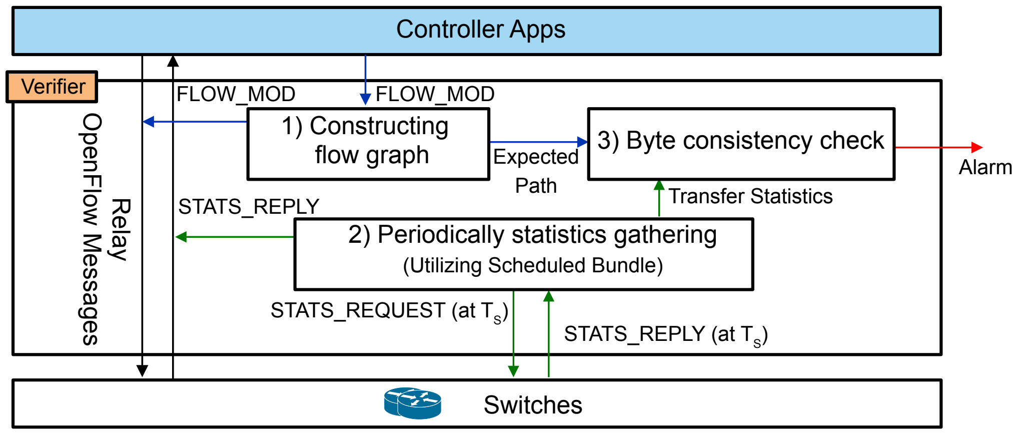


Fig. 2: Our method's workflow.

assumed in this paper. The part implemented by the proposed method is indicated by Verifier in this figure, and it intercepts OpenFlow messages between the switches and the controller applications. Furthermore, we assume that all the switches and all the controllers are accurately time synchronized.

B. *Threat Model*

We focus on data plane security that attacks can detect from packet transfer statistics analysis as control plane security is well studied [13]. We assume that a compromised switch may drop, inject, or delay packets, and it does not handle the packets according to the rules specified by the controller correctly. The cause of these behaviors may be misconfiguration or switch failure. Our method can be utilized not only to detect compromised switch behaviors but also to discover network defects. As with the assumption of SPHINX, we assume that the controller applications are trusted and majority of switches are legitimate. In other words, messages from the controller are trusted, in contrast, messages from any switches may be forged by compromised switches. We consider that the verifier knows reliable physical topology information, and assume closed SDN system, since to focus analysis on only OpenFlow control messages.

Additionally, we assume that the time of all switches and the controller are synchronized accurately by time synchronization protocol such as PTP. Even if a compromised switches synchronized the time like legitimate switches and disguise the byte transfer statistics, many other switches report legitimate statistics. Consequently, since the difference of byte transfer statistics arises compared with the compromised switches and many legitimate switches, our method can detect attacks by compromised switches (See Section III-C3). Additionally, for the countermeasure of attacks to PTP protocol, we can utilize prior art such as [14]. Therefore, the security of time synchronization is out of scope in this paper.

C. *Sequence of Validation*

Fig. 2 shows the workflow of our method, which involves three sequences. Our method validates whether the packet transmissions correctly performed on the path, which supposes by the trusted controller. The sequence of validation to specific traffic flow by our method is the following:

1) Calculate the path that the controller supposes, using physical topology information and FLOW_MOD messages which send from the controller application.
2) Get actual transfer statistics of all switches at the same time by using Scheduled Bundle.
3) Validate transfer state consistency using the expected path by the controller, and difference of statistics between neighbor switches.

As described in Section III-A, our method intercepts OpenFlow message, such as FLOW_MOD message and STATS_REPLY messages, then relay to the destination. Our method also relays OpenFlow messages which does not relevant to the verification.

In Section III-C1 through Section III-C3, we present these mechanisms in detail.

1) *Calculate Current Path*

Our method needs to construct a flow graph, which is a graph theoretic represent of network assumed by a trusted controller, to obtain a current path, similar to SPHINX. The flow graph constructs only using FLOW_MOD messages issued by the trusted controller. It includes match field and instruction, which contains src/dst MAC address, src/dst IP address, and in/out port information of the switches. The flow graph is not suffering from untrusted switches since untrusted STATS_REPLY messages do not use in constructing flow graph.

The current path assumed by the trusted controller can obtain by combinations of the information of FLOW_MOD messages and physical topology information. The current path uses for identifying the switches through which specific traffic passes when verification execution.

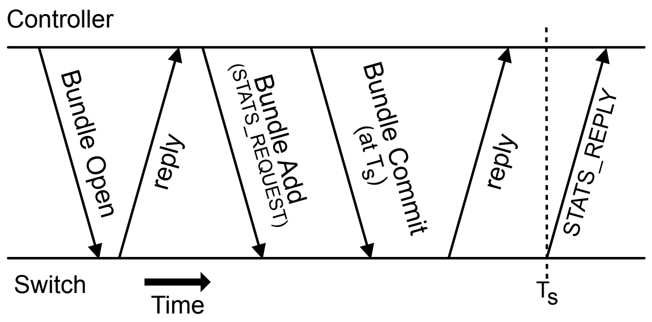


Fig. 3: Sequence of Statistics Gathering with Scheduled Bundle.

2) Statistics Gathering with Scheduled Bundle

Our method gathers transfer statistics from switches at the same time periodically. In general, when statistic gathering, a controller sends STATS_REQUEST messages periodically, and its timing depends on the controller performance. On the other hand, our method uses STATS_REQUEST wrapped by Scheduled Bundle, since it gathers statistics between each switch simultaneously. It can gather statistics without depending on controller performance.

Fig. 3 illustrates a flow of gathering transfer statistics at timestamp *Ts* with Scheduled Bundle. First, the controller sends BUNDLE_OPEN message to the switch, followed by BUNDLE_ADD message which encapsulating STATS_REQUEST message to gather statistics for all flow entries. Finally, the controller sends BUNDLE_COMMIT message with the timestamp of schedule execution timing *Ts*.

To verify byte consistency, our method mainly uses *byte_cnt* and match field information that contained in STATS_REPLY messages. Our method associates flow and statistics according to match field, and calculate *byte_cnt* difference from already collected STATS_REPLY information.

The difference of statistics between the switches may occur by the timing of FLOW_MOD message send by the controller application depends on the mechanism of routing control. Additionally, the statistics of simultaneity gathering also depends on the performance of the switches, such as the size of flow table and the accuracy of schedule execution. Thus, our method uses moving averages of the difference last four statistics report (i.e. use *byte_cnt* difference) at the same time between each switch, called *ByteDiff*. Since this interval is sufficient to eliminate the effects of scheduling errors and traffic bursts, our mechanism can avoid false alarms. In comparison with SPHINX's Σ, since the scheduling mechanism gathers the statistics simultaneously, *ByteDiff* of our method does not depend on the controller performance.

3) Algorithm

Algorithm 1 describes the step of executing the consistency check by a given flow graph and gathered statistics between switches simultaneously. The algorithm needs the flow graph as input, and the flow graph includes a current path relevant to traffic flow *F*. Our method verifies whether compromised switches attack the network from the same two points as SPHINX.

Algorithm 1 Proposal algorithm

```
Input: $F$:traffic flow, $\tau$:threshold
Output: $\mathbb{O}$:violation switches of traffic flow$F$
  function VERIFY($F$,$\tau$)
      Initialize:
          $FG := \text{GET\_FLOWGRAPH}(F)$
          $CurrP := \text{GET\_CURRENTPATH}(FG)$
          $\mathbb{O} := \emptyset$
          $PrevByte := \infty$
      for all $S \in CurrP$ do
          $FE := \text{GET\_FLOWENTRY}(S,F)$
          $ByteDiff := \text{GET\_BYTESTATSDIFFERENCE}(FE)$
          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
          $FE := \text{GET\_FLOWENTRY}(S,F)$
          $ByteDiff := \text{GET\_BYTESTATSDIFFERENCE}(FE)$
          if $ByteDiff \neq 0$ then
              $\mathbb{O} := \mathbb{O} \cup S$
```

Firstly, this algorithm validates the statistics of switches over a current path of traffic flow *F*, in order from the nearest switch from a source host. Since the algorithm uses the statistics gathered at the same time, all the switches that passed through must report the similar value of the statistics between the switches. Thus, even if a compromised switch disguises that the statistics are similar to an honest switch, it can be detected by the honest downstream switch.

The algorithm needs considering the difference of the statistics values which occurs by propagation delay, scheduling error, and the difference of flow table size maintaining by the switches. For that reason, it compares the statistics of neighbor switches based on the statistics of the validation already passed switches, called *PrevByte*, using a threshold *τ*. The algorithm reports a violation if it observes much different from the simultaneous statistics of the neighbor switch over the current path.

Secondly, the algorithm verifies whether the statistics of switches, that are not included in the current path associated with traffic flow *F*, are zero. In this way, it can verify that no traffic has been injected and dropped by the switches that are out of the current path.

The algorithm needs the threshold (*τ*) as an input, which is used as the margin of the statistics value similarity. To consider *ByteDiff* varied with communication situation, the algorithm calculates a maximum/minimum *ByteDiff* with multiplying *PrevByte* by the threshold. Additionally, since the performance of statistics gathering depends on the switch performance, which occurs from the accuracy of schedule execution, the flow table size [12], and the implementation of the switch [15], *τ* needs to be determined in consideration of the switch performance. In this algorithm, if the value of *τ* is too large, false negatives may occur and a genuine alarm may not be outputted. On the other hand, If the value of *τ* is too small, the algorithm can cause false positives. Therefore, the administrator must carefully determine the value of *τ*.

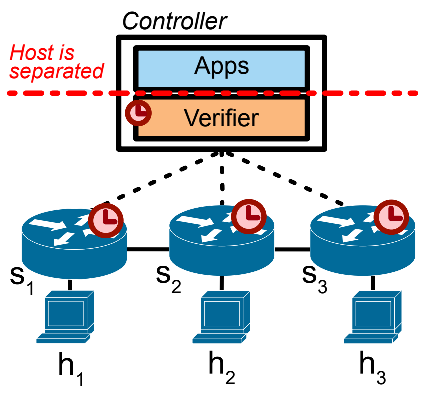


Fig. 4: Microbenchmark topology.

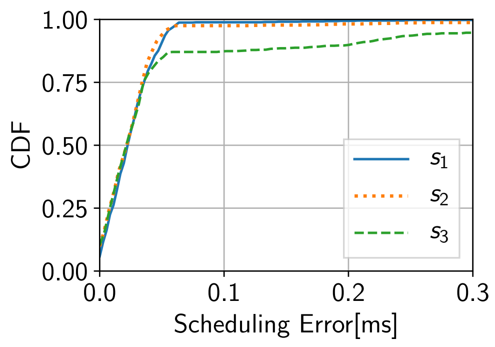


Fig. 5: Scheduling error in our emulated environment.

IV. Evaluation

A. Experiment Setup

1) Implementation

We suppose our method to integrate into an application of the controller. However, since we consider the effect of processing for this experiment, we implemented our method as a proxy between the controller application and the switches, separately from the controller application. We implemented it in Stopcock [16], which is an implementation of OpenFlow proxy, and we used custom Loxigen [17] script to achieve compatibility with Scheduled Bundle. We used the OpenFlow switch called ofsoftswitch13_EXT-340 [18], which compatible with Scheduled Bundle.

We determined that the interval of the statistics gathering is three seconds and the time of executing schedule is after one second at the first BUNDLE_OPEN message sent. Since ofsoftswitch13_EXT-340 is not supported multiple scheduling, these intervals are sufficient value to eliminate the duplicated schedule.

2) Experiment Environment

We evaluated our prototype by micro-benchmarking on an emulated network with Mininet [19]. Our testbed emulates the network on a single machine, and each node refers to the same hardware clock. Thus, we can assume that the time of each emulated node is synchronized. Although time synchronization emulation with perfect precision is impossible due to resource conflict, it is sufficient accuracy for out method to use statistical average to reduce performance attributes. Fig. 4 shows that emulated topology in our testbed by using Mininet (linear, k = 3, n = 1). The verification program placed on the same host as running Mininet to emulate time synchronization. To mitigate the impact of resource conflicting, we used minimal topology and separated the controller application host and the Mininet host including the verification program. We hosted the emulated network and the verification program on a machine equipped with Intel Xeon E3-1220v5, 3.00GHz, quad-core and 32GB RAM. We used Floodlight v1.1 [20] as the SDN controller application, and it hosted on a machine with Intel Core i5, 3.10GHz, quad-core and 8GB RAM.

3) Scheduling Accuracy

We measured the accuracy of scheduling to show the adequacy of using emulated testbed. Fig. 5 shows the results of experiments measuring the accuracy of scheduling in the testbed. In this experiment, we measured the difference between actual execution time and scheduled execution time, by sending 1K Scheduled Bundle from the verification program every three seconds to each switch. The switches also put a load on by generating traffic from h_1 to h_3. The scheduling error depends on the processing power of the machine, and the frequency of resource confliction. Therefore, the scheduling error of s_3 is larger than others. We observed that the scheduling error of our testbed is less than 0.3 millisecond in the 90 percentiles at all switches. From this result, we confirmed that this environment is sufficient to evaluate our method, since our method can mitigate the scheduling error with using moving average.

B. Accuracy of Verification

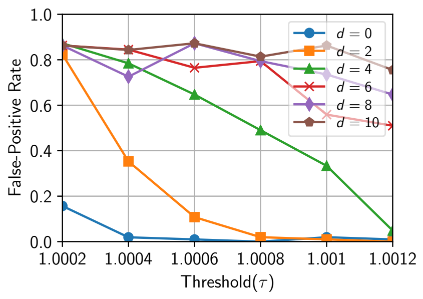
We evaluated the accuracy of the verification in the 3-hop path which uses TCP flows from h_1 to h_3 with iperf. We executed verifications with SPHINX and our method and compared the accuracy of the verification. Since our testbed is a minimal configuration, we simulated the variation of the controller performance by delay (d), which inserts before sending STATS_REQUEST from the verification program. In fact, Tootoonchain et al. [11] reported that as increasing the number of connected switches cause I/O handling overhead and resource contention on the task, the latency of controller response increase.

1) False alarm

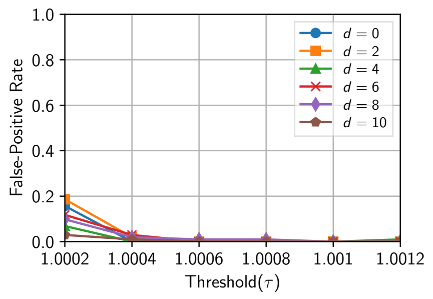
We analyzed the probability of false alarm caused by the impact of the controller performance degradation. In this experiment, it is not preferable to raise an alarm in verification because all switches are legitimate.
Fig. 6a and Fig. 6b illustrate false-positive rate in SPHINX and our method. We observed that SPHINX increases false alarms evidently when the controller performance occurs degradation (i.e. d become increasing), as it depends on the controller performance. In contrast, we observed that even if d increase, the false positive rate of our method is without increasing, it similar to SPHINX at $d = 0$.

2) Lack of genuine alarm

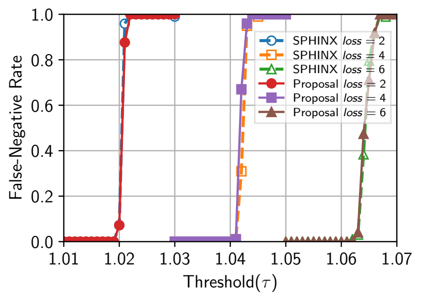
We evaluated the probability of the lack of genuine alarm given the variety of the controller performance using the same



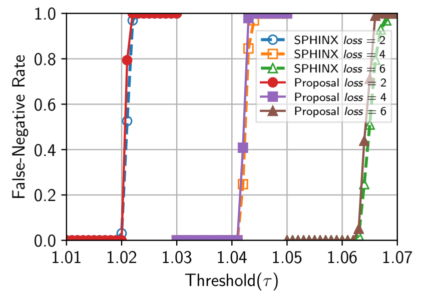
(a) SPHINX's false-positive rate with variation in τ and d.



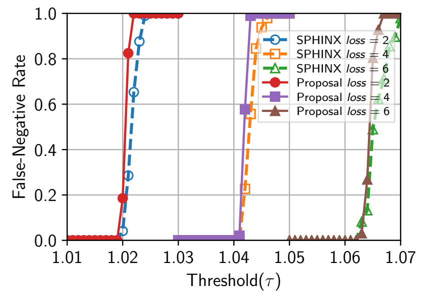
(b) Our method's false-positive rate with variation in τ and d.



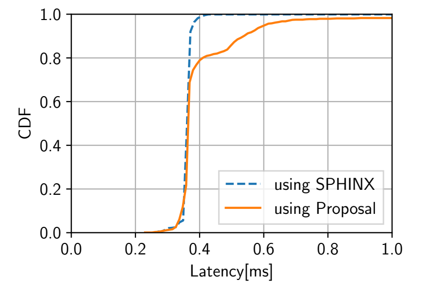
(c) Comparison of false-negative rate at d = 0 vs τ and link loss rate.



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



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



(f) Comparison of our method's and SPHINX's ping latencies.

Fig. 6: Experiment results.

conditions, and compared each method. We performed packet drop on a link of between s2 to s3 by given link loss rates, such as 2%, 4%, and 6%, to emulate malicious behaviors by compromised switch or link. In this experiment, although it is preferable that alarms are generated in all verifications, false negatives may occur when τ increases.

Fig. 6c through Fig. 6e illustrate false-negative rates of each method at the three kinds of controller performance value d (d = 0, 5, 10). We observed that the false negative rate of our method is slightly higher than SPHINX when d increases. However, our method can set small τ than SPHINX (see Section IV-B1). Thus, our method can achieve the performance of the false-negative rate equivalent to SPHINX by tuning τ.

C. *Overheads*

We measured the overhead of statistics gathering from switches with SPHINX and our method. We measured the ping latencies between h1 to h3 while gather statistics every three seconds. Fig. 6f shows the result of this experiment. We observed that the ping latencies of our method are partly higher than SPHINX, since the implementation of Scheduled Bundle by ofsoftswitch13_EXT-340 affects packet processing.

V. Discussion

A. *Limitations*

Our method has a few following limitations that similar to SPHINX.

- Our method cannot identify ingress or egress switch is compromised or not since it depends on STATS_REPLY messages from untrusted switches. Thus, if the edge switch is compromised, our method cannot detect the attack even if the switch that passes through after is legitimate. Therefore, our method cannot apply to End-to-End verification currently. However, it is possible to detect attacks from compromised switches by managing all the hosts by the verifier as well as the switches, and enabling the gathering of statistics from the hosts at the scheduled time.
- Our method may miss some transient attack since the span of verification depends on the verification program. To fix this limitation, it is necessary to shorten the statistics gathering interval or change to a highly accurate network monitoring method such as WedgeTail.
- Our method cannot verify traffic integrity. To overcome the issue, cryptographic mechanisms can support it.

B. *Future Work*

We need further experiments which evaluate the scalability, the overhead and the accuracy in a real-world environment which is synchronized time accurately. Additionally, we plan to reduce the statistics gathering overhead of Scheduled Bundle by periodic scheduling, since our current prototype sends Scheduled Bundle every timing of statistics gathering.

Additionally, the compatibility of our method with distributed controller environment such as ONOS [21] is required further investigation. Even if these sites manage separately, it is possible to synchronize the time of all the switches and the controllers with sufficiently high precision by utilizing time synchronization method such as PTP. Thus, we believe that improved our method is suitable for distributed controller environment. However, since distributed controller environment may have link delays caused by the distances between sites, we plan to improve the method to consider a link specific delay.

VI. Conclusion

Software Defined Network (SDN) is attracting rising attention as a future networking paradigm. However, although SDN security such as control plane well studied, the attacks on data plane by compromised switches can be a more serious threat. Unfortunately, existing solutions have some issues relevant to scalability such as increasing false alarm rely on controller performance. This paper showed that the first step to a countermeasure against compromised switches by utilizing to gather statistics with the timing schedule. By gathering statistics simultaneously from the switches by scheduling, our method can detect attacks by compromised switches without depending on the controller performance. Additionally, our method considered that the difference of statistics which occurs from scheduling error. From the results of the experiments, we confirmed that the false positive rate of our method is lower than SPHINX even if the controller performance decrease.

References

[1] D. Kreutz, F. M. Ramos, and P. Verissimo, “Towards secure and dependable software-defined networks,” in *Proc. HotSDN*. ACM, 2013.

[2] M. Dhawan, R. Poddar, M. Kshiteej, and M. Vijay, "SPHINX: detecting security attacks in software-defined networks," in *Proc. NDSS*. Internet Society, 2015.

[3] A. Shaghaghi, M. A. Kaafar, and S. Jha, "Wedgetail: An intrusion prevention system for the data plane of software defined networks," in *Proc. AsiaCCS*. ACM, 2017.

[4] "CVE-2016-2074," accessed: May 13, 2018. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE- 2016- 2074/

[5] T. Mizrahi and Y. Moses, "Time4: Time for SDN," IEEE Transactions on Network and Service Management, vol. 13, no. 3, pp. 433-446, 2016.

[6] 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 Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[7] Open Networking Foundation, "OpenFlow Switch Specification Version 1.5.0," 2014, accessed: May 13, 2018. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow- switch- v1.5.1.pdf

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

[9] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying Network-Wide Invariants in Real Time," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, vol. 42, no. 4, sep 2013, p. 467.

[10] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real Time Network Policy Checking Using Header Space Analysis," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI'13)*, 2013, pp. 99-112.

[11] A.Tootoonchian,S.Gorbunov,Y.Ganjali,M.Casado,andR.Sherwood, "On controller performance in software-defined networks," in *Proc. HotICE*. USENIX, 2012.

[12] 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. SIGCOMM*. ACM, 2011.

[13] S. Khan, A. Gani, A. W. Abdul Wahab, M. Guizani, and M. K. Khan, "Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art," IEEE Communications Surveys Tutorials, vol. 19, no. 1, pp. 303-324, 2017.

[14] T. Mizrahi, "Time synchronization security using IPsec and MACsec," in *Proc. ISPCS*, no. Icv, 2011, pp. 38-43.

[15] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An Open Framework for OpenFlow Switch Evaluation," in *Lecture Notes in Computer Science*, 2012, vol. 7192, pp. 85-95.

[16] P. Wood, "Stopcock," 2014, accessed: May 13, 2018. [Online]. Available: https://github.com/tignetworking/stopcock/

[17] Project Floodlight, "Loxigen," 2018, accessed: May 13, 2018. [Online]. Available: https://github.com/floodlight/loxigen/

[18] TimedSDN, "ofsoftswitch13_EXT-340," 2015, accessed: May 13, 2018. [Online]. Available: https://github.com/TimedSDN/ofsoftswitch13_EXT- 340/

[19] Mininet, "Mininet," 2018, accessed: May 13, 2018. [Online]. Available: http://mininet.org/

[20] Project Floodlight, "Floodlight," 2018, accessed: May 13, 2018. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[21] 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. HotSDN. ACM, 2014.

Takahiro Shimizu received his B.E. degree in computer and information science from Tokyo University of Agriculture and Technology, in 2018. Since April 2018, he has been a graduate student in the Graduate School of Engineering, Tokyo University of Agriculture and Technology. His research interests include Software Defined Network (SDN) security and monitoring.

Naoya Kitagawa received his B.Sc. and M.Sc. degree in information science from Chukyo University, Toyota, Japan, in 2009 and 2011 respectively, and his Ph.D. degree in information science from Nagoya University, Nagoya, Japan, in 2014.

In April 2014, he joined Information Technology Center, Nagoya University as a postdoctoral fellow. Since October 2014, he has been an assistant professor in the Institute of Engineering, Tokyo University of Agriculture and Technology. His research interests include the Internet, network security, and distributed system. He is a member of IPSJ.

Kohta Ohshima received his B.E. and M.E. degrees in electronics and information engineering and his Ph.D. degree in 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 an senior lecturer in the Saitama University of Technology, and he was an associate professor in the same department from 2016 to 2016. Since 2016, he has been a associate professor in 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 his B.E. and M.E. degrees in electronic engineering and his 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. From April 1990 to March 1994, he was an Assistant Professor in the same department. 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 in the Information Technology Center (at present, the Center for Information Technology and Management), Okayama University. Since April 2014, he has been a professor in the Institute of Engineering, Tokyo University of Agriculture and Technology. His research interests include distributed system, network architecture and Internet. He is a member of IEICE, IPSJ and IEEE.