Byte Consistency Verification Method with Dynamic Threshold Adjustment for Each Node in Software-Defined Networking

Naoya Kitagawa
Research and Development Center for Academic Networks
National Institute of Informatics
Tokyo, Japan
kitagawa@nii.ac.jp

Jumpei Sato
Graduate School of Marine Science and Technology
Tokyo University of Marine Science and Technology
Tokyo, Japan
m234020@edu.kaiyodai.ac.jp

Kohta Ohshima
Marine Electromechanical Engineering Division
Tokyo University of Marine Science and Technology
Tokyo, Japan
kxoh@kaiyodai.ac.jp

***Abstract*—Software-defined networking (SDN), which enables flexible routing control based on communication content, has been widely studied as a countermeasure against possible attacks on the data plane by compromised SDN switches and hosts. We previously proposed a byte consistency verification method that uses information such as transfer volume collected from SDN switches to detect anomalous communications, even when the communications are encrypted. In addition, we improved the anomaly detection performance of this method by implementing high-precision time synchronization and an SDN switch function for each host. In this study, we extend the scope of information collection to each host (in addition to SDN switches) and propose a data plane anomaly detection method that monitors the communication volume of each process at each host. We also propose a method that automatically adjusts the threshold, which can be set individually for each node, used for detection. Furthermore, we implement and evaluate the proposed method on a network testbed. The results confirm that it can be used to improve anomaly detection accuracy.**

Index Terms*—Software-defined networking; Data plane verification; Byte consistency verification; Anomaly detection.

I. Introduction

This paper is an extended version of our study presented at the Twentieth International Conference on Networking and Services (ICNS 2024) [1]. Network equipment that uses software-defined networking (SDN) and network functions virtualization (NFV) technology has recently been introduced into carrier and data center networks; further widespread use is expected [2]. Unlike conventional router devices, which have fixed settings, SDN enables flexible routing control using various types of information, such as the content of transmitted data, information on sending and receiving terminals, and networks passed through. The SDN switches that make up an SDN network cooperate according to control information from the SDN controller, enabling fine control of communication on a flow-by-flow basis.

In the operation of SDN, it is important to ensure compatibility with security-related technologies. Encrypted communication is becoming a mainstream measure against information leaks, with Google reporting that 95% of its total communication traffic was encrypted as of November 2023 [3]. While encrypted communication can ensure end-to-end security, it makes it difficult for network operators to use intrusion detection and prevention systems, which provide security by checking the payload of exchanged packets. In addition, network administrators must also be aware of countermeasures against SDN switches and silent failures. SDN networks are often realized using software switches, making them possibly more vulnerable than networks consisting of conventional hardware switches [4] [5] [6] [7]. Specifically, SDN controllers may not be able to detect SDN switches that are compromised or defective. To solve these security issues, byte integrity verification has been proposed, where anomalies are detected by collecting and processing communication status data from a group of SDN switches.

We previously proposed a method for increasing the granularity of anomalies that can be detected in SDN communications by using high-precision time synchronization via IEEE1588 PTPv2 [8] to ensure the time resolution of collected communication status data and by handling transfer volume information in units of flows [9]. Furthermore, to solve the problem of conventional byte consistency verification, where the accuracy of information collected from a terminal SDN switch cannot be verified, we developed a method for expanding the range of devices that can detect anomalies by incorporating a reporting function similar to that of SDN switches in the host connected to a terminal SDN switch [10]. The results of our previous research indicate that the quality and variety of data that can be collected from SDN switches and hosts are useful for improving the anomaly detection

performance of SDN.

In this paper, we confirm the applicability of byte consistency verification for anomaly detection in SDN networks by collecting communication status data for each host. In our approach, statistical data on the communication status, which can be obtained using commands provided by the host operating system (Linux), are formatted to be compatible with SDN networks. They can be used by SDN controllers and nodes that perform byte integrity verification. We implement this method on a network testbed to obtain per-process communication volume information measured at each host and confirm that it is applicable to anomaly detection in SDN networks.

In addition, a conventional anomaly detection method (see Section II) depends on the knowledge and experience of the administrator because the threshold values need to be set manually. Since conventional methods use the same threshold value for the entire network, they cannot detect minute anomalies or identify the location of anomalies. To address these issues, we propose and implement a method for detecting anomalous switches that automatically sets the threshold value for each node individually. The improvement in anomaly detection accuracy is determined through experiments using a testbed.

The rest of this paper organized as follows. Section II describes related techniques and existing research. Section III explains the proposed network verification scheme, which deals with the process-level communication volume of hosts. Section IV describes the dynamic adjustment of thresholds. Section V describes an experiment in which the proposed method was implemented on a testbed. Section VI discusses considerations based on the results of evaluation experiments. Finally, Section VII presents the concluding remarks.

II. Related Work

In this section, we review related technology and existing research.

A. Software-Defined Networking

SDN allows network devices to be centrally controlled through software. In a conventional network, shown in Figure 1, the network administrator configures each router for routing control. The router forwards packets according to its configuration. In contrast, in an SDN network, forwarding control instructions can be issued to all SDN switches by configuring the SDN controller. The SDN switches perform packet forwarding based on these instructions. Therefore, SDN allows dynamic control based on the operation status of each SDN switch. Flexible control in SDN is achieved by separating the data plane, which handles data forwarding functions, and the control plane, which handles control functions [11].

OpenFlow [12] is widely used for implementing SDN. There are several OpenFlow controller implementations, such as Floodlight [13]. Although SDN allows for flexible control of the network, several security issues have been reported [4] [14]. For example, there are known attacks in which malicious switches attack the data plane or mislead the SDN controller about the network topology. Methods have been developed to solve these problems [5] [6].

B. Data Plane Security in SDN

If an SDN network is compromised, unintended packets may be discarded or generated and routes may be changed. To prevent such problems, verification techniques can be used to protect the data plane. SPHINX verifies compromised switches using byte consistency verification [5]. This method detects anomalies by having each switch collect and compare transfer volume information.

Figure 2 shows the operation of byte consistency verification by SPHINX. A report of the forwarding volume information from each switch is received. Based on the received information, the method calculates a moving average ($\sum$) of the transfer volume information for each SDN switch and a value ($\sum_{avg}$) obtained by averaging the moving average for each SDN switch over all SDN switches. Then, the method checks whether the average value deviates from the moving average value of each SDN switch by the inequality in Equation 1 using a predetermined value of threshold τ.

$$\frac{1}{\tau} < \frac{\sum}{\sum_{avg}} < \tau \tag{1}$$

If threshold τ is excessively small, false positives (FPs) are likely to occur; if it is excessively large, false negatives (FNs) are likely to occur. The appropriate value of τ depends on the network configuration and type of switches used. It is thus necessary to set an appropriate value for each network. Since SPHINX performs byte integrity verification using network switch forwarding volume information, it cannot verify whether an edge switch is malicious and it does not support flow aggregation. Various other SDN data plane security measures have been proposed [14].

C. WhiteRabbit

As described in Section II-B, for SPHINX, detection accuracy is affected by variations in the timing of obtaining statistic information from switches. To address this issue, WhiteRabbit reduces the deterioration of verification accuracy due to acquisition timing deviations by using IEEE1588 PTPv2 for high-precision time synchronization and scheduling the timing of the acquisition of transfer volume information [9]. However, WhiteRabbit, like SPHINX, does not verify edge switches and does not support flow aggregation.

D. Edge Switch Validation with In-host Switches

As mentioned in Sections II-B and II-C, byte consistency verification using only SDN switch information cannot verify edge switches. To solve this problem, we previously proposed a method for obtaining the communication volume of each host [10]. This method builds a switch inside the host to obtain the host's communication volume and behaves like any other SDN switch, allowing byte integrity verification between the edge switch and the host. However, the method requires the threshold τ in Equation 1 to be larger than that for the

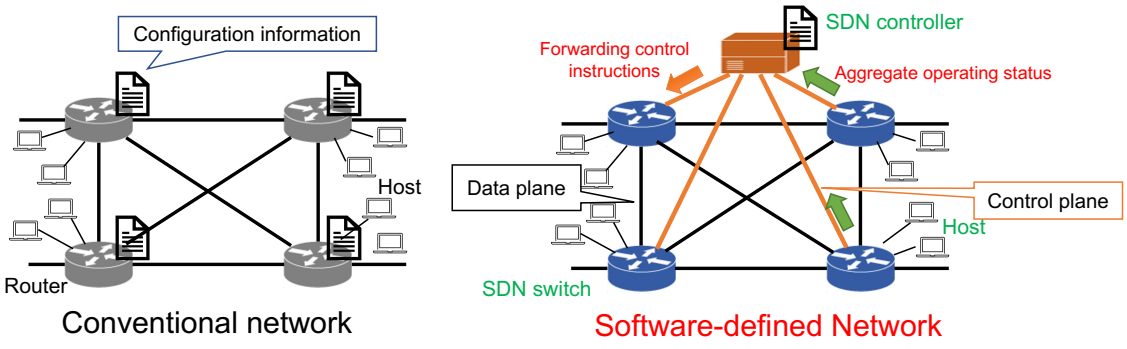

Fig. 1. Comparison of conventional network and software-defined network.

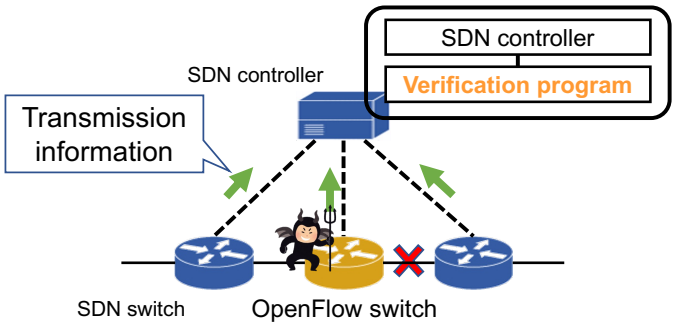

Fig. 2. Byte consistency verification by SPHINX.

conventional method, which may make it miss minute network anomalies or attacks that take place in a very small amount of time.

III. Proposed Method

To overcome the issues described in Section II, in this section, we describe a network verification scheme that deals with the process-level communication volume of hosts. Figure 3 shows an overview of the proposed method. As shown, host information is collected by implementing an in-host information collection function on hosts in a conventional SDN network. In addition, we deploy a host information collection server to compare the SDN controller's collection of each SDN switch's forwarding volume information. This allows the verification system to perform host-information-aware verification. This system improves the accuracy of detecting abnormal networks by classifying communication volume using detailed host information, which cannot be obtained using the conventional method.

This system requires the implementation of the following two functions.

1) A function for each host to send its collected data (process-level traffic information) to the host information collection server.
2) A function for the SDN controller to send the traffic information of each switch to the host information collection server.

In addition, the host information collection server needs to know which host sent the data and compare the data with the transfer volume information of each switch. Furthermore, each host needs to implement a function to collect its own process-level traffic and send the collected information to the server.

A. Host Information Collection Server

The host information collection server monitors the traffic information of all hosts that have executed the intra-host information transmission agent and alerts the user according to the conditions based on the statistics of the traffic information. The host information collection server collects per-process communication volume information from each host, compares it with the transfer volume information of each switch collected by the SDN controller, and sends an alert to the network administrator if any abnormality is found.

B. Host Information Collection Agent

The host information collection agent, which is implemented on each host, executes the *ss* (socket statistics) command provided by the Linux operating system as an external command to obtain the cumulative number of received packets as statistical information for each process. Then, the agent sends the acquired information to the host information collection server. By repeating these processes periodically, the host information collection agent collects transfer volume information for each host.

IV. Automatic Adjustment of Threshold

In this section, we describe a method that automatically sets threshold τ for byte consistency verification and allows the threshold to be fine-tuned for each node, thereby improving the granularity of anomaly detection. As described in Section III, in our scheme, the SDN controller collects transfer volume information from the host switches and from each SDN switch. The anomaly detection system uses this information to detect anomalies. Conventional anomaly detection methods such as

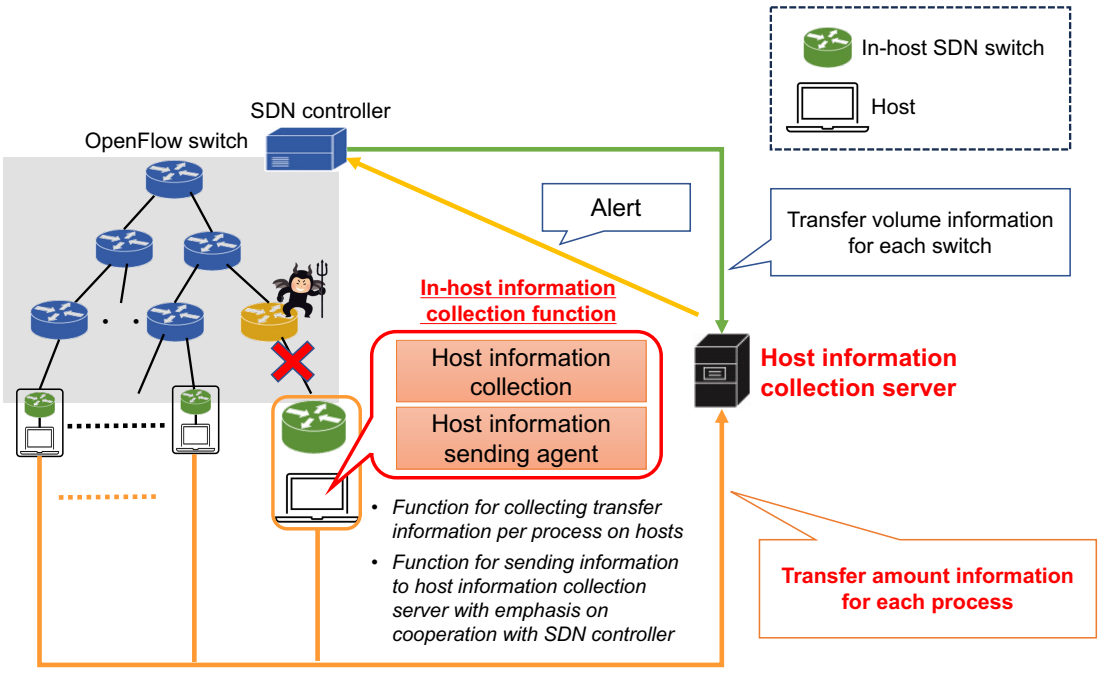

Fig. 3. Overview of proposed method.

SPHINX require manual setting of threshold values used for detection at the discretion of the administrator. In contrast, we automate the setting of threshold values used for detection so that they can be adjusted without relying on the knowledge and experience of the network administrator. This also allows the setting of individual thresholds for nodes, which is not possible with the conventional method. Setting an appropriate threshold for each node enables the detection of minute anomalies and the identification of switches with anomalies that are missed with the conventional method.

A. Calculation Method

The first step in the thresholding calculation is to determine the reference threshold value, as shown in Equation 2. Figure 4 shows an example of switch placement. As shown, when traffic flows from left to right, the switch closest to the origin is defined as $FormerSwitch$ (FSW) and the switch closest to the end is denoted as $LatterSwitch$ (LSW) .

$$Reference\ Threshold\ = \frac{FSW's\ transfer\ volume}{LSW's\ transfer\ volume} \quad (2)$$

Then, as shown in Figure 5, the upper and lower threshold limits (tolerance rate) are set and the reference threshold is given as the range $\pm\ x\ \%$. By giving the threshold as a range, the sensitivity of anomaly detection can be adjusted and FPs can be prevented.

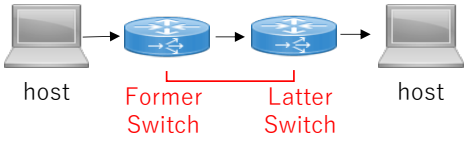

Fig. 4. Example of switch placement.

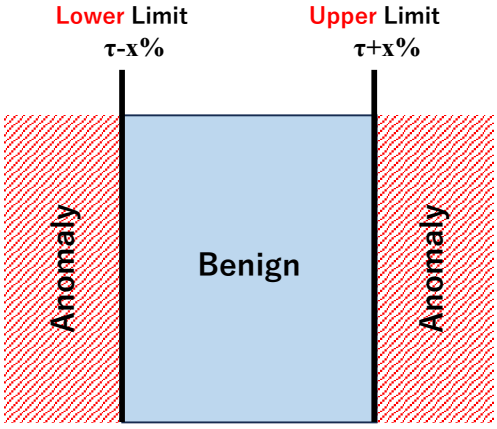

Fig. 5. Threshold tolerance.

B. Individual Threshold Setting for Nodes

Figure 6 shows the method used to set individual threshold values for nodes. Using the calculation method described in Section 3.2, threshold values $\tau 1$–$\tau 4$ are set between nodes SW1 and SW2, SW2 and SW3, SW3 and SW4, and SW4 and SW5, respectively. By setting individual threshold values in this way, it is possible to identify which threshold value was used to detect an anomaly and thus the switch to which

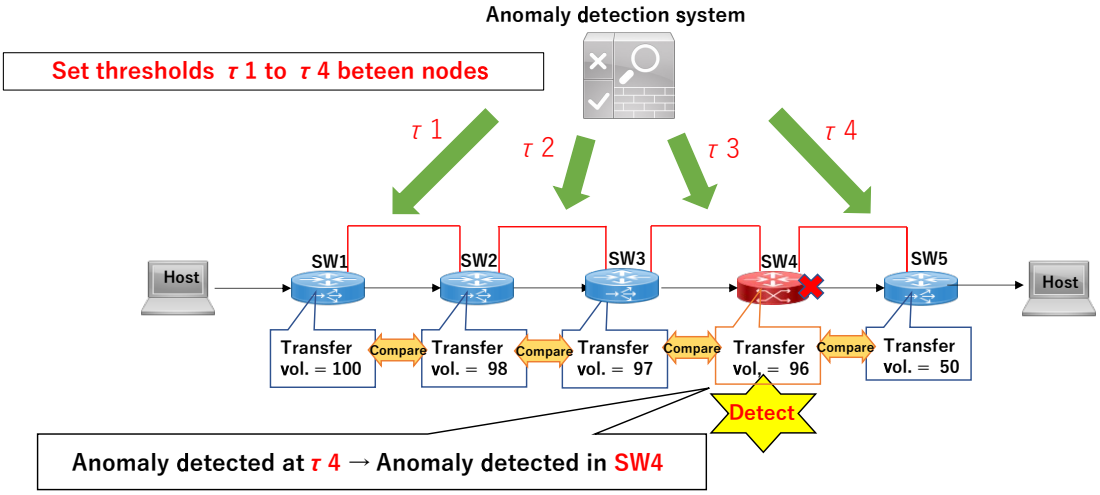

Fig. 6. Individual threshold setting and anomaly detection for nodes.

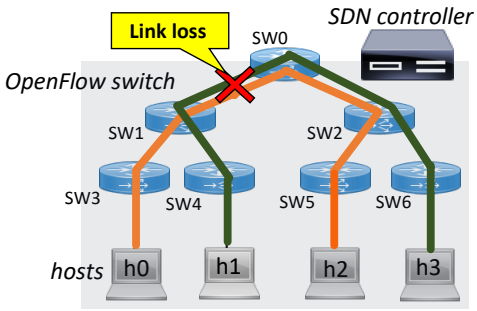

Fig. 7. Network configuration used in evaluation experiment.

TABLE I
SPECIFICATIONS OF MICROCLOUD NODES IN DETERLAB USED IN EXPERIMENT.

Type	Specifications
CPU	Intel(R) Xeon(R) E3-1260L Quad-Core Processor Running at 2.4 GHz
Memory	16 GB
Storage	250 GB SATA Western Digital RE4 Disk Drive
OS	Ubuntu 16.04 LST

that threshold value was assigned can be identified as the anomalous switch. In the example shown in Figure 6, the anomaly is detected at threshold value $\tau 4$, which means that SW4 is anomalous.

V. EXPERIMENT

A. Environment

To verify and evaluate the operation of the host information collection function based on this method, we implemented an experimental network on DeterLab, a network testbed operated by the University of Southern California Information Sciences Institute and the University of Utah [15].

We used a total of 12 nodes on DeterLab, each with an SDN controller, a verification component, SDN switches (7 nodes), and hosts (4 nodes). As shown in Figure 7, the network for this experiment had a tree network topology with $Depth = 2$ and $Fanout = 2$, where $Depth$ indicates the depth of the hierarchy from the root node and $Fanout$ indicates how many nodes are connected in one branch.

Table I shows the specifications of the MicroCloud on DeterLab used in this experiment. All 12 nodes in this experiment used equipment with the same specifications. We used Floodlight v1.2 [18] as an SDN controller. We also implemented an OpenFlow proxy, stopcock, between the group of switches and the controller as the verification component for route verification. ofsoftswitch13_EXT340 [19] was used as the SDN switches. Since this experimental network consisted of actual equipment rather than simulators or emulators, the evaluation environment was close to that in actual operation.

B. Evaluation of Detection Accuracy

In this section, we describe an experiment conducted to determine the anomaly detection accuracy for the proposed method and the conventional method SPHINX. In this evaluation experiment, we investigated the effectiveness of the thresholds automatically set by the proposed method and the effect of partial threshold setting on anomaly detection accuracy.

We used the FN rate for malicious traffic and the FP rate for benign traffic as evaluation metrics. We measured TCP communications over a five-hop path using iperf and compared the anomaly detection accuracy of the proposed method with that of SPHINX based on the amount of forwarded data sent from each switch to the SDN controller.

1) False Negative Rate: We set up a link with a loss rate between SW0 and SW1 and discarded packets on this link, as

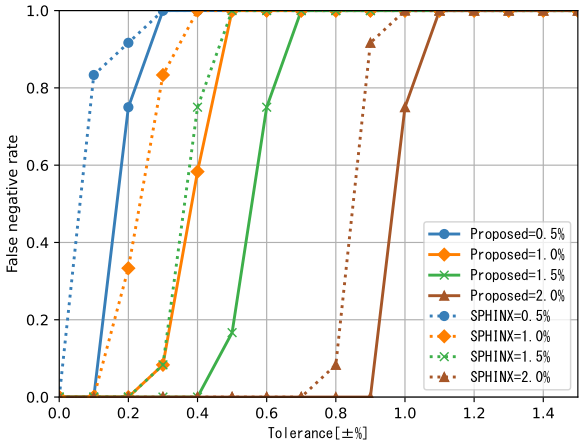

Fig. 8. FN rate comparison between SPHINX and proposed method.

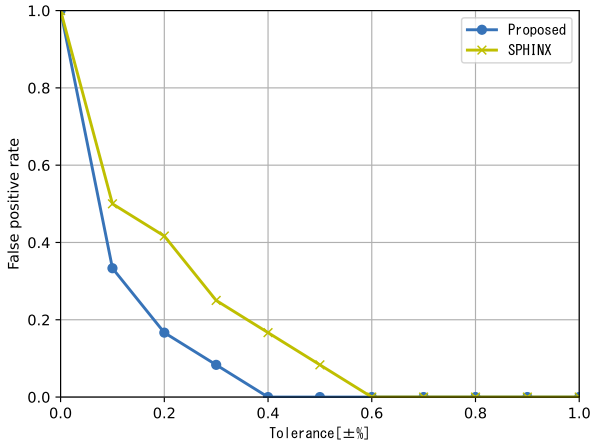

Fig. 9. FP rate comparison between SPHINX and proposed method.

shown in Figure 7. The link loss rates in this experiment were 0.5%, 1.0%, 1.5%, and 2.0%.

Figure 8 shows a comparison of the FN rate between the proposed method and SPHINX. The vertical axis indicates the FN rate and the horizontal axis indicates the tolerance rate x %, which is a $\pm$ x % variation of the reference threshold obtained in Equation 2 in Section IV-A. An FN rate that is sufficiently low in the range where the tolerance is greater than 0% indicates that there is no detection failure (i.e., that anomalies were detected). For SPHINX, the FN rate increased rapidly when the link loss rate was 0.5%, with the tolerance rate increasing from 0%. This indicates that SPHINX was unable to detect an anomaly when the link loss rate was 0.5%. In contrast, for the proposed method, when the link loss rate was 0.5%, the FN rate remained at 0 up to a tolerance rate of $\pm$ 0.1%, confirming that our method could detect anomalies. The results for a link loss rate of 2.0% indicate that the tolerance rate x at which the FN rates for SPHINX and the proposed method begin to increase is 0.7% and 0.9%, respectively.

2) False Positive Rate: We measured the FP rate after generating traffic flows, as done in the evaluation of the FN rate. Figure 9 compares the FP rates for SPHINX and the proposed method. As shown, the tolerance rate at which the FP rate becomes zero is 0.6% for SPHINX and 0.4% for the proposed method.

C. Effect of Threshold Calculation Method

To evaluate the effectiveness of the threshold calculation method presented in Section IV-A, we evaluated the FN and FP rates using another threshold calculation method that automatically sets the threshold. The calculation method is shown in Equation 3.

$$Reference\,Threshold = \frac{Ingress\ SW's\ transfer\ volume}{Egress\ SW's\ transfer\ volume} \quad (3)$$

1) False Negative Rate: As done above, the link loss rate was set to 0.5%, 1.0%, 1.5%, and 2.0%. Figure 10 shows a comparison of the FN rate between the proposed threshold calculation method, shown in Equation 2, and the calculation method in Equation 3. For a link loss rate of 0.5%, minute anomalies were detected only when the proposed method was used. It can also be seen that the proposed method has a larger tolerance rate x at which the FN rate begins to increase when the link loss rate is 2.0%.

2) False Positive Rate: The FP rate was measured after traffic flows were generated for the evaluation experiment. Figure 11 shows a comparison of the FP rate between the proposed threshold calculation method, shown in Equation 2, and the calculation method in Equation 3. The acceptable rate at which the FP rate becomes zero is 0.4% for the proposed method and 0.8% for the calculation method in Equation 3.

VI. Considerations and Discussion

A. Comparison of SPHINX and Proposed Method

When there is a link with a loss rate, the FN rate increases rapidly for SPHINX, with the tolerance rate increasing from 0% when the link loss rate is 0.5%. This result indicates that SPHINX is unable to detect an anomaly when the link loss rate is 0.5%. In contrast, for the proposed method, when the link loss rate is 0.5%, the FN rate remains at 0 up to a tolerance rate of $\pm$ 0.1%. This indicates that the proposed method can detect minute anomalies that SPHINX cannot.

In the experiment with a link loss rate of 2.0%, the tolerance rate x at which the FN rates for SPHINX and the proposed method begin to increase was 0.7% and 0.9%, respectively. This confirms that the proposed threshold setting method is superior to that of SPHINX.

For benign traffic, the tolerance rate at which the FP rate becomes 0 is 0.6% for SPHINX and 0.4% for the proposed

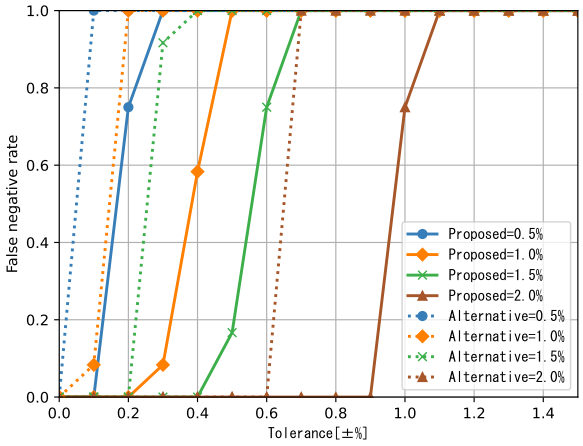

Fig. 10. FN rate comparison between proposed threshold calculation method and alternative method in Equation 3.

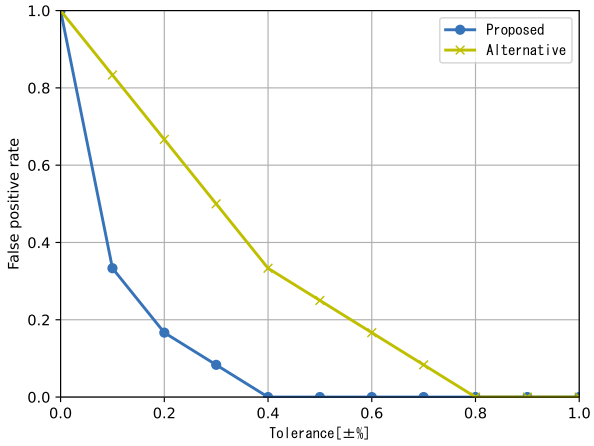

Fig. 11. FP rate comparison between proposed threshold calculation method and alternative method in Equation 3.

method. The FP rate is thus not considered to be significantly different between SPHINX and the proposed method. As can be seen in Figures 8 and 9, the acceptable rate of no false detection and no missed detection for the proposed method is 0.4% to 0.9%. Link loss rates of 1.5% and 2.0% within this range can be correctly detected without false detection or missed detection.

These results show that there is a trade-off between the FN rate and the FP rate. The improved FN rate for the proposed method despite similar FP rates between SPHINX and the proposed method can be attributed to the improvement in anomaly detection accuracy by the automatic setting of individual thresholds.

B. Discussion of Threshold Calculation Methods

As described in Section V-C, to evaluate the validity of the threshold calculation scheme of the proposed method, the FN and FP rates obtained for an alternative calculation method were compared. Regarding the FN rate, it was confirmed that the alternative calculation method was unable to detect small link loss rates (i.e., small anomalies). In addition, the threshold tolerance X was larger for the proposed method for all link loss rates. Regarding the FP rate, the proposed method had an FP rate of 0.4% and the alternative method had an FP rate of 0.8%. These results confirm the validity of the proposed threshold calculation method.

VII. Conclusion

This study proposed a method for collecting forwarding volume information for each host in an SDN network to improve network verification accuracy.

The threshold values used for detection are automatically set so that they can be adjusted without relying on the knowledge and experience of the network administrator. This allows the setting of individual thresholds for nodes, which is not possible with the conventional method. Setting an appropriate threshold for each node enables the detection of minute anomalies and the identification of switches with anomalies that are missed with the conventional method.

We evaluated the effectiveness of the automatically set thresholds and the impact of the threshold range. The results confirm that the proposed method improves the FN rate and maintains the FP rate compared to those for SPHINX. The improvement in the FN rate and maintenance of the FP rate confirm the effectiveness of the proposed threshold calculation method and demonstrate that applying individual thresholds improves anomaly detection accuracy.

Acknowledgement

We would like to thank the team at the University of South California Information Sciences Institute and the University of Utah, which operated the DeterLab project, for providing the network testbed for this research.

A part of this work was supported by JSPS KAKENHI Grant Number JP19K20252 and JP24K14925.

References

[1] N. Kitagawa, N. Moriyama, and K. Ohshima, “Anomaly Detection by Monitoring Communication Volume at the Process Level of Each Host in SDN,” Proc. of The Twentieth International Conference on Networking and Services (ICNS 2024), pp. 1-6, 2024.

[2] R. Souza, K. Dias and S. Fernandes, “NFV Data Centers: A Systematic Review,” IEEE Access, vol. 8, pp. 51713-51735, 2020.

[3] “HTTPS encryption on the web,” https://transparencyreport.google.com/https/overview?lang=en&hl=en (Accessed: Nov. 10, 2024).

[4] D. Kreutz, F. M. V. Ramos, and P. Verissimo, “Towards secure and dependable software-defined networks,” Proc. of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, pp. 55-60, 2013.

[5] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, “SPHINX: Detecting Security Attacks in Software- Defined Networks,” 2015, doi: 10.14722/ndss.2015.23064.

[6] A. Shaghaghi, M. A. Kaafar, and S. Jha, “WedgeTail: An intrusion prevention system for the data plane of software defined networks,” Proc. of the 2017 ACM Asia Conference on Computer and Communications Security, pp. 849-861, 2017.

[7] A. Feldmann, P. Heyder, M. Kreutzer, S. Schmid, J. Seifert, H. Shulman, et al., "NetCo: Reliable Routing with Unreliable Routers," Proc. of 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 128-135, 2016.
[8] IEEE, 1588-2008, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", pp. 1-300, 2008.
[9] T. Shimizu, N. Kitagawa, K. Ohshima and N. Yamai, "WhiteRabbit: Scalable Software-Defined Network Data-Plane Verification Method Through Time Scheduling," IEEE Access, vol. 7, pp. 97296-97306, 2019.
[10] T. Amano, T. Shimizu, N. Kitagawa, and K. Ohshima, "SDN Data-Plane Verification Method using End-to-End Traffic Statistics," IEICE Tech. Rep., vol. 120, no. 413, NS2020-163, pp. 238-243, 2021 (in Japanese).
[11] K. Benzekki, A. Fergougui, and A. Elalaoui, "Software- defined networking (SDN): a survey," Security and communication networks 9.18, pp. 5803-5833, 2016.
[12] N. McKeown, T. Anderson, H.i Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM computer communication review 38.2, pp. 69-74, 2008.
[13] S. Scott-Hayward, S. Natarajan and S. Sezer, "A Survey of Security in Software Defined Networks," IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 623-654, 2016.
[14] C. Black and S. Scott-Hayward, "A Survey on the Verification of Adversarial Data Planes in Software-Defined Networks," Proc. of the 2021 ACM International Workshop on Software Defined Networks & Network Function Virtualization Security, pp. 3-10, 2021.
[15] "DeterLab," https://www.isi.deterlab.net/ (Accessed: Nov. 10, 2024).
[16] "Prometheus," https://prometheus.io (Accessed: Nov. 10, 2024).
[17] "GitHub - prometheus/Prometheus: The Prometheus monitoring system and time series database," https://github.com/prometheus/prometheus (Accessed Nov. 10, 2024).
[18] "Floodlight Controller," https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview (Accessed: Nov. 10, 2024).
[19] "ofsoftswitch13_EXT-340," https://github.com/oronanschel/ofsoftswitch13_EXT-340 (Accessed: Nov. 10, 2024).