Anomaly Detection by Monitoring Communication Volume at the Process Level of Each Host in SDN

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

Naoki Moriyama
Graduate School of Marine Science and Technology
Tokyo University of Marine Science and Technology
Tokyo, Japan
m224018@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 Network (SDN), which enables flexible routing control based on communication contents, has been widely studied as a countermeasure against possible attacks on the data plane by compromised SDN switches and hosts. We have 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 have improved the anomaly detection performance of this method by implementing a 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 by monitoring the communication volume of each process at each host. Furthermore, we implemented and evaluated this method on a network testbed and confirmed that it can be used to improve anomaly detection accuracy.**

***Index Terms*—Software Defined Network, Data-plane Verification, Byte Consistency Verification**

I. Introduction

Recently, network equipment using Software Defined Network (SDN) and Network Functions Virtualization (NFV) technology has been introduced into carrier networks and data center networks, and further widespread use is expected [1]. Compared to conventional router devices that input fixed settings, SDN has the feature of being able to flexibly control routes using various information such as the content of transmitted data, information on sending and receiving terminals, and networks passed through. The SDN switches that make up the 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 mainstream as a measure against information leaks, with Google reporting that 95% of its total communication traffic was encrypted as of November 2023 [2]. While encrypted communication can ensure end-to-end security, it also makes it difficult for network operators to use IDS and IPS, which provide security by checking the payload of exchanged packets. Also, network administrators must also be aware of countermeasures against SDN switches and silent failures. SDN is often realized using software switches, which are suggested to be more vulnerable than networks consisting of traditional hardware-type switches [3] [4] [5] [6]. These papers specifically point to the possibility that SDN controllers may not be able to detect SDN switches that are compromised or behave unintentionally. As a method to solve these security issues, a security measure using byte integrity verification has been proposed, which detects anomalies by collecting and processing communication status data collected from a group of SDN switches.

We have proposed a method to increase the granularity of anomalies that can be detected in SDN communications by using time information synchronized with high precision by IEEE1588 PTPv2 [7] to ensure the time resolution of collected communication status data, and by handling transfer volume information in units of flows [8]. Furthermore, in order to solve the problem of conventional byte consistency verification, where the accuracy of information collected from the terminal SDN switch cannot be verified, we have developed a method to expand the range of devices that can detect anomalies by incorporating a reporting function similar to that of SDN switches in the host connected to the terminal SDN switch [9]. Based on the results of our previous research, we focused on the fact that the quality and variety of data that can be collected from SDN switches and hosts is useful for improving the anomaly detection performance of SDN.

In this paper, we confirmed the applicability of byte consistency verification for anomaly detection in SDN networks by collecting communication status data of each host for the purpose of collecting more detailed communication status data than the conventional method. In our approach, statistical data on communication status, which can be obtained from commands provided by the host OS (Linux), is formatted to be compatible with SDN networks, and can be used

 ISBN: 978-1-68558-141-1

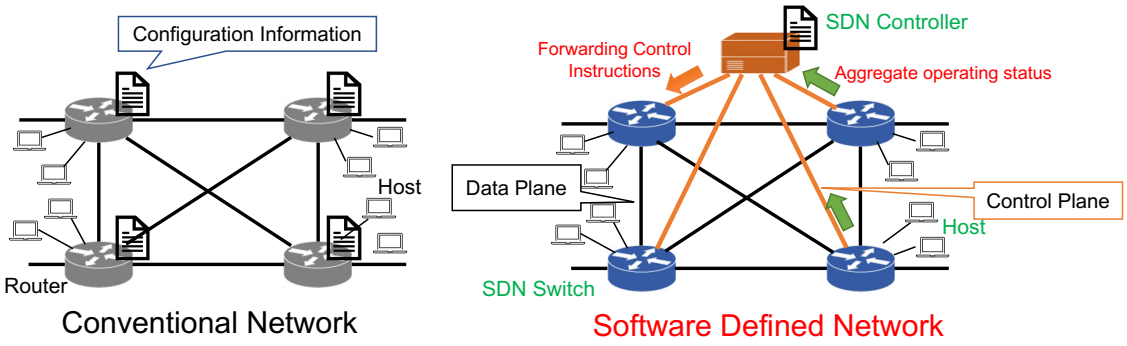

Fig. 1. Comparison of Conventional Network and Software Defined Network.

by SDN controllers and nodes that perform byte integrity verification. Additionally, we implemented this method on a network testbed to obtain per-process communication volume information measured at each host and confirmed that it is applicable to anomaly detection in SDN networks.

The rest of the paper organized as follows. Section II describes the techniques and research associated with this study. Section III explains the proposed network verification scheme that deals with the process-level communication volume of hosts. Furthermore, Section IV describes an experiment in which the proposed method was operated on a test bed. Finally, Section V presents our concluding remarks.

II. Related Work

In this section, we will explain the technology and previous research related to this study.

A. Software Defined Network

SDN is a technology that centrally controls network devices through software. In a conventional network, as shown in Figure 1, the network administrator configures each router with information for routing control, and the router forwards packets according to the configuration. On the other hand, SDN can issue forwarding control instructions to the entire SDN switches by configuring the SDN controller, and the SDN switches perform packet forwarding based on these instructions from the SDN controller. Therefore, SDN allows dynamic control based on the operating 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 [10].

OpenFlow [11] is widely used as a specification for implementing SDN, and OpenFlow controller implementations such as Floodlight exist [12]. Although SDN allows for flexible control of the network, several security issues have been reported with SDN [3] [13]. For example, there are known attacks in which malicious switches attack the data plane or mislead the SDN controller about the network topology, and methods have been reported to solve these problems [4] [5].

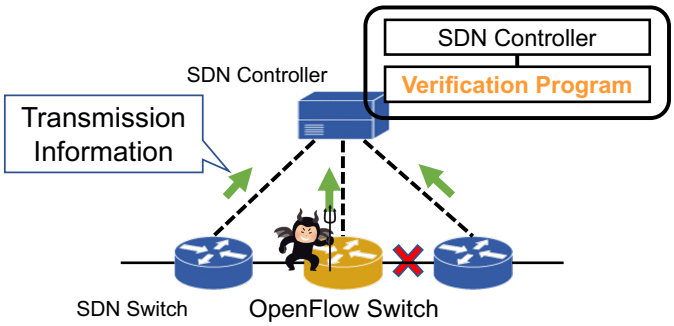

Fig. 2. Byte Consistency Verification by SPHINX.

B. Data Plane Security in SDN

If the network is compromised in SDN, unintended packets may be discarded or generated, routes may be changed. In order to prevent such problems from occurring, there are verification techniques for protecting the data plane. SPHINX verifies compromised switches using a technique called byte consistency verification [4]. 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. This technique receives a report of the forwarding volume information from each switch. Next, from the information received, this 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, this method verifies whether the average value deviates from the moving average value of each SDN switch by the inequality in Equation 1, using a predetermined threshold τ value.

$$\frac{1}{\tau} < \frac{\sum}{\sum_{avg}} < \tau \quad (1)$$

If the threshold τ is smaller than an appropriate value, false positives are likely to occur, and if it is larger than the appropriate value, false negatives are likely to occur. Therefore

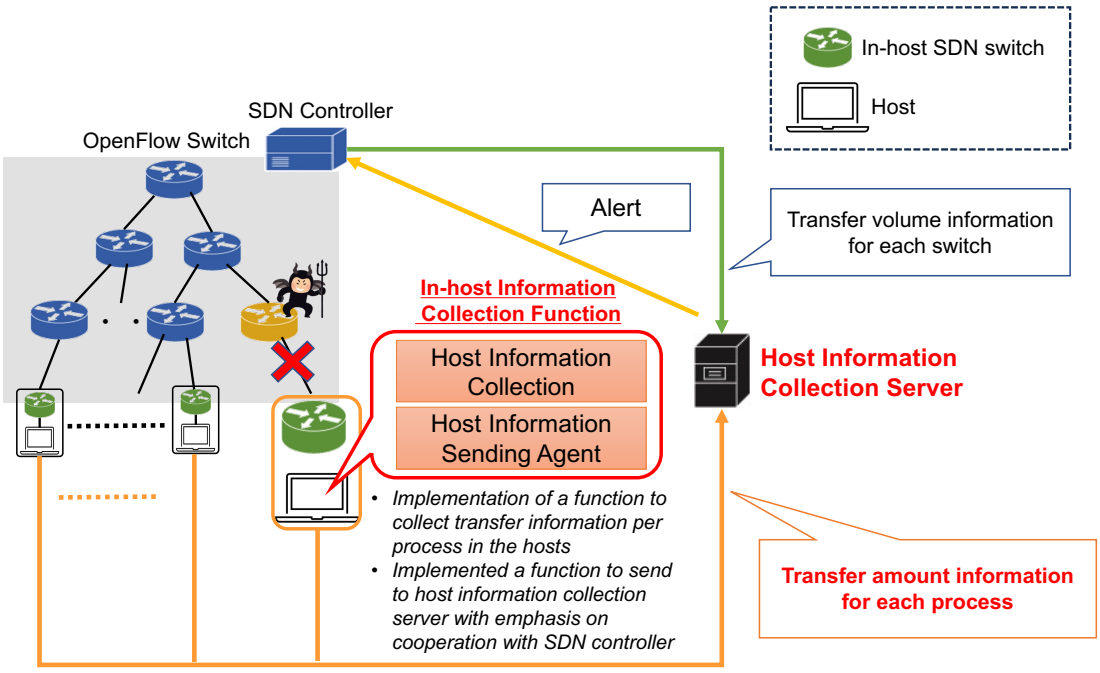

Fig. 3. Overview of the Proposed Method.

the appropriate value for τ varies depending on the network configuration, the type of switch used, it is necessary to set an appropriate value for each network. However, since SPHINX performs byte integrity verification using network switch forwarding volume information, it cannot verify whether an edge switch is malicious or not, and it does not support flow aggregation. In addition, various other SDN data plane security measures have been proposed [13].

C. WhiteRabbit

As described in Section II-B, SPHINX has the issue that the accuracy of detection is affected by variations in the timing of obtaining statistic information from switches. To address this issue, WhiteRabbit has improved the deterioration of verification accuracy due to acquisition timing deviations by using IEEE1588 PTPv2 for high-precision time synchronization and scheduling the timing of acquisition of transfer volume information [8]. 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, there is a problem in that byte consistency verification using only SDN switch information cannot verify edge switches. To solve this problem, we proposed a method to obtain the communication volume of each host [9]. 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, this method requires the threshold τ in Equation 1 to be larger than the conventional method, which may miss minute network anomalies or attacks that take place in a very small amount of time.

III. Proposed Method

In order 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 in Figure 3, host information is collected by implementing an in-host information collection function on hosts on conventional SDN. In addition, we deployed 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. By using this system, it is possible to improve the accuracy of detecting abnormal networks by classifying communication volume using more detailed host information that could not be obtained using the conventional method.

This system requires the implementation of the following two functions.

1) A function for each host to send the collected data to the host information collection server after collecting its own process-level traffic information.
2) A function for the SDN controller to report 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 from the host and compare it with the transfer volume information of each switch. Furthermore, the hosts need to implement a function to collect its

 ISBN: 978-1-68558-141-1

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 and compares it with the transfer volume information of each switch collected by the SDN controller, and outputs 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 command provided by the Linux OS as an external command execution to obtain the cumulative amount of packet reception 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. Experiment

To verify and evaluate the operation of the host information collection function using this method, we implement an experimental network on DeterLab, a network testbed operated by University of Southern California Information Sciences Institute and University of Utah [14].

We used a total of 18 nodes on DeterLab, each with an SDN controller, a verification component, SDN switches (7 nodes), hosts (8 nodes), and a Prometheus server. The network for this experiment is a tree network topology with $Depth = 2$ and $Fanout = 2$. $Depth$ indicates the depth of the hierarchy from the root node, and $Fanout$ indicates how many nodes are connected in one branching. We used Prometheus [15] [16] to implement the host information collection server. Prometheus is a network monitoring system developed by the Cloud Native Computing Foundation that can collect and analyze time-series data. Additionally, as mentioned in Section 3.2, we used Linux's ss command, which can obtain the status of network sockets, to implement the host information collection agent. We primarily used the bytes_received entry (the cumulative number of bytes received by the socket) in the internal TCP information that can be obtained by running the ss command with the i option added.

Table I shows the specifications of the MicroCloud on DeterLab used in this experiment. All 18 nodes we used in this experiment used equipment with the same specifications. We used Floodlight v1.2 [17] as an SDN controller. Also, we implemented an OpenFlow proxy, stopcock [18], between the group of switches and the controller as the verification component for route verification and used ofsoftswitch13_EXT340 [19] as the SDN switches. Since this experimental network consists of actual equipment rather than simulators or emulators, we can verify the operation in an environment that is closer to actual operation.

TABLE I
Specifications of MicroCloud nodes in DeterLab used in This Experiment.

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

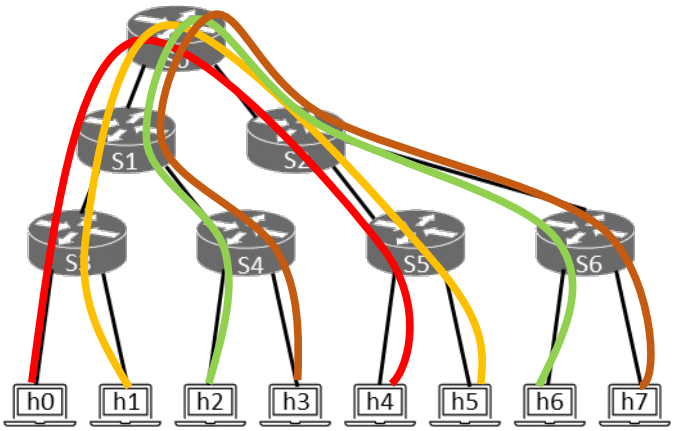

Fig. 4. Traffic Flows in This Experiment.

A. Experimental Method

In this experiment, we generated traffic between hosts in the network built on the testbed described in this section to verify that the host information collection system could correctly transmit host information. Figure 4 shows the traffic flow for a TCP 3-hop path that uses iperf to generate traffic to port 5201. As shown in this figure, we generated traffic simultaneously and mutually between a total of four groups: $h0$ and $h4$, $h1$ and $h5$, $h2$ and $h6$, and $h3$ and $h7$, and confirmed whether the transfer amount information could be collected correctly by the Prometheus server. As a validation of the operation, we verify that the following four items are reported correctly: (1) information reported by the host information collection agent to the Prometheus server, (2) the host's own IP address and port number, (3) the source IP address and port number, and (4) the received bytes. In addition, by verifying that time-series data is produced from the reports and that reports can be received from multiple hosts simultaneously, we evaluate the feasibility of this host information collection system in a real operational environment.

B. Evaluation of the experiment

As an example of a communication volume report sent from a single host to the Prometheus server, Figure 5 shows the

 ISBN: 978-1-68558-141-1

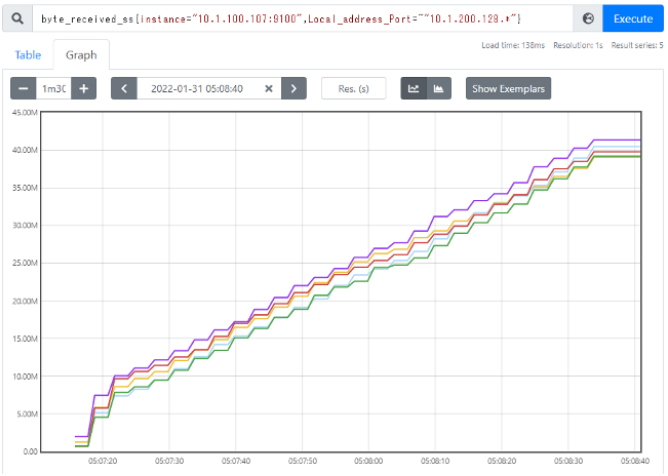

Fig. 5. Number of Received Bytes Associated with iperf Communication in h0 Transition.

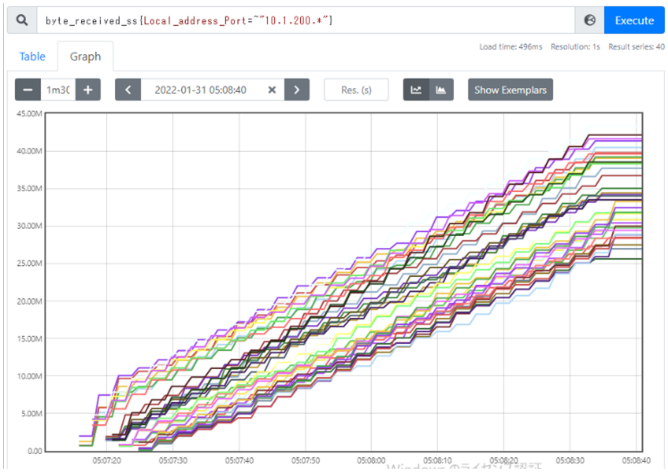

Fig. 6. Number of Bytes Received Related to iperf Communication for All Hosts.

transition of the transfer volume information regarding iperf sent from $h0$. As in the $h0$ example, we observed that iperf traffic was generated on all other hosts as well, with each host reporting an increasing number of bytes received and organized as time-series data. Then, Figure 6 visualizes the change in transfer volume information for all hosts rather than a single host. As shown in this figure, we confirmed that the Prometheus server receives traffic reports from all hosts. By multiplying the information collected from these hosts with the transfer volume information held by SPHINX and WhiteRabbit, byte consistency verification can be achieved with additional host information and is expected to improve the reliability of anomaly detection.

V. Conclusion

With the spread of SDN, technologies to protect the data plane are becoming increasingly important, and there is growing interest in protecting the data plane, which is achieved through software switches. One technology for protecting the data plane is byte consistency verification, which detects anomalies based on communication status data collected from a group of SDN switches.

We have proposed a method to improve detection accuracy by handling transfer amount information on a flow-by-flow basis using high-precision time synchronization. We have also developed a technology that expands the range of devices that can be detected by implementing a switch on the host. Based on the results of these previous studies, we hypothesized that using host information for network verification in SDN could improve the accuracy of network verification, and proposed a method to collect forwarding volume information for each host in this paper.

In this paper, in order to evaluate the proposed method, we used the open source software Prometheus to obtain per-process communication volume information measured at each host and validated a scheme to use it to detect anomalies in SDN networks. To achieve this verification, we developed an agent that allows each host to send traffic information for each process to the Prometheus server, and implemented the proposed system on DETERLab, a network testbed, to confirm its operation. When traffic was generated between the hosts to verify the operation, we checked the traffic volume of each host from the Prometheus server.

As future work, we plan to conduct verification experiments that cross the communication volume information at the process level of each host obtained in this study with SPHINX and WhiteRabbit data.

Acknowledgement

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

This work was supported by JSPS KAKENHI Grant Number JP19K20252.

References

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

[2] “HTTPS encryption on the web,” https://transparencyreport.google.com/https/overview?lang=en&hl=en (Accessed: Dec 10, 2023).

[3] 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.

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

[5] 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.

[6] 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.

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

 ISBN: 978-1-68558-141-1

[8] 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.
[9] 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).
[10] K. Benzekki, A. Fergougui, and A. Elalaoui, “Software- defined networking (SDN): a survey,” Security and communication networks 9.18, pp. 5803-5833, 2016.
[11] 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.
[12] 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.
[13] 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.
[14] “DeterLab,” https://www.isi.deterlab.net/ (Accessed: Dec. 10, 2023).
[15] “Prometheus,” https://prometheus.io (Accessed: Dec. 10, 2023).
[16] “GitHub - prometheus/Prometheus: The Prometheus monitoring system and time series database,” https://github.com/prometheus/prometheus (Accessed Dec. 10, 2023).
[17] “Floodlight Controller,” https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview (Accessed Dec. 10, 2023).
[18] P. Wood, “stopcock,” https://github.com/tignetworking/stopcock/ (Accessed Jun. 24, 2019).
[19] “ofsoftswitch13_EXT-340,” https://github.com/oronanschel/ofsoftswitch13_EXT-340 (Accessed Dec. 10, 2023).

 ISBN: 978-1-68558-141-1