

Quick Blocking Operation of IDS/SDN Cooperative Firewall Systems by Reducing Communication Overhead

Akihiro Takai*, Yusei Katsura**, Nariyoshi Yamai*, Rei Nakagawa*, and Vasaka Visoottiviseth***

* Graduate School of Engineering, Tokyo University of Agriculture and Technology, Tokyo, Japan

** Graduate School of Science and Technology, Nara Institute of Science and Technology, Nara, Japan

*** Faculty of Information and Communication Technology, Mahidol University, Nakhon Pathom, Thailand

s224164x@st.go.tuat.ac.jp, katsura.yusei.ky6@is.naist.jp, nyamai@cc.tuat.ac.jp,

rnakagawa@go.tuat.ac.jp, vasaka.vis@mahidol.edu

Abstract—An Intrusion Detection System (IDS) / Software Defined Networking (SDN) cooperative firewall system has attracted much attention recently because it has many advantages of dynamic network configuration with SDN and scalable IDS hosts. In the IDS/SDN cooperative firewall system, an SDN switch relays traffic between a client and a server and mirrors traffic from a client to an IDS host. The IDS host monitors the mirrored traffic and notifies the SDN switch to block malicious traffic according to the detection of the attack. At this point, malicious packets reach the server until the IDS detects the attack and notifies it. In this paper, we propose a method to speed up mirroring and notification by integrating IDS and SDN switch hosts as a method to shorten the blocking time and compare it with existing methods. The experimental system was constructed using Raspberry Pi3 B+ and 4B boards. As a result, it was confirmed that the proposed method completes the blocking operation faster than the existing method. We also investigated the breakdown of the blocking time to confirm the effect of the proposed method.

Keyword— Firewall, Intrusion Detection System, OpenFlow, Software Defined Network

I. INTRODUCTION

TO prevent computing devices of the organization from being compromised, a firewall, intrusion detection system (IDS), and intrusion prevention system (IPS) that can detect and block malicious files are essential. For quick and flexible network management and maintenance in large

organization networks, the Software-Defined Network (SDN) should be used [1]. Upon integrating SDN with IDS, network administrators can deploy a flexible firewall system, namely the IDS/SDN cooperative firewall system [2], [3]. In this integration, the SDN switch that relays the bidirectional communication traffic of the target network mirrors the traffic to the IDS, and the IDS will detect anomaly packets that may be attacks from an outsider and inform the SDN controller. Once the SDN controller learns about attacks, it will create rules or policies to push to SDN switches to block malicious traffic coming into the network to be protected. In addition, multiple IDS servers can be used to provide load balance between IDS servers. Furthermore, by using the integration of SDN and IDS, not only a flexible firewall, but network administrators can also configure flexible network routes, for example, forwarding anomaly traffic to honeypots for further security analysis.

The large delay required for the period to mirror packets from the SDN switch to the IDS and notify attack detection from the IDS to the SDN controller (the blocking time) allows malicious packets to enter the protected network in the meantime. In the existing IDS / SDN cooperative firewall system, the OpenFlow, which is the leading implementation of the SDN concept, conventionally uses the REST API for notification from IDS [2]. However, because most IDS implementations do not support the REST API, the system must use the log monitoring tools to observe the change of IDS alert logs and configure the SDN controller to block this anomaly traffic via REST API, resulting in increasing the blocking time.

To reduce the blocking time, in the previous work, we have proposed a method to change the notification method from REST API to Syslog, which is faster because of reducing communication overhead [4]. To further reduce the blocking time along with using the fast Syslog notification method, this paper demonstrates a method of integrating IDS hosts and SDN switch hosts, which are detached physical hosts in existing systems, into a single physical host, so that mirroring and notification can be completed within a single physical host. Therefore, combining the integration method and the fast Syslog notification mitigates the communication overhead compared with the existing system of the separate physical hosts and reduces the blocking delay of malicious traffic. To evaluate the effectiveness of the system, we conducted an experiment to compare the blocking time of the existing system in which the IDS and SDN switch hosts are

Manuscript received December 8, 2022. This work is a follow-up of the invited journal to the accepted & presented paper entitled "Quick blocking operation of firewall system cooperating with IDS and SDN" of the 24th International Conference on Advanced Communication Technology (ICTACT2022).

Akihiro Takai, he is a master's student in the Graduate School of Engineering at Tokyo University of Agriculture and Technology, Japan. (e-mail: s224164x@st.go.tuat.ac.jp)

Yusei Katsura, he is a Ph.D. student in the Graduate School of Science and Technology at Nara Institute of Science and Technology, Japan. (e-mail: katsura.yusei.ky6@is.naist.jp)

Nariyoshi Yamai, he is a professor in the Graduate School of Engineering, Tokyo University of Agriculture and Technology, Japan. (corresponding author, phone: +81-42-388-7695, e-mail: nyamai@cc.tuat.ac.jp)

Rei Nakagawa, he is an assistant professor in the Graduate School of Engineering at Tokyo University of Agriculture and Technology, Japan. (e-mail: rnakagawa@go.tuat.ac.jp)

Vasaka Visoottiviseth, she is an associate professor in the Faculty of Information and Communication Technology at Mahidol University, Thailand. (e-mail: vasaka.vis@mahidol.edu)

detached and the proposed system in which the IDS and SDN switch hosts are integrated. In the experiment, two kinds of experimental systems, the existing method and the proposed method, were constructed with real devices, and a total of four blocking times were measured by the combination of two kinds of notification methods, REST API and Syslog. We make following contributions.

- Proposal of quick blocking operation of integrating IDS hosts and SDN switch hosts using the fast notification method.
- Implementation of the proposed method using Raspberry Pi 4B and 3B+ as devices, Suricata as IDS, and Open vSwitch (OvS) as SDN switch implementation.
- Experimental verification of the reduction of blocking time by host integration in combination with two types of notification methods, REST API and Syslog.

II. BACKGROUND AND EXISTING WORKS

A. Software Defined Network

In the traditional network, once any network policy changes, network administrators have to carefully configure all switches manually. It requires a great deal of network maintenance. To solve this problem, the SDN is introduced. In the SDN concept, network tasks are divided into the control plane and the data plane. The control plane, which is responsible for making decisions on how packets should be handled, is the task of the SDN controller. On the other hand, the data plane or forwarding plane, which is responsible for handling packets based on the instructions from the control plane, is the task of the SDN switch. Instructions from the SDN controller can be either forwarding or dropping packets on the SDN switch. It can manage or control the forwarding table residing on SDN switches. On the other hand, SDN switches can focus on accelerating the packet forwarding.

By using SDN, administrators can centrally maintain the network policy via the SDN controller. Instead of manually updating each switch, SDN enables the network administrators to distribute the policy evenly across multiple switches. This can simplify network management.

In SDN, there are Northbound and Southbound APIs that are APIs operating between data plane, control plane and application plane. The Northbound APIs are available on an SDN controller and allow applications or the application plane to interact with the controller, which is the control plane. Applications and services are, for example, load-balancers and firewalls. On the other hand, the Southbound APIs are available between the SDN controller in the control plane and other forwarding devices, e.g. switches, and routers, which are the data plane or forwarding plane. By using the Southbound APIs, administrators can adjust the network according to the change requirements.

If you want to submit your file with one column electronically, please do the following:

1) OpenFlow

OpenFlow is a well-known technology maintained by the Open Networking Forum (ONF) [5] that implements the SDN concept. The OpenFlow protocol is a set of specifications that provides the southbound interface and defines how the controller interacts with the data plane. The newest version of the OpenFlow Switch specification is version 1.5.1. Each OpenFlow switch will store information

about the flow entry received from the controller into the flow table. Each flow entry contains an explicit action to handle each flow.

On an OpenFlow switch, there are two main components: the switch-agent and the data plane. The switch-agent uses the OpenFlow protocol to communicate with one or more controllers. Moreover, it communicates with the data plane using the internal protocol.

Every OpenFlow message begins with the same header structure containing version, type, length, and transaction id (xid). The OpenFlow message types can be, for example, FlowMod, PacketIn, FlowRemoved, PacketOut, and StatsReq. The FlowMod message is used to allow the controller to modify the flow entries on the OpenFlow switch. On the other hand, the PacketIn message type is used by the switch to send incoming packets to the OpenFlow controller. Normally, there are two cases where PacketIn message type is used: (1) there is an explicit action for this behavior specified in the flow entry and (2) the flow does not match any flow entry in the table. Moreover, to make the OpenFlow controller understand that the traffic comes from which switch, each switch will be assigned a Datapath ID.

There are many OpenFlow controller software, for example, Ryu, Treman, Opendaylight, etc. In this research, we selected Ryu as the SDN controller software.

2) Ryu

Ryu [6] is a framework that implements the OpenFlow controller and is developed in Python. Once the Ryu controller receives an OpenFlow message from a switch, it will trigger an event. An event handler contains an event class and the state of the switch. States of OpenFlow switch can be, for example, the HANDSHAKE_DISPATCHER, which is the initial state that exchanges the HELO message, and the MAIN_DISPATCHER, which is the normal state. When the controller receives a PacketIn message from the switch, the EventOFPPacketIn event class will be called. For example, with the API `set_ev_class(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)`, we can define the process when the OpenFlow switch is in the normal operation state and receives a PacketIn message [7].

B. Related Works

Here, we surveyed related works that utilize Snort IDS [8] and SDN together to dynamically filter anomaly traffic. Nam *et al.* studied SDN security enhancement using open-source IDS/IPS Suricata [9][10]. They proposed requirements for implementing SDN security. They mentioned whether SDN solutions, Suricata IDS/IPS, automated intrusion prevention, and mirroring are necessary or not when they want to implement a firewall, network scan detection, abnormal traffic detection, intrusion detection, and intrusion prevention. They concluded that to implement intrusion prevention they need Suricata IDS/IPS, automated intrusion prevention, as well as mirroring because the SDN alone cannot handle this. However, no implementation and performance were shown in their paper.

Hendrawan *et al.* studied the performance degradation when integrating the SDN architecture with an IDS, either the signature-based Snort or the anomaly-based Bro [11]. The throughput, delay, and packet loss, which are required performance metrics of quality of service (QoS), are observed. The results of their experiment on the mininet virtual network [12] showed that integrating SDN with Bro IDS gave better performance than using Snort IDS in all

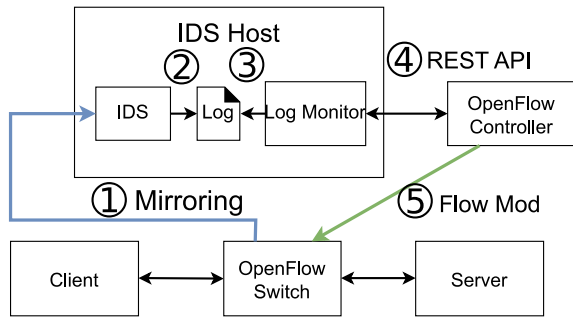


Fig. 1. System architecture of the Original IDS/SDN Cooperative Firewall System.

aspects: throughput, delay, and packet loss. However, the delay is not much different, while the CPU usage and memory usage of Snort are lower than those of Bro.

R. Sutton et al. designed and developed a system that utilized an SDN to divert traffic from some suspicious traffic multiple Snort IDS machines for inspection [13]. The authors developed ‘PySnorter,’ a Python tool for routing traffic to dedicated Snort machines. Once Snort alerts the malicious traffic, it pulls the information from the alert and generates a REST API command that is then sent to the SDN controller. For the experiments, they implemented a virtualized network using GNS3, which is the graphical version of Network Simulator-3, and used Open vSwitch. However, latency was not measured.

C. The Original IDS/SDN Cooperative Firewall System

In firewall systems cooperating with IDS and SDN, the IDS generally do not have the capability to send alert messages directly to the SDN controller, but instead has the capability to send them to an alert handler as Syslog messages or record them into a log file. Typically, Syslog messages are in the common event format (CEF) or log event extended format (LEEF) to facilitate analysis. However, similar to what was proposed in the [12], in the case of a firewall system recording alert messages in a log file and using OpenFlow as an SDN platform, the typical behavior of the firewall system to block malicious packets consists of the following six steps, as illustrated in Fig. 1.

- (1) When the OpenFlow switch receives packets from the client that are destined to the server, it will delay the forwarding process to the destination and copy the traffic to IDS via port mirroring.
- (2) Once the IDS receives those packets, it will analyze and detect malicious packets. If the IDS finds malicious packets, it will alert and write the information to the log file.
- (3) The log monitoring tool continuously monitors that log file.
- (4) When the log monitoring tool detects a change in the log file, that is, a new alert message written in the log file, it will use the REST API to send the information of that malicious packet to the OpenFlow controller.
- (5) The OpenFlow controller uses the received information to create a flow entry that determines how an OpenFlow switch must behave when it receives packets from that malicious flow. Then, the OpenFlow controller will use the FlowMod message to send that Flow entry to the OpenFlow switch.
- (6) The OpenFlow switch updates the flow table based on the information specified in the FlowMod message. Then, it can process packets that are delayed in step 1.

In this case, because most IDS do not support REST APIs,

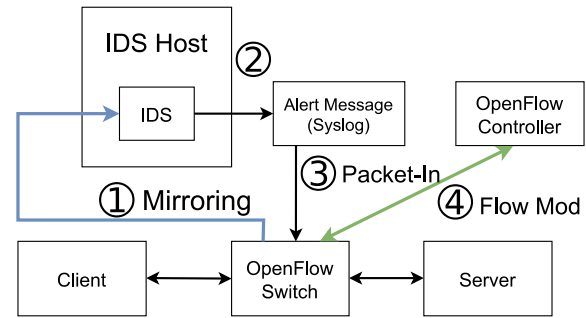


Fig. 2. System architecture of the Quick block operation by Syslog notification.

they have to use log monitoring tools to monitor log change and use a command line tool such as ‘curl’ to transfer REST APIs. Therefore, a processing overhead will occur. Moreover, REST APIs use HTTP methods and run over TCP protocol.

D. Quick Block Operation by Syslog Notification

To reduce communication overhead due to REST API, we have proposed the fast notification method for the IDS/SDN cooperative firewall system as illustrated in Fig. 2. There are five steps to perform.

- (1) When the OpenFlow switch receives packets from the client that are destined to the server, it will delay the forwarding process to the destination and copy the traffic to IDS via port mirroring.
- (2) If the packet matches with any signatures in the IDS, the IDS will send the alert message to the OpenFlow switch by using, for example, the Syslog message or the SNMP trap.
- (3) When the OpenFlow switch receives the alert message sent via UDP, it will forward the message as the PacketIn message type to the OpenFlow controller.
- (4) The OpenFlow controller uses the received information to create a flow entry that determines how an OpenFlow switch must behave when it receives packets from those malicious flows. Then, the OpenFlow controller will use the FlowMod message to send that Flow entry to the OpenFlow switch.
- (5) The OpenFlow switch updates the flow table based on the information specified in the FlowMod message. Then, it can process packets that are delayed in step 1.

Note that this method can reduce one step compared with the existing method mentioned in Section II B, because we send the alerts as the Syslog message directly to OpenFlow switch, but the existing method needs to write the alerts in the log file first and use the log monitoring tools to detect the log change. The characteristics of REST API and Syslog as notification methods can be summarized as follows.

- REST API

The REST API submissions use an external program Swatchdog to monitor logs and curl to send packets. The POST method of the REST API is used to send notifications as well as host blocking. Since TCP is used for communication, it is expected that there will be overhead to establish the connection.

- Syslog

Most IDSs can notify anomalies using Syslog. Syslog can communicate via either UDP or TCP. In an IDS/SDN cooperative firewall system, it is considered better to use UDP for low latency communication.

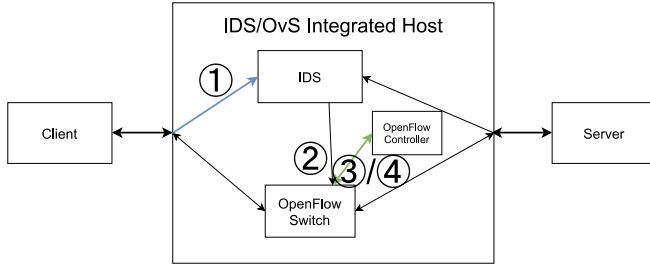


Fig. 3. System architecture of the proposed work.

 TABLE I
SOFTWARE SPECIFICATION

Device	Software
OpenFlow controller	Ryu 2.7
OpenFlow switch	Open vSwitch 2.10.1 (OvS)
Intrusion Detection System	Suricata 6.0.6
Log monitoring tools	swatch

III. PROPOSED WORK

To further reduce the blocking time, we adopt an approach to reduce the communication overhead between separated physical hosts of IDS and SDN, as well as that of existing notification method, REST API. Therefore, we propose a method to reduce the blocking time by running an IDS and an SDN switch implementation, such as Open vSwitch, on a single device, thereby reducing the communication overhead in the mirroring in step 1 and the notification in step 4, as illustrated in Fig. 3. In the IDS/SDN cooperative firewall system experimented in the previous study [4], the host acting as an SDN switch and the IDS host are different devices. Therefore, we propose a method to reduce the blocking time by running an IDS and an SDN switch implementation, such as Open vSwitch, on a single device, thereby reducing the communication overhead in the mirroring in step 1 and the notification in step 4. An IDS/SDN cooperative firewall system that integrates an IDS host and an SDN switch host in a single device is called an integrated system, while one that has different hosts for each is called a detached system.

In our implementation, we implement two approaches: (1) the existing detached system and (2) our proposed integrated system. Details of them are described below.

IV. IMPLEMENTATION

For the implementation, as mentioned earlier in Section II, we select Ryu as the OpenFlow controller. For the specification of OpenFlow, we use OpenFlow version 1.3. For the IDS, we select Suricata, which is a famous Open Source Software (OSS). Suricata has a large community and has more than thousands of detection signatures available. Table I summarizes the software and its version used in our experiments.

Moreover, we used two Raspberry Pi 3B+ boards and a Raspberry Pi 4B board to emulate each device. First, Raspberry Pi 3B+ is used to implement a client host and a server host, while Raspberry Pi 4B is used for implementing IDS/OvS integrated host. In the detached system, we use Raspberry Pi 3B+ for the client host, server host and IDS host,

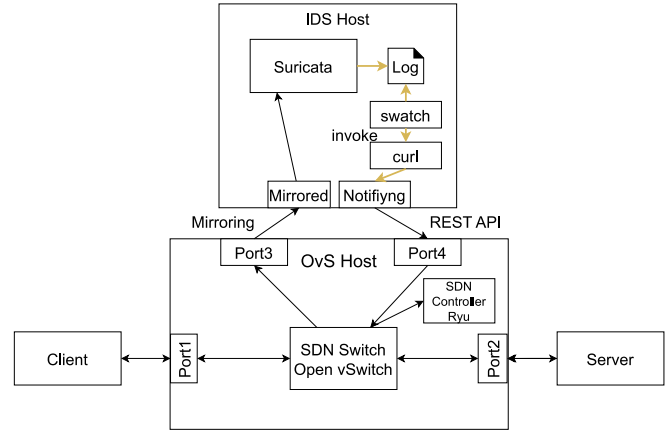


Fig. 4. Implementation of the detached system.

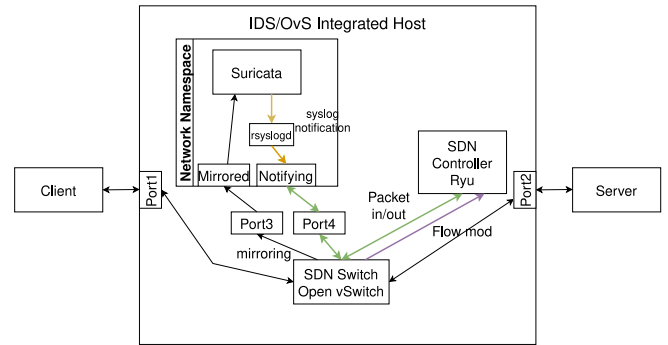


Fig. 5. Implementation of the integrated system.

while Raspberry Pi 4B is used for implementing OvS host.

A. Implementation of the Detached System

First, we implement the existing approach that uses Suricata to write alerts to the log file and uses swatch to monitor the log file. As shown in Fig. 4, in this approach Suricata and the swatch are running on the same Raspberry Pi board. Moreover, the Raspberry Pi that emulates both Ryu controller and Open vSwitch is equipped with four ports: Port 1 connecting to the client, Port 2 connecting to the server, Port 3 which is a one-way directional port just to forward packets to Suricata to analyze, and Port 4 which is also a one-way directional port for Ryu to receive REST APIs over TCP from the swatch utility. Moreover, there are only FlowMod messages sent from Ryu to OvS when the Ryu controller wants to modify the flow table on OvS.

B. Implementation of the Integrated System

Fig. 5 shows a diagram of the IDS and SDN switch. A Linux network namespace is running on the IDS/OvS integrated host. Using the Network Namespace function, IP-related processes can be divided into multiple processes within a single Linux unit. Suricata runs within a Network Namespace on the integrated host. This Network Namespace has two Ethernet: one corresponds to port 3 and is used to receive packets from the client and server ports (Ports 1 and 2), and the other corresponds to port 4 and is used by Suricata to notify the outside of Network Namespace.

First, packets arriving from the client are forwarded to the server and simultaneously mirrored to the Network Namespace which the IDS operates. Packets arriving from

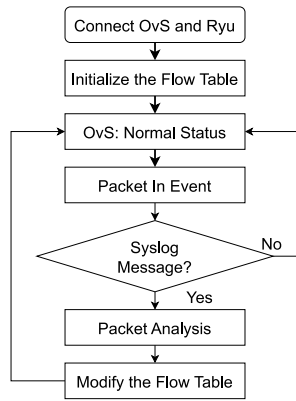


Fig. 6. Flowchart of the process inside the Ryu controller.

 TABLE II
FLOW TABLE IN THE INITIAL STATE

Match	Action
in_port 1	OutPut (Port 2, Port 3)
in_port 2	OutPut (Port 1, Port 3)
in_port 4	OutPut (Controller)

 TABLE III
FLOW TABLE AFTER GOT ATTACKS

Match	Action
in_port 1	OutPut (Port 2, Port 3)
in_port 2	OutPut (Port 1, Port 3)
in_port 4	OutPut (Controller)
ip_src 192.168.10.21	Drop
tcp_dst 80	
eth_type = 0x0800	

the server are also forwarded to the client and mirrored to the network namespace. The IDS monitors the packets and notifies the SDN controller across the network namespace using Syslog if any malicious packet is detected. The SDN controller receives the notification and modifies the Open vSwitch flow table to drop the packets from the host that sent the packets deemed to be malicious. This allows the mirroring and notification procedures of the blocking operation to be completed in a single device.

Fig. 6 shows the flowchart of the processes inside our Ryu controller. Once the OvS connects with the Ryu controller, it will initialize the flow table as shown in Table II and set the state of the OvS as normal. When the Ryu controller receives the PacketIn message, it will check whether the message is a Syslog message or not. If so, it will analyze the message and modify the flow table. An example of a flow table that is modified after receiving DDoS attacks is shown in Table III. A new flow entry is added to the flow table specifying the “Drop” action when the packet contains the header that matches the specified condition.

V. EVALUATION RESULTS

For evaluation, we compare our proposed work with the existing approach and the proposed approach described in Section IV A and Section IV B, respectively. To measure the blocking time, we observe the packet timestamp using the tcpdump command. To investigate the breakdown of

 TABLE IV
BLOCKING TIME OF EACH APPROACH

	REST API	Syslog
Detached System	75.6 ms	9.6 ms
Integrated System	67.3 ms	4.6 ms

blocking time, we define the communication overhead (i.e., the blocking time) using the following periods of processing in the IDS/SDN cooperative firewall system.

1. Mirroring — the time taken to mirror packets from OvS to IDS.
2. Log monitoring — the time it takes for Swatch to detect IDS writes to the log.
3. Start script — time required for Swatch to start the REST API submission script.
4. Notification — the time taken to notify in REST API.
5. Syslog — the time taken for IDS to notify in Syslog.
6. Packet In/Flow Mod — the time it takes for OvS to packet in a Syslog packet and invoke Flow Mod processing.

Both existing and proposed approaches measure the blocking time from the time OvS host receives a malicious packet until SDN controller Ryu receives a Syslog or REST API Packet-in packet and calls the Flow Mod process. Table IV summarizes the blocking time for each method. First, we note the performance differences due to differences in the notification methods. When comparing Figs. 7 and 8, and Figs. 9 and 10, respectively, using Syslog as the notification method, the overhead of log monitoring, REST API sending script invocation, and notification processing required when using REST API was reduced, and the blocking operation could be performed at a speed of approximately 60 ms. In particular, we observed that it was taking a long time to send REST APIs using curl.

Next, we discuss the differences between the results of the detached and integrated systems. Comparing Figs. 7 and 9, and Figs. 8 and 10, respectively, the integrated system completed the blocking operation faster, and the breakdown suggests that this is due to faster mirroring, notification, Packet-in, and Packet-out. Furthermore, comparing the detached and integrated systems in the results using Syslog as the notification method, the time spent for communication was approximately 0.6 ms for the detached system and approximately 0.3 ms for the integrated system. When using the REST API, the time spent for mirroring was approximately 2.2 ms for the detached system and 0.27 ms for the integrated system. These results indicate that the proposed method reduces the communication overhead. Although there are differences in the boards on which the IDS is running, the fact that mirroring and notification are completed within a single host is one of the factors contributing to the reduction in blocking time. To more accurately observe the effect of the proposed method, it is necessary to conduct experiments on a unified board on which the IDS operates.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a method to reduce blocking time by integrating IDS hosts and SDN switch hosts along with the fast notification method and we have confirmed the effectiveness of the proposed method through experiments.

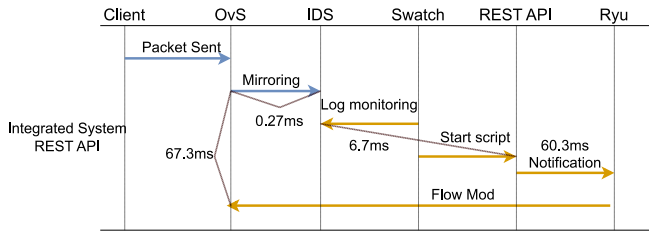


Fig. 7. Blocking time of Integrated System when using REST API.

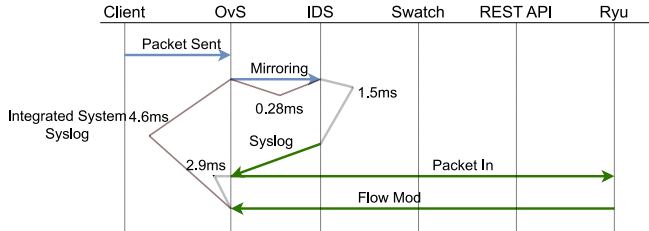


Fig. 8. Blocking time of Integrated System when using Syslog.

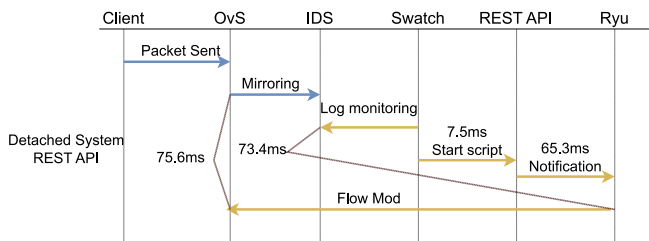


Fig. 9. Blocking time of Detached System when using REST API.

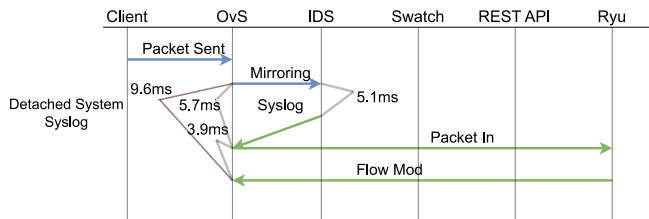


Fig. 10. Blocking time of Detached System when using Syslog.

In the experiments, the blocking time was measured by combining two types of notification methods and two types of IDS/SDN cooperative firewall systems: the integrated system proposed here and the detached system. The experimental results confirm that the notification method using Syslog is faster than the REST API because the Syslog notification method removes the process of monitoring IDS log for REST API. In addition, integrating IDS and SDN hosts reduced the communication overhead between the IDS and SDN, which were physically separated in conventional, and as a result, further reduced the blocking time at no load on the system. We also confirmed that the reduction of communication overhead, which is the goal of the proposed method, contributes to faster blocking operation by examining the breakdown of the blocking time. However, the proposed method and the existing method differ in the host on which the IDS operates, and we have not confirmed how the blocking times of the two systems change when a traffic load is added to them. Since the integrated system is more susceptible to the huge network traffic load than the detached system since IDS and OvS are running on a single host, the performance of the integrated system is expected to be

degraded under high load. Therefore, for future work, an immediate verification method is required.

Moreover, the IDS/SDN cooperative firewall system using the IDS/OvS integrated host has an advantage over existing IDS/SDN cooperative firewall systems in terms of parallelization of OvS hosts. This is because the IDS and OvS hosts are integrated, which simplifies the network configuration. Parallelization of IDS and OvS hosts is effective to suppress the increase in shutdown time of IDS/SDN linked firewall system under load environment. In the future, we will construct an OvS host parallelized IDS/SDN cooperative firewall system using an IDS/OvS integrated host and compare its performance with existing parallelized configurations and examine its qualitative cost.

REFERENCES

- [1] 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, pp 69–74, April 2008.
- [2] P. Zanna, B. O'Neill, P. Radcliffe, S. Hosseini, and M. S. Ul Hoque, "Adaptive threat management through the integration of IDS into Software Defined Networks", in *Proc. 2014 Int. Conf. and Workshop on the Network of the Future (NOF)*, pp. 1-5, 2014.
- [3] Y. Katsura, H. Kimiyama, T. Tsutsumi, N. Yonezaki, J. Ichikawa, and M. Maruyama, "Proposal of real-time brute-force attack detection and blocking system using software switch", *IEICE Technical Report*, vol. 118, no. 465, NS2018-272, pp. 461-464, 2019 (in Japanese).
- [4] Y. Katsura, P. Sakarin, N. Yamai, H. Kimiyama, and V. Visoottiviseth: "Quick blocking operation of firewall system cooperating with IDS and SDN," in *Proc. 24th Int. Conf. Advanced Communications Technology (ICAICT 2022)*, pp.393-398, February, 2022.
- [5] The Open Networking Foundation, "SDN Technical Specifications" [Online]. Available: <https://opennetworking.org/software-defined-standards/specifications/>
- [6] Ryu SDN Framework Community, "Ryu SDN Framework" [Online]. Available: <https://ryu-sdn.org/>
- [7] Nippon Telegraph and Telephone Corporation Revision d6cda4f4, "Ryu application API" [Online]. Available: https://ryu.readthedocs.io/en/latest/ryu_app_api.html
- [8] Cisco and/or its affiliates, "Snort – Network Intrusion Detection & Protection System" [Online]. <https://snort.org/>
- [9] The Open Information Security Foundation (OISF), "Home – Suricata" [Online]. Available: <https://suricata.io/>
- [10] K. Nam and K. Kim, "A Study on SDN security enhancement using open source IDS/IPS Suricata," in *Proc. 2018 Int. Conf. Information and Communication Technology Convergence (ICTC)*, pp. 1124-1126, Jeju, South Korea, 2018.
- [11] H. Hendrawan, P. Sukarno, and M. A. Nugroho, "Quality of Service (QoS) Comparison Analysis of Snort IDS and Bro IDS Application in Software Define Network (SDN) Architecture," in *Proc. 2019 7th Int. Conf. Information and Communication Technology (ICoICT)*, pp. 1-7, Kuala Lumpur, Malaysia, 2019.
- [12] Open Networking Foundation, "MININET - Open Networking Foundation" [Online]. Available: <https://opennetworking.org/mininet/>
- [13] R. Sutton, R. Ludwiniak, N. Pitropakis, C. Chrysoulas and T. Dagiuklas, "Towards an SDN Assisted IDS," in *Proc. 2021 11th IFIP Int. Conf. New Technologies, Mobility and Security (NTMS)*, pp. 1-5, Paris, France, 2021.



Akihiro Takai was born in Japan in 1999 and received his B.E. from Tokyo University of Agriculture and Technology (TUAT), Japan in 2022. Currently, he is a master's student at Tokyo University of Agriculture and Technology, Japan. His research interests include computer networks.



Yusei Katsura was born in Japan in 1996 and received his B. Info. Env. degree from Tokyo Denki University, Japan in 2019, and his M. S. degree in computer engineering from Nara Institute of Science and Technology (NAIST), Japan in 2022. He was a research student at Tokyo University of Agriculture and Technology (TUAT), Japan in 2019-2020. He is currently a Ph.D. student at Nara Institute of Science and Technology. His research interests include computer networks.



Nariyoshi Yamai was born in Japan in 1961 and received his B.E. and M.E. degrees in electronic engineering and Ph.D. degree in information and computer science from Osaka University, Japan in 1984, 1986, and 1993, respectively. Currently, he is a professor at the Institute of Engineering, Tokyo University of Agriculture and Technology (TUAT), Japan. His research interests include distributed systems, network architecture, network security, and the Internet.



Rei Nakagawa was born in Japan in 1993, received his B.S. and M.S. degrees from the Tokyo University of Science, Japan, in 2016, and 2018 respectively, and a Ph.D. degree in informatics and engineering from the University of Electro-Communications (UEC), Japan, in 2021. He has been an assistant professor at the Institute of Engineering, Tokyo University of Agriculture and Technology (TUAT), Japan since April 2021. His research interests include network architecture, video streaming technology, software defined network, and information centric networking.



and digital forensics.

Vasaka Visoottiviseth was born in Thailand in 1975, received her M.E. and B.E. degrees from Tokyo University of Agriculture and Technology (TUAT), Japan in 1999 and 1997, respectively, and received her Ph.D. degree in computer engineering from Nara Institute of Science and Technology (NAIST), Japan in 2003. Currently, she is an associate professor at Mahidol University, Thailand. Her current research interests include mobile and wireless computing, network security,