

Quick Blocking Operation of Firewall System Cooperating with IDS and SDN

Yusei Katsura*, Pranpariya Sakarin**, Nariyoshi Yamai***, Hiroyuki Kimiyama****,
Vasaka Visoottiviseth**

* 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

*** Institute of Engineering, Tokyo University of Agriculture and Technology, Tokyo, Japan

**** School of Informatics, Daido University, Nagoya, Japan

katsura.yusei.ky6@is.naist.jp, pranpariya.sak@gmail.com, nyamai@cc.tuat.ac.jp, kimiyama@daido-it.ac.jp,
vasaka.vis@mahidol.edu

Abstract— Firewalls, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs) are normally used to filter anomaly traffic and prevent attacks from the Internet. However, preconfiguring firewalls and IDS on multiple devices is an exhausting work for the network administrators. Software Defined Network (SDN) is the concept proposed to make the network management easier by using an SDN controller and SDN switches. In this research, we propose a system that integrates IDS together with SDN in order to block anomaly traffic in a fast manner. Once the IDS detects anomaly traffic, it will send an alert message back to the SDN switch. Then, this alert message will be sent as a PacketIn message to the SDN controller in order to set up rules to block the attack. To evaluate our system, we conduct experiments to compare the performance of our proposed system using syslog and Socket API with the existing method that uses REST API, and another comparison method, in term of processing time. Our experiment results confirm that our proposed method can result in the smaller latency and can quickly block malicious traffic.

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

I. INTRODUCTION

In order to prevent computing devices in the organization from being compromised, firewall, intrusion detection system (IDS), and intrusion detection system (IPS) that can detect and block malicious files are essential. For the quick and flexible network management and maintenance in large organization networks, Software Defined Network (SDN) [1] should be utilized. While integrating SDN with IDS, the network administrators can deploy a flexible firewall system [2][3]. In this integration, IDS will detect anomaly packets that may be attacks from the outsider and inform the SDN controller. Moreover, multiple IDS servers can be used for providing the load balance between IDS servers as well.

Once the SDN controller learns about attacks, it will create rules or policies to push to SDN switches in order to block that malicious traffic coming into the network. Moreover, by using the integration of SDN and IDS, not only a flexible

firewall, network administrators can also configure flexible network routes, for example, forwarding anomaly traffics to honeypots for further security analysis as well.

For the communication between SDN and IDS, OpenFlow, which is the leading implementation of the SDN concept, supports the controller's configuration via REST API. Network administrators can use this API to manage their OpenFlow controller remotely. However, because most IDS do not support REST API, in the existing research work [2], once the IDS detects anomaly traffic and generates alert logs, the system will use the log monitoring tools to observe the log's change and configure the SDN controller to block those anomaly traffic via REST API, based on the IDS log's information.

However, because REST API runs over a TCP connection, there is a need to do the connection establishment, i.e. TCP 3-way handshake, before sending data. This will cause a delay and make the network unable to block malicious incoming traffic immediately.

In this research, we focus on reducing the latency since the IDS detects anomaly traffics until our system can block that traffic by using syslog and PacketIn approach instead of using the log monitoring utility and REST API. In our approach, alerts from the IDS are sent as syslog messages over the UDP connection, in the form of PacketIn to the OpenFlow controller. A similar approach is used in [4], where the DNS server sends a UDP message telling the client's IP address to the OpenFlow controller via an OpenFlow switch as PacketIn. However, our approach uses syslog messages from the IDS as they are. Therefore, by using our approach, many existing IDSs can be used without any modification.

To evaluate the effectiveness of our system, we conduct experiments to compare the latency between the existing system using log monitoring utility and REST API and our proposed system using syslog and PacketIn.

II. BACKGROUND AND EXISTING WORKS

A. Software Defined Network

In the traditional network, once any network policy changes, the network administrators have to carefully configure all switches manually. It requires a great deal of network maintenance. To solve this problem, the Software-Defined Networking (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 just focus on accelerating the packet forwarding.

By using SDN, administrators can centrally maintain the network policy via the SDN controller. Instead of updating each switch manually, SDN enables the network administrators to distribute the policy evenly across multiple switches. This can simplify the 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, routers, which are the data plane or forwarding plane. By using the Southbound APIs, administrators can adjust the network according to the change requirements.

B. 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 OpenFlow Switch specification is version 1.5.1. Each OpenFlow switch will store information of 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, transaction id (xid). OpenFlow message types can be, for example, FlowMod, PacketIn, FlowRemoved, PacketOut, and StatsReq. The FlowMod message is used to allow the controller to modify 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 that 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.

C. Ryu

Ryu [6] is a framework that implements the OpenFlow controller and is developed in Python. Once Ryu controller receives an OpenFlow message from the switch, it will trigger an event. An event handler contains an event class and the state of 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, EventOFPPacketIn event class will be called. For example, when using 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].

D. Related Works

Here we surveyed related works that utilize Snort IDS [8] and SDN together to dynamically filter anomaly traffic.

K. Nam and K. Kim studied on SDN security enhancement using open source IDS/IPS Suricata [9]. They proposed requirements for implementing SDN security. They mentioned whether the SDN solutions, Suricata IDS/IPS, automated intrusion prevention, and mirroring are necessary or not, when we want to implement a firewall, network scan detection, abnormal traffic detection, intrusion detection and intrusion prevention. They concluded that to implement the intrusion prevention, because the SDN alone cannot handle this, we need Suricata IDS/IPS, automated intrusion prevention, and also the mirroring. However, no implementation and its performance were shown in their paper.

H. 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 [10]. The throughput, delay and packet loss, which are required performance metrics of quality of service (QoS), are observed. Their experiment results on the mininet virtual network [11] showed that integrating SDN with Bro IDS yielded in the better performance than using Snort IDS in all 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 that of Bro.

R. Sutton, et al. designed and developed a system that utilized an SDN to divert traffic of some suspicious traffic multiple Snort IDS machines for inspection [12]. 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 generate a REST API command that is then sent to the SDN controller. For the experiments, they implemented on a virtualized network using GNS3, which is the graphical version of Network Simulator-3, and used vSwitch. However, the latency was not measured.

E. Existing SDN and IDS Integration Approach

In firewall systems cooperating with IDS and SDN, the IDS generally does 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 Common Event Format (CEF) or Log Event Extended Format (LEEF) to facilitate analysis. However, similar to what proposed in [12], in case of a firewall system recording alert messages into a log file and using OpenFlow as a SDN platform, typical behavior of the firewall system to perform blocking malicious packets consists of the following six steps, as illustrated in Figure 1.

(1) When the OpenFlow switch receives packets from the client that is 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 found 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, they have to use log monitoring tools to monitor the log change and use a command line tool such as “curl” for transferring REST APIs. Therefore, a processing overhead will occur. Moreover, REST APIs itself use HTTP methods and run over TCP protocol.

III. PROPOSED WORK

For our proposed work, as illustrated in Figure 2, there are five steps to perform.

(1) When the OpenFlow switch receives packets from the client that is destined to the server, it will delay the forwarding process to the destination and copy the traffic to IDS via port mirroring.

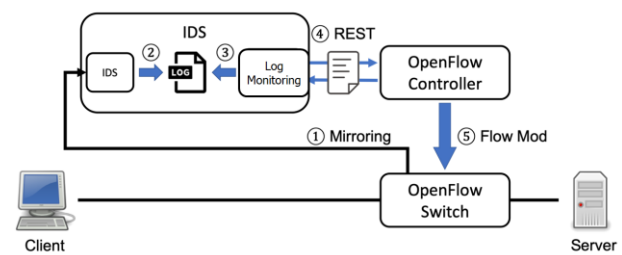


Figure 1. System architecture of the existing approach

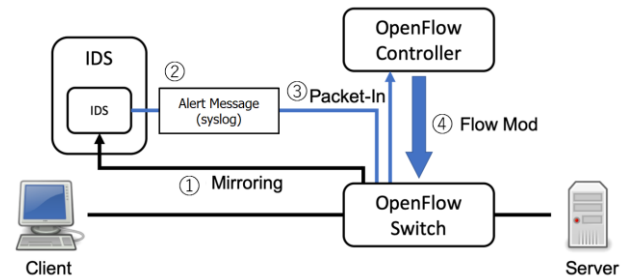


Figure 2. System architecture of the proposed work

(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, syslog message or 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 that 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 our proposed method can reduce one step when comparing to the existing method mentioned in Section II D, 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.

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 Snort which is a famous Open Source Software (OSS). Snort has a big community and has more than thousands of detection signatures available. Table 1 summarizes software and its version used in our experiments.

Moreover, we use four Raspberry Pi 2 boards to emulate each device. First, one Raspberry Pi is used for implementing both OpenFlow controller and OpenFlow switch, while another Raspberry Pi is used for running Snort IDS. Other two Raspberry Pi boards are used as a client and a server.

TABLE 1. SOFTWARE SPECIFICATION

Device	Software
OpenFlow controller	Ryu 2.7
OpenFlow switch	Open vSwitch 2.3.0 (OVS)
Intrusion Detection System	Snort 2.9.9
Log monitoring tools	swatch
DoS attacking tools	hping3

In our implementation, we implement three approaches: (1) the existing method, (2) another comparison method, and (3) our proposed method. Details of them are described as below.

A. Implementation of existing approach

First, we implement the existing approach that uses Snort to write alerts to the log file and uses swatch utility to monitor the log file. As shown in Figure 3, in this approach Snort and swatch are running on the same Raspberry Pi board. Moreover, the Raspberry Pi that emulates both Ryu controller and Open vSwitch (OVS) equips with four ports: Port 1 connecting to the client, Port 2 connecting to the server, Port 3 is one-way directional port just for forwarding packets to Snort to analyze, while Port 4 is also 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 another comparison method

Other than implementing the existing method and our proposed method, we also implement another comparison method. In this method, we just change the way that swatch informs the alert to the controller. As illustrated in Figure 4, instead of using REST API and sending the information directly to the Ryu controller, in this approach we just send the information as UDP packets in order to cut off the latency caused by TCP 3-way handshake. Therefore, instead of having Port 4 attached to Ryu controller, in this implementation Port 4 is attached to OVS and is used for receiving alerts in UDP packets from swatch utility to the OVS. Then, the alert message will be transferred from OVS to Ryu as a PacketIn message, similar to our proposed work.

C. Implementation of our proposed work

Figure 5 shows the implementation of our proposed work. Different from Figure 3, in this implementation Port 4 is used for receiving alerts from Snort via rsyslog messages over UDP packets. Then, the OpenFlow switch will forward this alert as PacketIn to the Ryu controller. Snort IDS has no need to write the alert in the log file. This technique can help reduce the latency time in writing and monitoring the log file. Moreover, by using rsyslog messages over UDP and PacketIn messages, our approach can also cut off the latency occurring by the TCP 3-way handshake.

Figure 6 shows the flow chart of processes inside our Ryu controller. Once the OVS connects with the Ryu controller, it will initialize the flow table as shown in Table 2 and set the

state of OVS as the normal state. When the Ryu controller receives the PacketIn message, it will check whether the message is a syslog message or not. If yes, it will analyze the message and modify the flow table. Example of a flow table that is modified after receiving DDoS attacks is as shown in Table 3. 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.

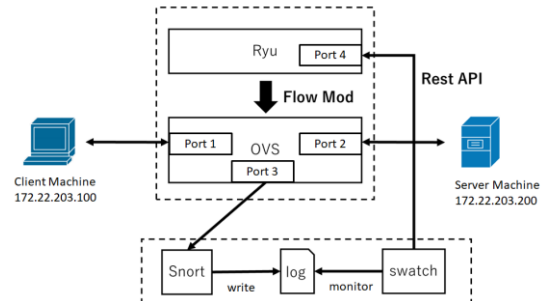
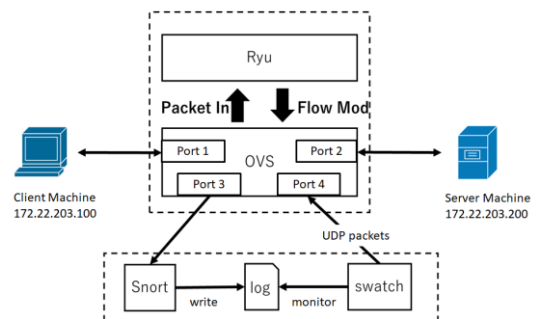
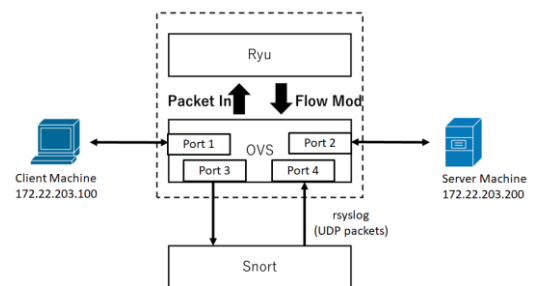
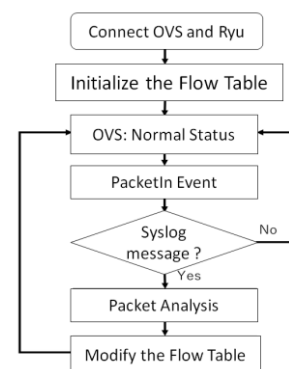
**Figure 3.** Implementation of existing approach**Figure 4.** Implementation of another comparison method**Figure 5.** Implementation of our proposed work**Figure 6.** FlowChart of process inside Ryu controller

TABLE 2. 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 3. 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 172.22.203.100 ip_dst 172.22.203.200 tcp_dst 80 eth_type = 0x0800	Drop

TABLE 4. PROCESSING TIME OF EACH APPROACH

Method	Attack Detection (ms)	Route management (ms)	Total (ms)
Existing approach	563.5	17.2	580.8
Comparison approach	484.0	12.6	496.6
Proposed Work	397.9	12.0	410.0

V. EVALUATION RESULTS

For the evaluation, we compare our proposed work with the existing approach and the comparison approach described in Section IV A. and Section IV B., respectively. To measure the processing time used for detecting attacks and time used for route management, we observe the packet timestamp by using the Wireshark program.

For the existing approach, the attack detection time is measured from the time that the OVS performs port mirroring to Snort IDS until the time that swatch utility sends the REST API packet to Ryu. On the other hand, the route management time is measured from the time that the swatch utility sends a packet to inform Ryu until the time that the FlowMod message is sent from Ryu to OVS.

Next, for the comparison method, the attack detection time is measured from the time that OVS performs port mirroring to Snort IDS until the time that the swatch utility sends a Socket API packet to OVS. On the other hand, the route management time is measured from the time that the swatch utility sends a packet to inform Ryu until the time that the FlowMod message is sent from Ryu to OVS.

Finally, for the proposed approach, the attack detection time is measured from the time that OVS performs port mirroring to Snort IDS until the time that Snort sends the rsyslog packet to OVS. On the other hand, the route management time is measured from the time that Snort sends the rsyslog packet to OVS until the time that the FlowMod message is sent from Ryu to OVS.

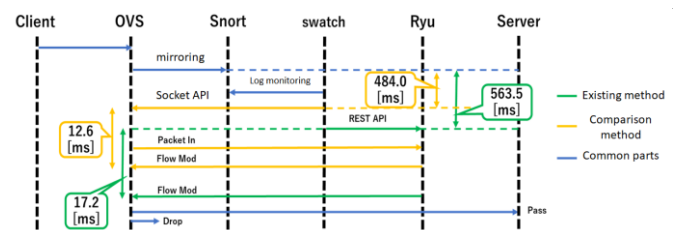
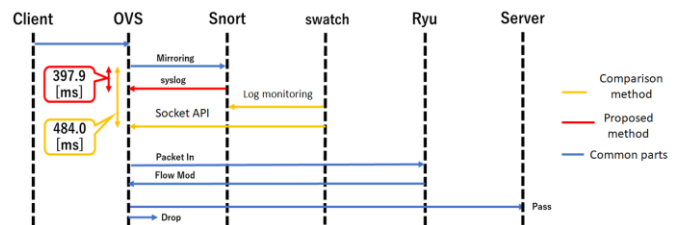
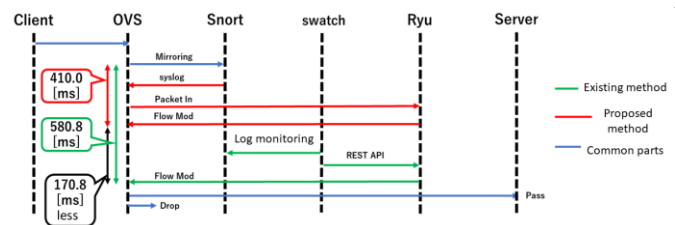
**Figure 7.** Comparison of the processing time between the existing approach and the comparison approach**Figure 8.** Comparison of the processing time between the comparison approach and the proposed approach**Figure 9.** Comparison of the processing time between the existing approach and the proposed approach.

Table 4 summarizes the processing time for detecting attacks and for route management. Each value is the average from five times of measurement. Compared to the existing approach, the comparison approach that uses UDP SOCKET API message instead of using the curl command to send a REST API message can reduce up to 79.5 ms for the attack detection time. Moreover, the route management time of the comparison approach is 4.6 ms better than that of the existing approach. This is because the REST API requires the TCP 3-way handshake at the beginning of the connection. Therefore, using UDP with PacketIn results in a shorter processing time than using the REST API message.

Figure 7 illustrates the comparison of the route management time and the total processing time between the existing approach and the comparison approach.

Next, as in Figure 8, compared to the comparison work, our proposed work can reduce the route management time for 86.1 ms. This is because in the comparison work, the swatch utility has to monitor the log file all the time; and once it detects a change in the log time, the swatch utility will generate a UDP packet to send to OVS. This process creates an overhead comparing to our proposed approach that uses a syslog message sent directly from Snort to OVS. Therefore, comparing to the comparison approach, we can reduce the overhead in writing a log file and monitoring the log file.

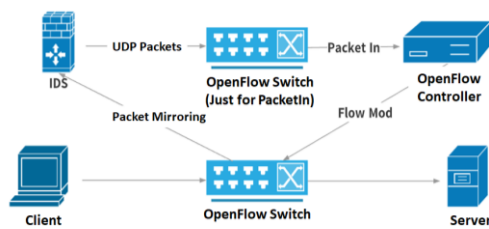


Figure 10. Using multiple OpenFlow switches to perform the load balancing in the IDS and SDN integration network.

Finally, Figure 9 illustrates the comparison of the total processing time between the existing approach and our proposed approach. We found that our approach can reduce up to 170.8 ms in total. Thus, we can conclude that by using the rsyslog with the PacketIn message instead of using the log monitoring tool with the REST API can accelerate the blocking process at our SDN switches and can prevent the malicious traffic to our network faster.

VI. CONCLUSIONS

Integrating IDS and SDN can help the network administrator to quickly block malicious traffic coming to the organization. In this paper, instead of using the log monitoring and the REST API to feed alerts from IDS to the SDN controller, we proposed our approach that uses syslog and PacketIn messages instead. From our evaluation experiments, we could confirm that our proposed method can efficiently reduce the total processing time occurring since the packet mirroring from IDS until the routing policy is propagated from the SDN controller to the switch. However, in our experiments we used the old Raspberry Pi 2 and the Snort open source. If we use a higher performance machine and IDS system/hardware, the total processing time may reduce much than reported in this paper. Furthermore, in our experiments we used only one OpenFlow switch. In the future, if we consider a large organization network, forcing an OpenFlow switch to perform both normal packet forwarding and forwarding PacketIn messages may give the switch more workloads and regrade the forwarding capability of the switch. As in Figure 10, we may provide another OpenFlow switch that is used just for forwarding the PacketIn message from IDS to the controller. Moreover, we can have multiple IDS servers in order to perform load balancing among them as well.

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, Volume 38, Issue 2, pp 69–74, April 2008, DOI: 10.1145/1355734.1355746.
- [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", 2014 International Conference and Workshop on the Network of the Future (NOF), pp. 1-5, 2014, DOI: 10.1109/NOF.2014.7119792.
- [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] H. Ikarashi, Y. Jin, N. Yamai, N. Kitagawa and Kiyohiko Okayama, "Design and Implementation of SDN-based Proactive Firewall System in Collaboration with Domain Name Resolution", IEICE Transactions on Information and Systems, Vol.E101-D, No.11, pp.2633-2643, November 2018, DOI: 10.1587/transinf.2017ICP0014.
- [5] The Open Networking Foundation, "SDN Technical Specifications". <https://opennetworking.org/software-defined-standards/specifications/>
- [6] Ryu SDN Framework. <https://ryu-sdn.org/>
- [7] RYU the Network Operating System (NOS), "Ryu application API". https://ryu.readthedocs.io/en/latest/ryu_app_api.html
- [8] Snort.org. <https://snort.org/>
- [9] K. Nam and K. Kim, "A Study on SDN security enhancement using open source IDS/IPS Suricata," 2018 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 2018, pp. 1124-1126.
- [10] 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," 2019 7th International Conference on Information and Communication Technology (ICoICT), Kuala Lumpur, Malaysia, 2019, pp. 1-7.
- [11] Mininet. <https://opennetworking.org/mininet/>
- [12] R. Sutton, R. Ludwiniak, N. Pitropakis, C. Chrysoulas and T. Dagiuklas, "Towards an SDN Assisted IDS," 2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 2021, pp. 1-5.



Yusei Katsura received B. Info. Env. from Tokyo Denki University, Japan in 2019. He was a research student at Tokyo University of Agriculture and Technology, Japan in 2019-2020. He is currently a Master student at Nara Institute of Science and Technology. His research interest is computer network.



Pranpariya Sakarin received B.Sc. from Faculty of ICT, Mahidol University, Thailand in 2019. She has worked for Refinitiv Software (Thailand) company limited as a Technical Specialist since 2020. Her research interest is computer network.



Nariyoshi Yamai received the B.E. and M.E. degrees in electronic engineering and the Ph.D. degree in information and computer science from Osaka University in 1984, 1986, and 1993, respectively. After working at Nara National College of Technology, Osaka University, and Okayama University, he has been a Professor at the Institute of Engineering, Tokyo University of Agriculture and Technology since April 2014. His research interests include distributed systems, network architecture, and Internet. He is a member of IEEE CS, IEEE ComSoc, IEICE and IPSJ.



Hiroyuki Kimiyama is a Professor in Daido University, Japan. He received B.E. and M.E. from Tohoku University, Japan, in 1988 and 1990, respectively. In 2010, he received Ph.D. from the graduate school information system, the University of Electro-Communications, Japan. He joined NTT corporation in 1990 and became a professor at Tokyo Denki University in 2016. Since 2019, he become a professor at School of Informatics Department of Information Systems, Daido university, Japan. His research interest is high-speed real-time distributed processing. He is a member of ACM, IPSJ and IEICE.



Vasaka Visootviseth is an associate professor at Mahidol University, Thailand. She received her Ph.D. degree in computer engineering from Nara Institute of Science and Technology (NAIST), Japan in 2003, M.E. and B.E. degree from Tokyo University of Agriculture and Technology (TUAT), Japan in 1999 and 1997, respectively. Her current research interests are mobile and wireless computing, Internet traffic measurement, and network security. She is a member of IEEE since 1998.