PAPER Special Section on Information and Communication System Security

Design and Implementation of SDN-Based Proactive Firewall System in Collaboration with Domain Name Resolution*

Hiroya IKARASHI[†a)], ***Nonmember***, **Yong JIN**[††b)], **Nariyoshi YAMAI**[†c)], ***Members***, **Naoya KITAGAWA**[†d)], ***Nonmember***, ***and*** **Kiyohiko OKAYAMA**[†††e)], ***Member***

SUMMARY Security facilities such as firewall system and IDS/IPS (Intrusion Detection System/Intrusion Prevention System) have become fundamental solutions against cyber threats. With the rapid change of cyber attack tactics, detail investigations like DPI (Deep Packet Inspection) and SPI (Stateful Packet Inspection) for incoming traffic become necessary while they also cause the decrease of network throughput. In this paper, we propose an SDN (Software Defined Network) - based proactive firewall system in collaboration with domain name resolution to solve the problem. The system consists of two firewall units (lightweight and normal) and a proper one will be assigned for checking the client of incoming traffic by the collaboration of SDN controller and internal authoritative DNS server. The internal authoritative DNS server obtains the client IP address using EDNS (Extension Mechanisms for DNS) Client Subnet Option from the external DNS full resolver during the name resolution stage and notifies the client IP address to the SDN controller. By checking the client IP address on the whitelist and blacklist, the SDN controller assigns a proper firewall unit for investigating the incoming traffic from the client. Consequently, the incoming traffic from a trusted client will be directed to the lightweight firewall unit while from others to the normal firewall unit. As a result, the incoming traffic can be distributed properly to the firewall units and the congestion can be mitigated. We implemented a prototype system and evaluated its performance in a local experimental network. Based on the results, we confirmed that the prototype system presented expected features and acceptable performance when there was no flooding attack. We also confirmed that the prototype system showed better performance than conventional firewall system under ICMP flooding attack.

key words: *firewall system, DNS, domain name resolution, EDNS, client subnet option, SDN, OpenFlow*

1. Introduction

Nowadays the cyber attacks happen every second and the number of cyber attacks continuously increasing. In order to survive the attacks, many organizations introduce security facilities such as firewall system, UTM (Unified Threat Management) system and IDS (Intrusion Detection System)/IPS (Intrusion Prevention System). These security facilities are usually deployed at the border of network and the administrators set security policies on them. As the cyber attack tactics are becoming more tricky, detailed investigations like Stateful Packet Inspection (SPI) [2] and Deep Packet Inspection (DPI) [3] for the incoming traffic become necessary on the security facilities. These detail inspections cause high workload on the security facilities and consequently decrease the network throughput. Therefore, in order to keep the network throughput high, network administrators have to give up using those security facilities or avoid the heavy inspections for the incoming traffic. Another critical issue in the real operation is the burden of configuration and update for the policies on the security facilities. Network administrators usually manually configure and update the security policies using the layer 3 or 4 information and only those pre-defined communications can be controlled. For example, if a firewall system refers to an external blacklist, the network administrator has to configure and update its security policies manually whenever the blacklist is updated.

In this paper, we propose an SDN-based proactive firewall system in collaboration with the domain name resolution. The firewall system consists of two firewall units and a proper one will be selected adaptively for investigating every incoming traffic based on the collaboration of SDN controller and the internal authoritative DNS server. Specifically, the client IP address of an incoming traffic will be obtained in advance by using EDNS Client Subnet Option [4] of DNS protocol during the name resolution stage and the SDN controller checks the client IP address on the whitelist and blacklist. As a result, the incoming traffic from the trusted clients will be directed to the lightweight firewall unit in order to avoid heavy processes while those from the unknown clients will be directed to the normal firewall unit which conducts detail investigations. As well known, an SDN controller can obtain the source IP addresses of incoming traffic without collaboration of DNS. However, the SDN controller can be easily the bottleneck in an SDN based system thus additional feature for obtaining and checking the source IP addresses will increase the workload of SDN controller much more. By collaboration with DNS, the incoming traffic with no preliminary name resolution will be dropped at the SDN switch without triggering packet_in events thus this type of traffic will not increase the workload of the SDN controller. Moreover, DNS infrastructure has

Manuscript received November 14, 2017.
Manuscript revised May 13, 2018.
Manuscript publicized August 22, 2018.

[†]The authors are with Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.
[††]The author is with Tokyo Institute of Technology, Tokyo, 152–8550 Japan.
[†††]The author is with Okayama University, Okayama-shi, 700–8530 Japan.

*This paper is a revised version of [1].

a) E-mail: hikarashi@net.cs.tuat.ac.jp
b) E-mail: yongj@gsic.titech.ac.jp
c) E-mail: nyamai@cc.tuat.ac.jp
d) E-mail: nakit@cc.tuat.ac.jp
e) E-mail: okayama@cc.okayama-u.ac.jp

DOI: 10.1587/transinf.2017ICP0014

been known as one of the largest distributed database systems over the Internet thus it is expectable to mitigate the concern of scalability issue by collaboration with DNS. In addition, network administrators can use the open whitelists and blacklists for checking the source IP addresses and do not need to configure the policies in every firewall system thus the administrative cost can be reduced. For example, "FireHOL" [5] is one of the open blacklists available for checking the malicious IP addresses. Finally, as we mentioned at the beginning, introduction of SPI and DPI will decrease the network throughput. In the proposed firewall system, the decrease of network throughput caused by security investigation can be mitigated by separating the incoming traffic initiated by the trustable clients and that by unknown or malicious clients.

It should be noticed that the main purpose of this research is to solve the existing issues using EDNS Client Subnet Option. Therefore the current status of EDNS support on the external DNS cache servers in the real network environment is beyond the scope of this paper. In fact, the networks with EDNS supported DNS full resolver can achieve benefits from our proposed system thus the deployment in the entire Internet can be in step by step. The contributions of this paper can be summarized as the following points: 1) heavy inspections are applicable for incoming traffic from suspicious clients without network throughput reduction; 2) administrative cost for security policy deployment can be significantly reduced; 3) the effectiveness of SDN-based firewall system has been confirmed for real network operation.

2. Existing Technologies and Related Work

2.1 Firewall System

Firewall system is an important security solution for all networks and so far various versions have been released [6], [7]. Basically, firewall system passes through legitimate traffic and blocks malicious packets at the border of a network. In order to be applicable for tricky cyber attacks, recent firewall systems also have complicated functions such as Web Application Firewall (WAF), host inspector and access controller, etc. By using these functions, each packet can be checked precisely and can be handled appropriately based on the results. However, the inspection also consumes much resource of the firewall system and cuts down network throughput. Moreover, network administrators need to configure and update the security policies of the firewall system in time manually based on the information source such as whitelist and blacklist. This burden not only increases the operational cost but also impacts the performance of firewall system in terms of real time inspection in case of update delay for the latest security policies. Although there are many ongoing researches for improving firewall system performance [8], [9], they are still insufficient especially against Distributed Denial of Service (DDoS) attacks.

2.2 SDN Technology and OpenFlow Protocol

Dynamic network architecture with multiple switches for traffic control has been proposed and heartily discussed from 1990s [10] and it influenced some core ideas of SDN technology. In recent years, with the growth of the demand for large scale and flexible network architecture, the requirement of SDN technology becomes increasing. For example, SDN has become a key technology to operate NFV (Network Function Virtualization) [11], [12]. Basically, an SDN based network consists of controllers and switches. As a popular implementation of SDN technology, OpenFlow protocol [13] is well used. In OpenFlow protocol, each packet passing through the switches is controlled (i.e. forward, drop and store, etc.) based on the flow entry maintained in each switch and each flow entry consists of rule, action and statistics. For the rule of a flow entry, 12 or more parameters indicating the information from layer 1 (physical layer) to layer 4 (transport layer) with the combination of the actions can be used and the number of available parameters depends on the version of OpenFlow protocol. OpenFlow protocol is well used for network traffic control [14], server load balancing [15] and security purpose. In addition, in the literature, SDN technology was also confirmed to be applicable for both physical and virtual network environment [16].

2.3 Related Work

Since we focus on SDN-based firewall system in this paper, we mainly discuss the related work that practically uses SDN technology for security purposes.

Nugraha et al. [17] proposed a method to detect and mitigate SYN Flooding attack using OpenFlow technology and sFlow methodology. In the method, incoming traffic will be analyzed by sFlow collector of each OpenFlow switch and when the packet rate exceeds the indicated threshold the OpenFlow switch will be notified and the corresponding flow table will be updated accordingly to block the traffic. Accordingly, the administrator needs to maintain the parameters for each OpenFlow switch based on the characteristic of every attack and its applicable target is also limited.

Yoon et al. [18] implemented four types of security functions with SDN including "firewall and IPS", "IDS", "scan and DDoS detector" and "stateful firewall and reflector networks", etc. In the firewall function, the SDN controller checks the packet_in message in the firewall application and updates the flow table of the SDN switch based on the check results. In this method, the administrator needs to maintain the security policies in the firewall application and all the investigation will be conducted by the SDN controller after the packet_in event has been triggered. Therefore, the SDN controller can be the bottleneck of the system which may cause throughput decrease and the administrative cost will also be increased.

Kim et al. [19] proposed a method for preventing DNS

Table 1 Brief comparison between existing solutions and the proposal.

Solutions	Applicable targets	Controller based investigation	Administrative cost
[17]	SYN Flooding attacks	No (switch based)	High
[18]	Manually configured	Yes	High
[19]	DNS amplification attack	Yes	High
[20]	Anomaly detection	Yes	High
Proposed method	whitelist and blacklist basis	No (DNS basis)	Low

amplification attacks using the history of DNS queries with SDN. In the method, all the DNS queries sent out will be stored in the SDN switch or SDN controller and only the DNS responses match to the stored DNS queries will be passed through. This approach needs to add the new features for storing the DNS queries sent out and checking all the incoming DNS responses to all SDN switches and the applicable target is also limited.

Tang et al. [20] proposed a deep learning approach for network intrusion detection in SDN environment. In the method, a deep learning module installed in the SDN controller analyzes and detects the network intrusion using the traffic statistic information sent from every SDN switch with a configured interval. Based on the analysis results, the SDN controller updates the flow table of each SDN switch. This approach requires high computational capability in the SDN controller and basically the detection will be conducted after the network traffic has been passed through.

We briefly compared the characteristics between the existing solutions and our proposed method. As shown in Table 1, in the conventional methods, investigations are mainly conducted by SND controller which may decrease network throughput and the security policies are maintained manually which may cause high administrative cost. In our research, we propose a proactive firewall solution with general security purpose based on whitelist and blacklist considering network throughput and low administrative cost on security policy as well as the deployment.

3. SDN-Based Proactive Firewall System in Collaboration with Domain Name Resolution

3.1 Design

In this research, we consider a target network topology consisting of a SDN-based internal network and the Internet. The internal network includes SDN controller, SDN switches, firewall systems, application servers as well as DNS servers and the traffic between the internal network and the Internet can be controlled by the SDN controller. Figure 1 shows a simple example of the target network topology. In general, a network has one firewall system set at the border in order to check and control all the incoming and outgoing network traffic. However, as can be seen from

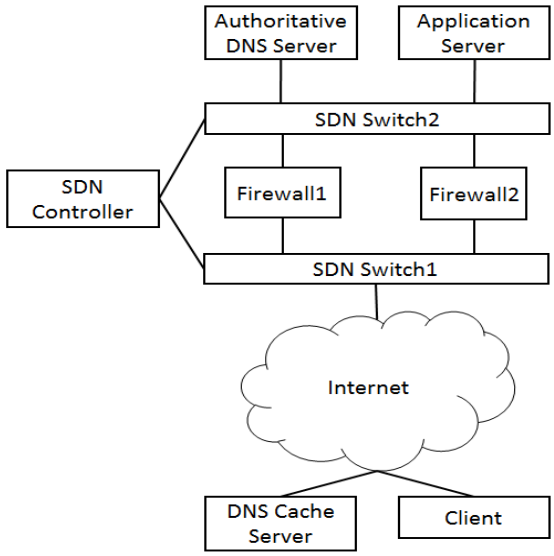

Fig. 1 An example of our target network topology

Fig. 1, there are two firewall units (Firewall1 and 2) set between the SDN switches (SDN Switch1 and 2). This is because the proposed system proactively forwards network traffic to different firewall unit for corresponding check in order to provide high performance. The "Authoritative DNS Server" provides authoritative name resolution service for the domain name which is used in the "Application server" and the "DNS Cache Server" provides recursive name resolution service to the client in the Internet. As a result, all the incoming and outgoing traffic of the internal network can be controlled by the SDN controller and switches.

The key idea of our proposed system is to forward incoming traffic to different firewall unit based on the subnet information of the client IP address which can be obtained by the collaboration of domain name resolution using EDNS Client Subnet Option. Specifically, when the external DNS cache server sends DNS query to the internal authoritative DNS server on behalf of the external client, the external DNS cache server notifies the subnet information of the client IP address to the internal authoritative DNS server. Here, we assume that the external client and DNS cache server are in the same subnet which is common in almost all networks. Then the internal authoritative DNS server sends the client subnet information to the SDN controller for checking in the whitelist and blacklist and based on the results the SDN controller updates the flow table of the SDN Switch1 to forward the traffic to the proper firewall unit. Recently, public DNS full resolvers such as Google Public DNS are well used in some networks and in these cases the external client may have different subnet from the DNS full resolver. However, what our proposed system needs is that the public DNS full resolver supports EDNS Client Subnet Option and this option was originally proposed by Google Public DNS. Thus with the increase of EDNS Client Subnet support in public DNS full resolvers, more networks can achieve benefits from our proposed firewall system.

Since almost all Internet users and service providers use DNS based domain name resolution service, every access to the internal application server from the external clients can be controlled by using the proposed system. Note that among the two firewall units, one is for just lightweight check to the clients included in whitelist while the other is for detail investigation such as DPI (Deep Packet Investigation) to those unknown or listed on blacklist. Moreover, the SDN controller only needs to update the rule for a particular subnet once since the name resolution results can be cached in the external DNS cache server. Thus the same DNS cache server will not query the internal authoritative DNS server until the TTL (Time To Live) of the domain name resolution result expires and this avoids unnecessary control process.

3.2 Notification of the Client Subnet Information

In the proposed system, we use EDNS Client Subnet Option to obtain the client subnet information. The EDNS was introduced for notifying the client subnet information (subnet address and net mask of the client) thus it also can notify the entire client IP address by setting the net mask to 32 bits. During the name resolution, when the external DNS cache server obtains the IP address of the internal authoritative DNS server which is authoritative for the target domain name, the external DNS cache server sends a query to the authoritative DNS server with attaching the client subnet information. Accordingly the internal authoritative DNS server can achieve the client subnet information and then notify the SDN controller.

Cache function of the external DNS cache server can be an issue in the proposed firewall system. Once the external DNS cache server finishes the name resolution, it caches the obtained information and uses the cache for the next queries until the TTL expires. In general, the external DNS cache server is located at the same place with the client thus once the client subnet information is notified to the SDN controller then the cache function is not an issue in this case. However, in case of that multiple client segments exist in an organization and all of them use the same external DNS cache server, the external DNS cache server may reply for some queries from other segments using the cache. Therefore we intend to disable the cache function of the external DNS cache server partially for this case. Although this approach may increase the queries to the internal authoritative DNS server, we consider that only one extra query per client will not cause attack or affect the effectiveness of the proposed firewall system. The description about the approach can be found in [21] and we omit the detail in this paper.

3.3 Collaboration of SDN Controller and Internal Authoritative DNS Server

In our proposed system, the internal authoritative DNS server obtains the client subnet information during the name resolution and notifies it to the SDN controller. Basically, SDN controller runs in event basis thus it cannot monitor DNS traffic for obtaining the client subnet information. Accordingly, we add a sub SDN switch between the SDN controller and the authoritative DNS server, and use PacketIn method of SDN protocol to transfer the notification message from the authoritative DNS server to the SDN controller. Note that the SDN controller cannot check the client subnet information of all incoming traffic since it will be vulnerable to DDoS attacks. Thus in the proposed system, the investigation for the client subnet information on the SDN controller only occurs once the domain name resolution is finished.

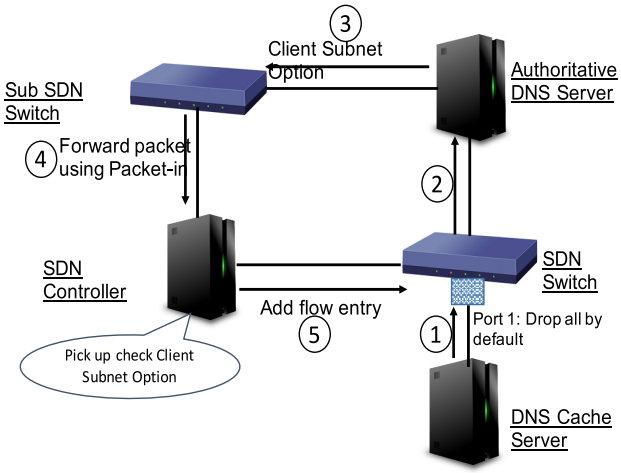

Fig. 2 Collaboration of SDN controller and DNS server

Figure 2 shows an example of network configuration for the collaboration between the SDN controller and the internal authoritative DNS server. The SDN Switch port directly connected to the external DNS Cache Server is set to "drop all by default" and only DNS packets are allowed to be forwarded to the internal authoritative DNS server. Note that the traffic between the SDN Controller and the internal Authoritative DNS Server uses the Sub SDN Switch. In fact, it is possible to use the SDN Switch directly instead of adding the Sub SDN Switch. However, considering that the Authoritative DNS Server may also need other communication such as Ping and Internet access which use the SDN Switch, thus for the simplicity we added the Sub SDN Switch between the Authoritative DNS Server and the SDN Controller only for the notification of client subnet information.

We also consider that the external DNS cache server can possibly attach fake client subnet information on its DNS query and the SDN controller may accidentally updates the flow table of the SDN switch to allow the malicious accesses. However, this kind of forged IP addresses can be filtered by other security functions such as ingress filtering [22] thus we do not consider the feature in the proposed firewall system.

3.4 Procedure of the Proposed Firewall System

Based on the design, we describe the detail procedure of the

proposed firewall system using an example that an external client accesses the internal application server from the start of the domain name resolution in the following.

1. In order to access the internal application server, the external client needs to know the IP address of the internal application server. Therefore, the external client sends a recursive name resolution request to its local DNS cache server (DNS Cache Server in Fig. 1).
2. The external DNS cache server sends the DNS queries on behalf of the external client. Starting from the root DNS server, eventually the external DNS cache server reaches to the authoritative DNS server corresponding for the target domain name (Authoritative DNS Server in Fig. 1) and completes the name resolution.
3. After the name resolution is finished, the external DNS cache server sends the same DNS query with attaching the client subnet information to the internal authoritative DNS server directly. Note that the notification can be performed from the first query but considering security and privacy we decide to notify it only to the authoritative DNS server of the target domain name.
4. When receives the DNS query from the external DNS cache server, the internal authoritative DNS server checks whether the client subnet information is included. If it is, the internal authoritative DNS server notifies the client subnet information to the SDN controller, otherwise replies answers to the external DNS cache server and the external DNS cache server replies back the answer to the client.
5. The SDN controller checks the client subnet information received from the internal authoritative DNS server on the whitelist and blacklist stored on its own. Based on the check result, the SDN controller updates the flow table of the SDN Switch (not Sub SDN Switch). That is, if the client subnet information is included in the whitelist, the SDN controller updates the flow table of the SDN switch to forward the incoming packet from the client to the Firewall1 which only performs lightweight checks; otherwise the SDN controller makes the SDN switch forward the incoming packet from the client to the Firewall2 which performs detailed investigations.
6. Finally, the external client starts access to the internal application server and the traffic will be handled by the SDN switch based on the policies set in the step 5.

With the above procedure, the incoming traffic from the external client can be controlled by the SDN network and also can be investigated flexibility based on the client subnet information on an appropriate firewall unit. In the proposal, we adopt IP address based investigation using whitelist and blacklist which has the risk of IP address spoofing attack. Basically, IP address spoofing attack is difficult in TCP communication since it needs session hijack, while for UDP communication, is common security issue. As a solution, we can consider to add the IP spoofing investigation feature into firewall systems and we omit the detail in this paper.

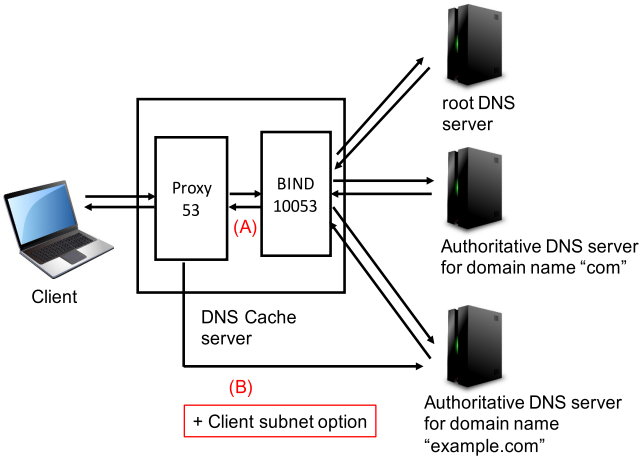

Fig. 3 Domain name resolution and client subnet option notification.

4. Implementation of Prototype System

Based on the design of our proposed system, we implemented a prototype system using OpenFlow architecture and DNS protocol. Since real firewall systems are too expensive to be used for academic experiments, we used two application servers instead for confirming the traffic control function and switching performance. In this section, we describe the detail of domain name resolution part and OpenFlow network configuration in the prototype system.

4.1 DNS Servers and Client Subnet Notification

For the sake of simplicity, instead of directly customizing DNS server program, we implemented the external DNS cache server by combining a DNS proxy and BIND (Berkeley Internet Name Domain) [24] which are running on port 53 and 10053 respectively. The DNS proxy consists of two Perl modules “Net::DNSServer” and “Net::DNSServer::Proxy” [23]. Figure 3 shows the configuration of the domain name resolution part of the prototype system. In the external DNS cache server, when the DNS proxy receives a query from the client, it forwards the query to the BIND for recursive name resolution. By receiving the DNS response from the BIND (A in Fig. 3), the DNS proxy can obtain the IP address of the authoritative DNS server of the target domain name. Then the proxy initializes another same query with attaching the client subnet information to the authoritative DNS server directly which eventually performs the client subnet information notification (B in Fig. 3). Note that although BIND supports EDNS Client Subnet Option there is no feature of attaching the client subnet information only on the query to the authoritative DNS server of the target domain name. As we described, considering security and privacy the client subnet information will be only notified to the target authoritative DNS server.

For the implementation of the authoritative DNS server (for domain name “example.com” in Fig. 3), we used Perl

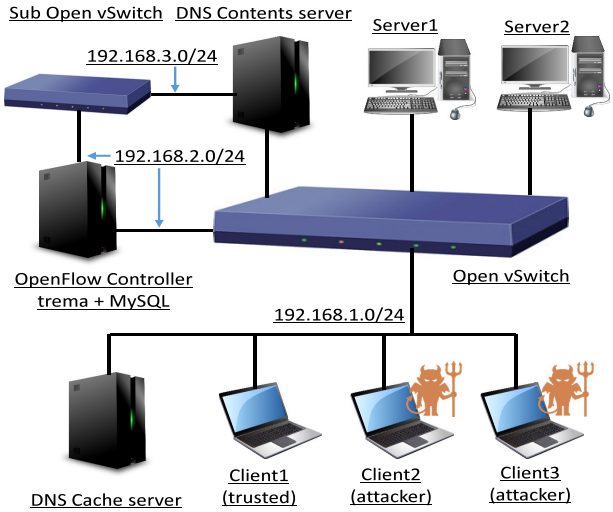

Fig. 4 Configuration of the local experimental network.

module "Net::DNS::Nameserver" [25] by listening on port 53. When receives a DNS query for A records of the domain name "www.example.com", the authoritative DNS server checks if the DNS query has client subnet option attached. If it does, the authoritative DNS server do both of replying the query and registering the client subnet information to the database running on the SDN controller. Otherwise, the authoritative DNS server only answers for the DNS query. For the database software, we used MySQL[†].

Finally, we used UDP (User Datagram Protocol) packet rather than TCP (Transmission Control Protocol) connection for the client subnet option notification in order to simplify the data transmission using the PacketIn method in the OpenFlow network. Therefore in the implementation of the prototype system we used the exact client IP address (the client subnet with 32-bit net mask) as the client subnet information.

4.2 OpenFlow Architecture Based Experimental Network

We constructed a local experimental SDN network using OpenFlow protocol version 1.0. Trema [26] and Open vSwitch [27] were used for the OpenFlow controller and OpenFlow switches respectively. The network configuration is shown in Fig. 4. The application servers "Server1" and "Server 2" play the roles of firewall units instead of real firewall systems. We consider that the main purpose of the proposed system is to mitigate the throughput reduction caused by the traffic congestion on a specific firewall system thus the "Server1" and "Server 2" are capable enough to simulate the performance of the proposed firewall system. Among the clients, the "Client1" is assumed to be trustable while others are unknown or malicious. For the network configuration, the application servers, clients, authoritative DNS server and DNS cache server are connected to the same Open vSwitch using the subnet 192.168.1.0/24 while the OpenFlow controller, Open vSwitch and sub Open vSwitch are connected using another subnet 192.168.2.0/24. On the other hand, the authoritative DNS server and sub Open vSwitch are configured using a third subnet 192.168.3.0/24 for their independent communication (registration of the client subnet information from the authoritative DNS server to the OpenFlow controller). Considering that the network traffic to the firewall system will not be high, we configured the links to the application servers to 100Mbps while for others to 1Gbps. The specs of each component are listed in Table 2.

[†]MySQL - https://www-jp.mysql.com/

Table 2 The specs of each component.

Component	CPU/Main Memory	OS
Client1	Core i5-4z 2.60GHz/8GB	Windows10 Pro
Client2,3	Core 2 Duo 2.66GHz/2GB	Debian GNU 8.3.0
Server1,2	Core i5-4210M 2.60GHz/8GB	Debian GNU 8.3.0
DNS servers	Core 2 Duo 2.66GHz/2GB	Debian GNU 8.3.0
OpenFlow controller	Xeon E31245 3.30GHz	Debian GNU 8.3.0
	8GB	
Open vSwitch	Pentinum 1403v2 2.60GHz/2GB	Debian GNU 8.3.0
Sub Open vSwitch	Core 2 Duo 2.66GHz	Debian GNU 8.3.0
	2GB	

5. Evaluations and Results

We evaluated the prototype system using the OpenFlow architecture based experimental network. In prior to the detailed evaluation, we describe the pre-configuration of the prototype system in the following.

When the OpenFlow controller is initialized, it deploys the pre-configuration to the Open vSwitches. Table 3 shows the flow entries in the pre-configuration which will be added to the Open vSwitches when the OpenFlow controller receives "switch ready" event from them. The "switch ready" event occurs when the Open vSwitch is successfully authenticated by the OpenFlow controller. We used the parameters of IP packet such as source and destination IP addresses, port numbers and IP protocol version in the pre-configuration. By default, all connections to the "Port1" which is the entry to the internal network are denied except those for operating the proposed firewall system itself. Therefore the internal communications (flows 1 and 2), name resolution related traffic (flows 3, 4 and 5) will be added by the network administrator in advance. Note that only flow 5 will be added to the sub Open vSwitch and others will be added to the main Open vSwitch. Here, the OpenFlow controller identifies the Open vSwitches using 64-bit DPID (Data path ID) which is defined by the network administrator. Other traffic from the external network is defined by default in the flow 6 with lowest priority as "drop" and especially ARP (Address Resolution Protocol) packets are defined as flood with the highest priority in the flow 7.

5.1 Feature Evaluation

In the feature evaluation, we considered a scenario that all

Table 3 Pre-configured flow entries of the Open vSwitch.

Flow No.	From	To	Action	Priority
1, 2	Server1, 2	Port1	Send out	1
3	DNS Client	DNS Server	Send out	2
4	DNS Server	DNS Client	Send out	2
5	DNS Server	Any	PacketIn	1
6	Other in Port1	Any	Drop	0
7	(ARP from any)	Any	Flood	MAX

```
[line1:] cookie=0xc,duration=150.112s,table=0,
    n_packets=0,n_bytes=0,idle_age=150,priority
    =1, ip,in_port=1,nw_src=192.168.1.1,nw_dst
    =192.168.1.4, actions=output:2
[line2:] cookie=0xd, duration=10.279s, table=0,
    n_packets=0, n_bytes=0,idle_age=10, priority
    =1,ip,in_port=1,nw_src=192.168.1.2,nw_dst
    =192.168.1.4,actions=mod_nw_dst:192.168.1.5,
    mod_dl_dst:00:30:1b:bc:c6:e1,output:3
```

Fig. 5 Flow entries created based on the traffic from the clients.

the three clients attempt to access the server1 and only the client1 will success while others will be forwarded to the server2 by the OpenFlow controller. As we mentioned, only the client1 plays as a trusted client while the other two play as unknown or malicious.

As the detail procedure, first, we started the OpenFlow controller, Open vSwitches including the sub Open vSwitch, the authoritative DNS server, the DNS cache server and all clients. These startup operations can be in any order. Next, we checked the OpenFlow controller received "switch ready" events from the Open vSwitches and confirmed that they were connected successfully. At this phase, the PacketIn method of running in the main Open vSwitch only handles the packets from the authoritative DNS server since other packets will be dropped by default based on the pre-configured flow entries. The OpenFlow controller also checks the source and destination IP addresses, DPID and communication protocol when receiving the PacketIn message in order to protect itself from attacks. When the OpenFlow controller receives the client subnet information from the authoritative DNS server, the client subnet will be investigated on the whitelist and blacklist and based on the result the OpenFlow controller sends instructions to the Open vSwitches. Finally, the Open vSwitch updates the destination IP address, MAC address and output port for the incoming packet based on the instructions from the OpenFlow controller. As a result, if the source IP address of the incoming packet is not included in the whitelist, the packet will be forwarded to the server1 otherwise to the server2.

Based on the above procedure, we make all the three clients access the server1 as following: 1) the client1 (included in the whitelist) accessed the server1 using its FQDN (Fully Qualified Domain Name); 2) the client2 (not included in the whitelist or blacklist) accessed the server1 using its FQDN; 3) the client3 (not included in the whitelist or blacklist) accessed the server1 using its IP address directly. All the accesses were performed by sending ICMP (Internet Control Message Protocol) [28] packets using PING command. Figure 5 shows the flow entries added to the flow table of the Open vSwitch after the three clients sent ICMP packets but it does not include the pre-configured flow entries shown in Table 3. The table shows two flow entries line1 and line2 which corresponding to the client1 and client2 respectively. The line1 indicates that the packet sent from "192.168.1.1" to "192.168.1.4" will be sent out from the port2 of the Open vSwitch. Similarly, the line2 indicates that the packet sent from "192.168.1.2" to "192.168.1.5" will be sent out from the port3 of the Open vSwitch. Consequently, we confirmed that the ICMP packets sent from the client1 were forwarded to the server1 and those sent from the client2 were forwarded to the server2. On the other hand, the flow entry corresponding to the client3 was not created in the flow table which means that all the ICMP packets sent from the client3 were dropped on the Open vSwitch since the client3 attempted to access the server1 using its IP address directly without domain name resolution.

5.2 Performance Evaluation

We also conducted performance evaluation using the same local experimental network. In order to measure and compare the performance of the prototype system with the conventional systems, we considered five different network conditions by generating background network traffic as following: 1) "no flood" which means there are no attack traffic; 2) "one flood" which means generating attack traffic from one client; 3) "two floods" which means generating attack traffic from two clients; 4) "one random" which means generating attack traffic from one client using random source IP address; 5) "two randoms" which means creating attack traffic from two clients using random source IP address.

5.2.1 Delay of Adding a Flow Entry in Open vSwitch with Collaboration of Domain Name Resolution

First, we measured the delay of creating one flow entry on the Open vSwitch, from the point when the Open vSwitch forwards a DNS query to the authoritative DNS server to the point when the OpenFlow controller finished adding the flow entry on the Open vSwitch. We synchronized the system clock of the authoritative DNS server and the OpenFlow controller using NTP (Network Time Protocol) [29] and measured the delay using Perl and Ruby scripts. By measuring the delay for 100 times, we obtained the average delays and standard deviations in the five different patterns as shown in Table 4. From the results, we can see that even with the critical attack traffic on the background (two randoms) the success rate of dig command reached to 62% with the delay less than 8 milliseconds and we consider the value is acceptable for a flow entry creation on a Open vSwitch.

Table 4 Pre-configured flow entries of the Open vSwitch.

Patterns	no flood	one flood	two floods	one random	two randoms
Min delay (ms)	2.21	2.13	2.33	2.19	2.37
Max delay (ms)	3.00	2.94	2.80	41.63	48.32
Average delay (ms)	2.51	2.50	2.59	7.21	7.62
Standard dev. (ms)	0.24	0.28	0.08	6.12	6.53
dig success rate (%)	100	100	100	77	62

5.2.2 Throughput Comparison of Prototype and Conventional Systems Considering Network Conditions

In the performance evaluation, we measured the throughput in two patterns independently: between a client on whitelist and the server1 and between an unknown client and the server1. In the two patterns, the client1 was used for the client on whitelist and unknown client correspondingly and in both cases the client2 and 3 were used for generating ICMP attack traffic to the server1 using its IP address directly as described in Sect. 5.1. With no doubt, the ICMP attack traffic from the client2 and 3 were dropped on the Open vSwitch since no corresponding flow entries will be created for those attack packets. We used "iPerf"[†] on the client1 and server1 for 5 times with 10 seconds per time and measured the average throughput. For the ICMP traffic from the client2 and 3, we used "Hping3"[††] with "–flood" option which indicates sending packets as fast as possible. In addition, in order to reproduce DDoS like attack, we also used "–rand-source" option which sets sender IP address randomly in IPv4 protocol. We evaluated and compared the performance with the following different system configurations.

1 Normal switch: Used normal layer 2 switch in the experimental network without using any SDN protocol.
2 CL to Sv1/Atk to Sv1 (only for the client on whitelist): All traffic is forwarded to the server1 (CL to Sv1) and attack traffic targets on the server1 (Atk to Sv1).
CL to Sv1/Atk to Sv2 (only for the client on whitelist): All traffic is forwarded to the server1 (CL to Sv1) and attack traffic targets on the server2 (Atk to Sv2).
3 CL to Sv2/Atk to Sv1 (only unknown client): All traffic is forwarded to the server1 (CL to Sv1) and the attack traffic targets on the server1 (Atk to Sv1).
CL to Sv2/Atk to Sv2 (only for the unknown client): All traffic is forwarded to the server1 (CL to Sv2) and the attack traffic targets on the server2 (Atk to Sv2).
4 No name resolution/Atk to Sv1: Assign firewall unit by checking the client IP address on the whitelist using SDN without using domain name resolution. This configuration creates flow entries for all packets arriving at Open vSwitch using PacketIn method and the attack traffic targets on the server1 (Atk to Sv1).
5 No name resolution/Atk to Sv2: The same as 4 except that the attack traffic targets on the server2 (Atk to Sv2).
6 Prototype/Atk to Sv1: Use the prototype system (OpenFlow protocol and domain name resolution) and the attack traffic targets on the server1 (Atk to Sv2). Note that the incoming traffic without domain name resolution will be dropped in this configuration.
7 Prototype/Atk to Sv2: The same as 6 except that the attack traffic targets on the server2 (Atk to Sv2).

[†]iPerf - https://iperf.fr/
[††]Hping - http://www.hping.org/

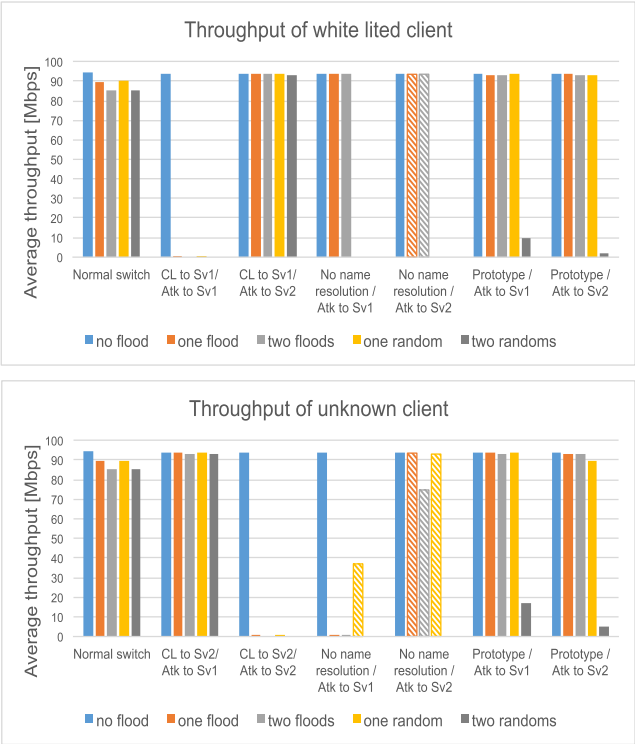

Fig. 6 Throughput comparison: prototype and conventional systems.

Using the above seven different system configurations, we measured and compared the throughput in the different five patterns described in Sect. 5.2 (no flood, one flood, two floods, one random and two randoms) and Fig. 6 shows the measurement results. We performed the evaluation for both clients on whitelist and unknown. From the graphs we can see the following points.

First, we can confirm that under "normal switch" or "no flood" condition, all system configurations provide similar throughput without significant performance decrease since there are no security check or attack traffic on the background. Then we also can summarize that when the target of attack traffic on the background differs from the communication destination of the clients such as the cases of "CL to Sv1/Atk to Sv2" and "CL to Sv2/Atk to Sv1", the results did not show significant throughput decrease in all five patterns.

Next, when the target of the attack traffic on the background is the same as the communication destination of the client such as the cases of "CL to Sv1/Atk to Sv1" and "CL to Sv2/Atk to Sv2", the results show significant through-

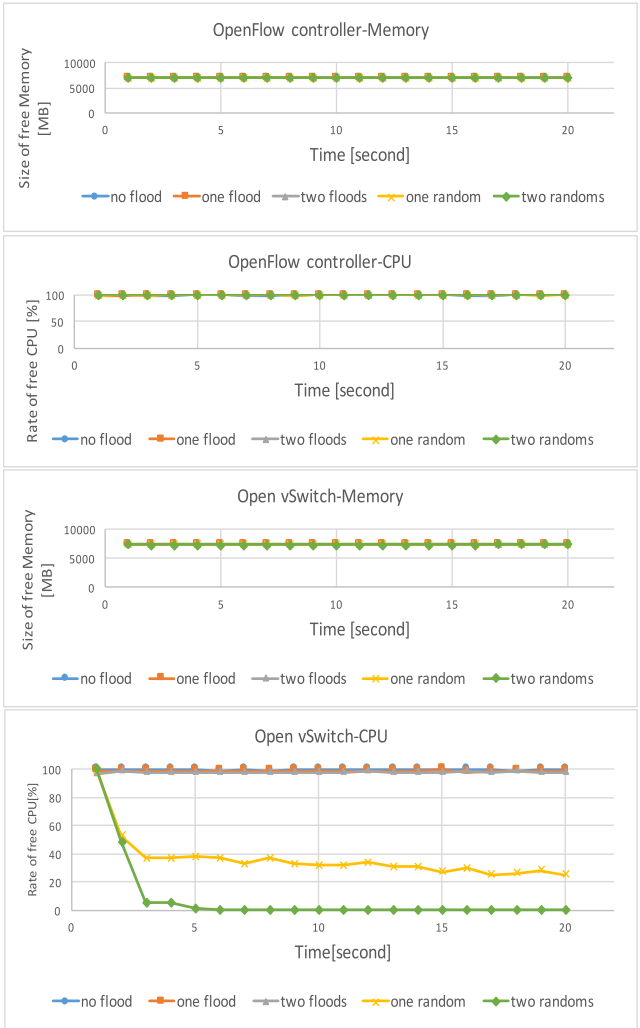

Fig. 7 CPU and memory status of OpenFlow controller and vSwitch.

put decrease in the patterns of "one random" and "two randoms" for the client on whitelist. For the unknown client, in all patterns except "no flood", the results show the same throughput decrease. We consider the reason is that the check and drop process for the attack traffic on the Open vSwitch cause high workload and consequently the throughput was decreased.

Continuously, when there was no name resolution process, only the two patterns "one flood" and "two floods" in the case of "No name resolution/Atk to Sv1" show similar results while all other cases show no traffic. Note that we used downward diagonal on other three cases which are "No name resolution/Atk to Sv2" for the clients on the whitelist, "No name resolution/Atk to Sv1" and "No name resolution/Atk to Sv2" for the unknown client. These downward diagonal bars mean that when we added the flow entries for the client1 manually under the attack traffic, the client1 showed the throughputs since during the attack traffic the Open vSwitch cannot create the flow entries due to the heavy workload.

Finally, except the pattern "two randoms", the prototype system shows almost the same throughput for both clients on whitelist and unknown. We consider the reason of the performance deterioration in the pattern "two randoms" is that the prototype system works by software unlike the "Normal switch" which is hardware basis. Thus we also consider that it is worth taking other advantages even if the performance may be lower than the "Normal switch".

From the above performance evaluation results, we can confirm that the proposed method can be expected to provide both lightweight and detail inspections for network traffic without throughput reduction and be applicable for real network environment with acceptable administrative cost.

5.3 Investigation of the Performance Deterioration

In order to investigate the reason of performance deterioration (in the "two randoms"), we measured the free memory size and CPU idle percentage of the OpenFlow controller and Open vSwitch using "vmstat" command. Considering the change of network condition, we measured under the same five patterns described in Sect. 5.2 (no flood, one flood, two floods, one random and two randoms). We performed the measurement with 1 second interval for 20 seconds and the results are shown in Fig. 7. From the results, we can confirm that DoS attacks (one random and two randoms) will not cause significant memory consumption on both of the OpenFlow controller and Open vSwitch. On the other hand, although the CPU idle percentage of the OpenFlow controller had no change during the ICMP flooding attack, the Open vSwitch was affected significantly. Especially when both clients used random source IP addresses in the ICMP flooding attack, the CPU idle percentage of the Open vSwitch was almost zero. Based on the result of CPU usage in the Open vSwitch, we consider that CPU resource exhaustion can be the reason of the performance deterioration. We consider that the sharp decrease of CPU idle percentage in Open vSwitch happened because the Open vSwitch ("ovs-vswitchd" and "ovs-dbserver") is software basis. Therefore we expect that it is possible to improve the performance of the proposed firewall system even under DoS attacks by using hardware based OpenFlow controller and switches. Many hardware based approaches to improve the performance of OpenFlow systems (RAM, processor, BUS, etc.) have been proposed [30]. Moreover, by using multiple OpenFlow controllers for redundancy and load balancing, the performance can be improved further [31], [32].

6. Conclusion

In this paper, we proposed an SDN-based proactive firewall system in collaboration with domain name resolution. The proposed firewall system includes two sub firewall units and the SDN controller forwards the incoming traffic to the proper one based on the client subnet information obtained from the internal authoritative DNS server using EDNS Client Subnet Option. Among the two sub firewall units, one

performs lightweight check for the incoming traffic while the other performs detailed investigations. By separating the incoming traffic based on the client subnet information, the proposed firewall system not only can inspect all the incoming traffic appropriately but also can keep the performance. We implemented a prototype system using OpenFlow protocol and Perl modules as well as BIND name server. Using the prototype system, we evaluated itsperformance of our proposed firewall system under several patterns of network condition. According to the evaluation results, we confirmed that the prototype firewall system worked correctly as we designed and can provide better performance compare to conventional systems. We also observed performance deterioration of the prototype system under some specific DoS attacks and we consider that the problem can be solved by using hardware based SDN architecture. The future work includes evaluations using real firewall facilities and performance analysis in real network environment.

Acknowledgements

This work was partially supported by JSPS Grants-in-Aid for Scientific Research (KAKENHI) Grant Number JP25330105.

References

[1] T. Otsuka, N. Yamai, K. Okayama, Y. Jin, H. Ikarashi, and N. Kitagawa, "Design and Implementation of Proactive Firewall System in Cooperation with DNS and SDN," The 31st Int'l Tech. Conf. on Circuits/Syst., Comput. and Commun. (ITC-CSCC 2016), Naha, Japan, pp.25–28, July 2016.

[2] J. Verdú, M. Nemirovsky, and M. Valero, "MultiLayer Processing - An execution model for parallel stateful packet processing," The 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, USA, pp.79–88, 2008.

[3] F. Yu, Z. Chen, Y. Diao, T.V. Lakshman, and R.H. Katz, "Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection," Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems, San Jose, California, USA, pp.93–102, 2006.

[4] C. Contavalli, W. van der Gaast, D. Lawrence, and W. Kumari, "Client Subnet Option in DNS Queries," RFC7871, IETF, 2016.

[5] "What are FireHOL and FireQOS?" https://firehol.org/ (accessed 2018-05-12).

[6] FORCEPOOINT, "Forcepoint Stonesoft Next Generation Firewall," https://www.forcepoint.com/ja/product/network-firewall/forcepoint-stonesoft-next-generation-firewall (accessed 2018-05-10).

[7] Dell, "Network Security Solutions | Firewall Hardware, Software & Services - SonicWall," https://www.sonicwall.com/ (accessed 2018-05-10).

[8] H. Hamed and E. Al-Shaer, "Dynamic Rule-ordering Optimization for High-speed Firewall Filtering," Proceedings of the 2006 ACM Symposium on Information, computer and communications security, Taipei, Taiwan, pp.332–342, March 2006.

[9] A. Ganesh, A. Sudarsan, A. Krishna Vasu, and D. Ramalingam, "IMPROVING FIREWALL PERFORMANCE BY USING A CACHE TABLE," IJAET, vol.7, no.5, pp.1594–1607, Nov. 2014.

[10] D.S. Alexander, W.A. Arbaugh, M.W. Hicks, P. Kakkar, A.D. Keromytis, J.T. Moore, C.A. Gunter, S.M. Nettles, and J.M. Smith, "The SwitchWare active network architecture," in IEEE Netw., vol.12, no.3, pp.29–36, May/June 1998.

[11] R. Narisetty and D. Gurkan, "Identification of Network Measurement Challenges in OpenFlow-based Service Chaining," Proc. 8th IEEE Workshop on Network Measurements (WNM2014), Edmonton, Canada, pp.663–670, Sept. 2014.

[12] AT&T, BT, CenturyLink, China Mobile, Colt, Deutsche Telekom, KDDI, NTT, Orange, Telecom Italia, Telefonica, Telstra, Verizon, "NetworkFunctions Virtualization - Introductory White Paper," Oct. 22–24, 2012, https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[13] "Open Networking Foundation" (online), available from https://www.opennetworking.org/.

[14] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," Proc. IEEE Commun. Mag., vol.51, no.2, pp.114–119, Feb. 2013.

[15] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild," The 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services, Boston, MA, pp.12–17, 2011.

[16] W. Chen, H. Li, Q. Ma, and Z. Shang, "Design and implementation of server cluster dynamic load balancing in virtualization environment based on OpenFlow," The 9th International Conference on Future Internet Technologies (CFI2014), Tokyo, Japan, June 2014.

[17] M. Nugraha, I. Paramita, A. Musa, D. Choi, and B. Cho, "Utilizing OpenFlow and sFlow to Detect and Mitigate SYN Flooding Attack," Journal of Korea Multimedia Society, vol.8, no.8, pp.988–994, Aug. 2014.

[18] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with SDN: A feasibility study," Computer Networks, vol.85, pp.19–35, 2015, ISSN 1389-1286.

[19] S. Kim, S. Lee, G. Cho, M.E. Ahmed, J. Jeong, and H. Kim, "Preventing DNS Amplification Attacks Using the History of DNS Queries with SDN," vol.10493, pp.135–152, 2017, 10.1007/978-3-319-66399-9_8.

[20] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), Fez, pp.258–263, 2016.

[21] T. Otsuka, Gada, N. Yamai, K. Okayama, and Y. Jin, "Design and Implementation of Client IP Notification Feature on DNS for Proactive Firewall System," 2015 IEEE 39th Annual Conference on Computer Software and Applications, Taichung, pp.127–172, July 2015.

[22] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC2267, IETF, May 2000.

[23] R. Brown, "Net::DNSServer-0.11," http://search.cpan.org/~bbb/Net-DNSServer-0.11/lib/Net/DNSServer.pm (accessed 2018-05-10).

[24] Internet Systems Consortium, "BIND, The most widely used Name Server Software," https://www.isc.org/downloads/bind/ (accessed 2018-05-10).

[25] N. Labs, "Net::DNS::Nameserver," http://search.cpan.org/dist/Net-DNS/lib/Net/DNS/Nameserver.pm (accessed 2018-05-10).

[26] "Trema:Full-Stack OpenFlow Framework in Ruby and C," https://trema.github.io/trema/ (accessed 2018-05-10).

[27] "Open vSwitch," http://www.openvswitch.org/ (accessed 2018-05-10).

[28] F. Gont and C. Pignataro, "Formally Deprecating Some ICMPv4 Message Types," RFC6918, IETF, April 2013.

[29] D. Mills, U. Delaware, J. Martin, Ed., J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC5905, IETF, June 2010.

[30] O.E. Ferkouss et al., "A 100Gig network processor platform for openflow," 2011 7th International Conference on Network and Service Management, Paris, France, pp.286–289, Oct. 2011.

[31] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "BalanceFlow: Controller load balancing for OpenFlow networks," 2012 IEEE 2nd International Conference on Cloud Computing and Intelligent Sys-

tems, Hangzhou, pp.780–785, Oct. 2012.
[32] H. Yao, C. Qiu, C. Zhao, and L. Shi, "A Multicontroller Load Balancing Approach in Software-Defined Wireless Networks," International Journal of Distributed Sensor Networks, vol.11, no.10, 2015.

Hiroya Ikarashi received his B.E. degree in computer and information sciences from Tokyo University of Agriculture and Technology, Japan in 2016. Since April 2016, he has been master's course in information science at Tokyo University of Agriculture and Technology graduate school. His research interests include Network Architecture and network security. He is a student member of IPSJ.

Yong Jin received his M.E. degree in electronic and information systems engineering and Ph.D. degree in Industrial Innovation Sciences from Okayama University, Japan in 2009 and 2012, respectively. In April 2012, he joined the Network Architecture Laboratory of National Institute of Information and Communications Technology, Japan, as a researcher. From October 2013, he joined the Global Scientific Information and Computing Center of Tokyo Institute of Technology as an assistant professor. His research interests include network architecture, network security, traffic engineering and Internet technology. He is a member of IPSJ and IEEE.

Nariyoshi Yamai received his B.E. and M.E. degrees in electronic engineering and his Ph.D. degree in information and computer science from Osaka University, Osaka, Japan, in 1984, 1986 and 1993, respectively. In April 1988, he joined the Department of Information Engineering, Nara National College of Technology, as a research associate. From April 1990 to March 1994, he was an Assistant Professor in the same department. In April 1994, he joined the Education Center for Information Processing, Osaka University, as a research associate. In April 1995, he joined the Computation Center, Osaka University, as an assistant professor. From November 1997 to March 2006, he joined the Computer Center, Okayama University, as an associate professor. From April 2006 to March 2014, he was a professor in the Information Technology Center (at present, the Center for Information Technology and Management), Okayama University. Since April 2014, he has been a professor in the Institute of Engineering, Tokyo University of Agriculture and Technology. His research interests include distributed system, network architecture and Internet. He is a member of IPSJ and IEEE.

Naoya Kitagawa received his B.Sc. and M.Sc. degree in information science from Chukyo University, Toyota, Japan in 2009 and 2011 respectively, and his Ph.D. degree in information science from Nagoya University, Nagoya, Japan in 2014. In April 2014, he joined Information Technology Center, Nagoya University as a postdoctoral fellow. Since October 2014, he has been an assistant professor in the Institute of Engineering, Tokyo University of Agriculture and Technology. His research interests include the Internet, network security, and distributed system. He is a member of IPSJ.

Kiyohiko Okayama received his B.S., M.S. and Ph.D. degrees in information and computer sciences from Osaka University, Japan, in 1990, 1992 and 2001, respectively. After he has worked in the Department of Information System at Osaka University and in the Graduate School of Information Science at Nara Institute of Science and Technology as a research associate, he joined the Department of Communication Network Engineering at Okayama University in 2000. From 2005 to 2011, he joined the Information Technology Center at Okayama University. Since 2011, he has been an associate professor in Center for Information Technology and Management at Okayama University. His research interests include network design and network security. He is a member of IEICE.