Regular Paper

NS record History Based Abnormal DNS traffic Detection Considering Adaptive Botnet Communication Blocking

Hikaru Ichise[1,2,a) Yong Jin[3,b) Katsuyoshi Iida[4,c) Yoshiaki Takai[4,d)

Received: May 10, 2019, Accepted: November 7, 2019

Abstract: DNS (Domain Name System) based name resolution is one of the most fundamental Internet services for both of the Internet users and Internet service providers. In normal DNS based name resolution process, the corresponding NS (Name Server) records are required prior to sending a DNS query to the authoritative DNS servers. However, in recent years, DNS based botnet communication has been observed in which botnet related network traffic is transferred via DNS queries and responses. In particular, it has been observed that, in some types of malware, DNS queries will be sent to the C&C servers using an IP address directly without obtaining the corresponding NS records in advance. In this paper, we propose a novel mechanism to detect and block abnormal DNS traffic by analyzing the achieved NS record history in intranet. In the proposed mechanism, all DNS traffic of an intranet will be captured and analyzed in order to extract the legitimate NS records and the corresponding glue A records (the IP address(es) of a name server) which will be stored in a white list database. Then all the outgoing DNS queries will be checked and those destined to the IP addresses that are not included in the white list will be blocked as abnormal DNS traffic. We have implemented a prototype system and evaluated the functionality in an SDN-based experimental network. The results showed that the prototype system worked well as we expected and accordingly we consider that the proposed mechanism is capable of detecting and blocking some specific types of abnormal DNS-based botnet communication.

Keywords: Botnet communication, DNS, NS record, glue A record, direct outbound query, NS history database

1. Introduction

Botnet, a malicious logical network constructed by cyber attackers, has become one of the critical security threats in cyberspace [4], [5]. Once a computer is infected by a bot program, which is a kind of malware and also a core program of a botnet, the bot-infected computer basically attempts several kinds of cyber attacks such as Advanced Persistent Threat (APT), Distributed Denial of Service (DDoS), spreading spam mails, ransomware attacks, phishing [6], [7], etc. In general, the botnet-based cyber attack can be divided into infection, botnet communication and attack. First, a computer in an intranet, which is an internal computer network within an organization such as universities, companies and governmental departments, etc., somehow gets infected by a bot program such as through web browsing, spam mail, or clicking a phishing site by mistake. Then, the bot program sends probes to its corresponding Command and Control (C&C) server to identify its existence as well as to update its status. After collecting a number of bot-infected computers, the C&C server can instruct them to perform several kinds of cyber attacks. Here, we refer to the communication between a bot-infected computer and the C&C server, which is the most important information transmission in a botnet-based cyber attack, as botnet communication. With regarding the above workflow, in this research we target botnet communication as a means to analyze and detect botnet-based cyber attacks.

Many recent reports indicate that DNS protocol [8], [9] has become used in botnet communication [10], [11], [12]. DNS protocol has been mainly used for name resolution, such as translating hostnames to IP addresses on the Internet. However, the increase of Internet services has led to wide uses of some minor records such as DNS TXT record. In Ref. [13], Xu et al. empirically showed that cyber attackers can effectively hide botnet communication by using a DNS-based stealthy messaging system that uses hash functions to encode the contents. In Refs. [14], [16], Ichise et al. analyzed DNS packet traces to differentiate the normal and abnormal uses of DNS TXT records. Consequently, DNS traffic which so far has been considered to be secure network traffic has also become a target of being monitored communication since network administrators cannot simply block all DNS traffic. Thus, it is important for the network administrator to detect and block DNS-based botnet communications.

Figure 1 shows a general DNS based name resolution process with a deployed DNS full resolver in an intranet. A client computer first sends a DNS query to the DNS full resolver for request-

[1] Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Hokkaido 060–0814, Japan
[2] Technical Department, Tokyo Institute of Technology, Ota, Tokyo 152–8550, Japan
[3] Global Scientific Information and Computing Center, Tokyo Institute of Technology, Ota, Tokyo 152–8550, Japan
[4] Information Initiative Center, Hokkaido University, Sapporo, Hokkaido 060–0811, Japan
a) hichise@nap.gsic.titech.ac.jp
b) yongj@gsic.titech.ac.jp
c) iida@iic.hokudai.ac.jp
d) takai@iic.hokudai.ac.jp

Preliminary versions of this paper have been presented in Refs. [1], [2], [3].

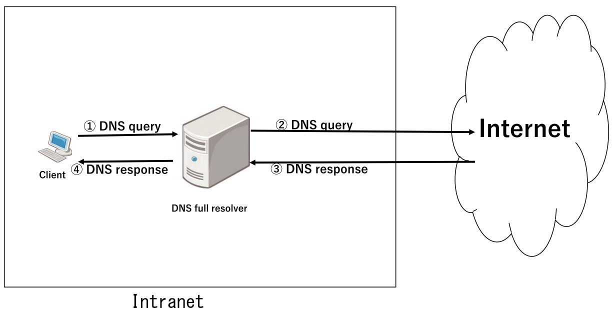

Fig. 1 An example of normal DNS name resolution.

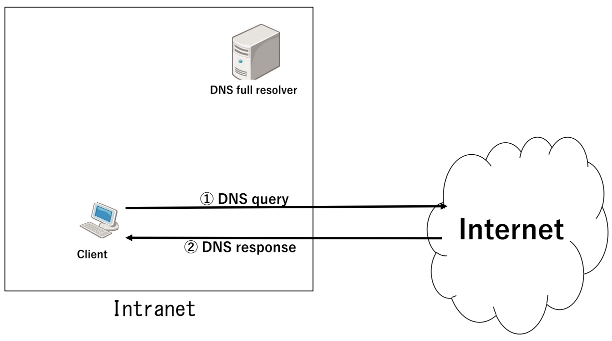

Fig. 2 An example of abnormal DNS name resolution.

ing name resolution. Then the DNS full resolver performs the name resolution and replies back the DNS response to the client computer. In the name resolution process, the DNS full resolver achieves the corresponding DNS NS (Name Server) records and glue A records of the authoritative DNS servers prior to sending DNS queries to them. Here, a NS record is used for indicating the hostname of an authoritative name server of the queried domain name, and its corresponding glue A record means the IP addresses of the authoritative name servers. In almost of all cases, the internal clients rely the DNS name resolution on the DNS full resolvers deployed in the intranet.

On the other hand, botnet communications may not follow the same process [17], [18] as we will explain in the next section. **Figure 2** shows a typical anomalous DNS traffic such as a type of botnet communication using DNS protocol. In this example, a client computer sends a DNS query to the Internet directly without using the DNS full resolver deployed in the intranet. We call this type of DNS name resolution request as a direct outbound DNS query. However, there also exist some exceptions that this type of direct outbound DNS Q&R can be used for normal use cases. For example, in the case of using public DNS servers such as Public DNS operated by Google [20] and/or Cloudflare [21], a client computer may send direct outbound DNS queries to the public DNS servers without using the DNS full resolvers deployed in the intranet. Therefore, in the proposed system, we also consider these exceptions and allow the computers in the intranet use the public DNS servers.

In summary, DNS name resolutions are supposed to be performed by DNS full resolvers in almost all normal cases, whereas it is not general in abnormal use cases. Based on this observation, we propose a detection and blocking method of anomalous DNS traffic. Our key idea is that normal name resolution obtains the NS and glue A records of the corresponding authoritative DNS servers prior to sending DNS queries to them while bot programs send direct outbound DNS queries without obtaining the NS and glue A records.

In this paper, we focus on detecting and blocking anomalous DNS traffic by analyzing the achieved NS records history. Based on our proposed method, we constructed a prototype system for detecting and blocking anomaly DNS traffic using virtual machines. In particular, for detecting a botnet communication that uses direct outbound DNS queries, we analyze the achieved NS record and the corresponding glue A record from our campus network and created a legitimate NS record history database. We evaluated the features of the prototype system and also performed some preliminary performance evaluations using a local experimental network. According to the evaluation results, we confirmed that the prototype system worked effectively as it was expectable to deploy our proposed method in real network environment.

2. DNS-based Botnet Communication and Related Work

As stated in the Introduction, the objective of our research is to construct a system for detecting and blocking DNS-based botnet communication. In this section, we introduce three types of DNS-based botnet communication; the way of using via-resolver DNS query, the way of using direct outbound DNS query and the way of using indirect outbound DNS query. Then we introduce some related researches.

2.1 Three Types of DNS-based Botnet Communication

We find out that there are three types of botnet communication using DNS; via-resolver DNS query, indirect outbound DNS query, direct outbound DNS query. The way of using via-resolver DNS query relies on DNS full resolver completely. In Ref. [22], the authors have reported the uses of DNS TXT records in botnet communication using via-resolver DNS query. Next, botnet communication using indirect outbound DNS queries is detected in a bot program named Morto [23], which uses direct outbound DNS queries after identifying its C&C server. **Figure 3** (a) shows that a bot-infected computer obtains the IP address of C&C servers by name resolutions via DNS full resolver at first, then sends DNS queries to a C&C server directly. On the other hand, a bot program named Feederbot never uses DNS full resolvers, which is called a direct outbound DNS query. Figure 3 (b) shows that the IP address of C&C servers are hard-coded in the bot program so that the bot-infected computer can send DNS queries to C&C server using the IP address directly. These two types of DNS-based botnet communication which are used in Morto and Feederbot, never obtain legitimate NS records and the corresponding glue A records before sending DNS queries to the C&C servers. In this paper, we define "legitimate NS record", "normal DNS query and response" and "abnormal DNS query and response" as follows:

- Legitimate NS record: NS records obtained from authoritative DNS servers.
- Normal DNS query and response (Q&R): DNS queries sent

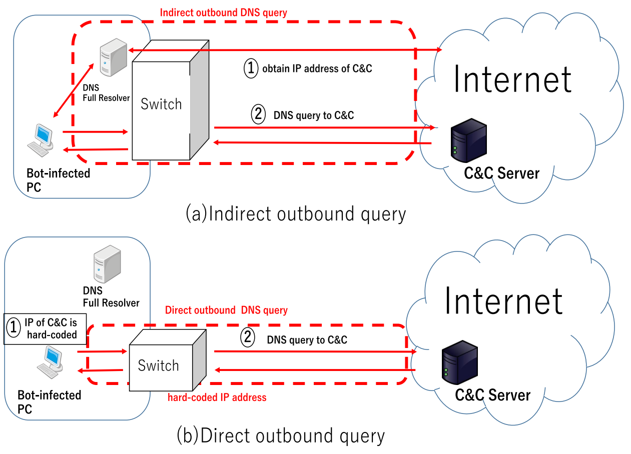

Fig. 3 Direct & indirect outbound DNS queries.

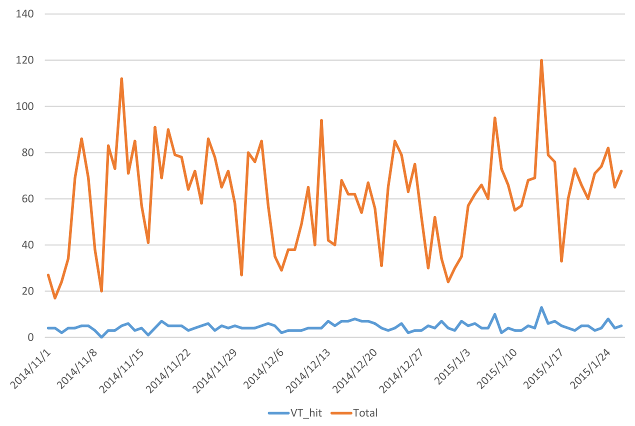

Fig. 5 In Ref. [16]. The number of direct outbound DNS.

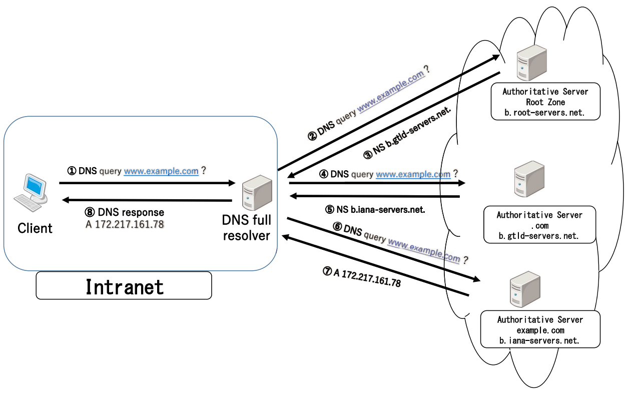
① DNS query www.example.com ?
⑧ DNS response A 172.217.161.78
Client
DNS full resolver
Intranet
② DNS query www.example.com ?
③ NS b.gtld-servers.net.
④ DNS query www.example.com ?
⑤ NS b.iana-servers.net.
⑥ DNS query www.example.com ?
⑦ A 172.217.161.78
Authoritative Server Root Zone b.root-servers.net.
Authoritative Server .com b.gtld-servers.net.
Authoritative Server example.com b.iana-servers.net.

Fig. 4 NS record process.

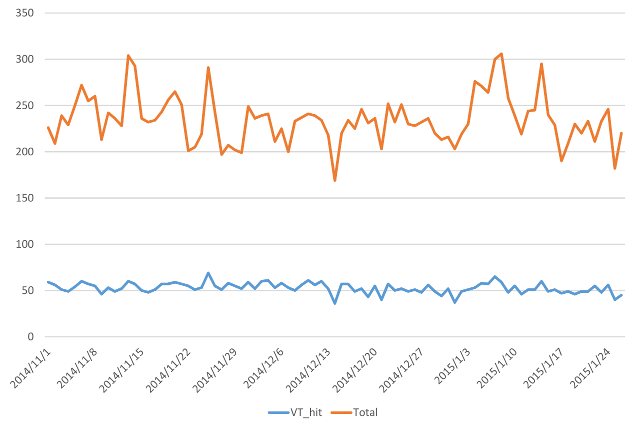

Fig. 6 In Ref. [16]. The number of indirect outbound DNS.

to DNS full resolver or authoritative DNS servers. DNS responses received from DNS full resolver or authoritative DNS servers.

- Abnormal DNS query and response (Q&R): DNS queries sent to other than DNS full resolver and authoritative DNS servers. DNS responses received from other than DNS full resolver and authoritative DNS servers.

Figure 4 illustrates DNS NS record resolution process in detail. First, the client sends a DNS query to the DNS full resolver. DNS full resolver then obtains NS record of all authoritative name severs from the root server. Finally, the DNS full resolver obtains the destination IP address of "www.example.com" and notifies the destination IP address to the client.

It should be noted that DNS full resolvers and authoritative DNS servers can also be compromised and reply wrong DNS responses. In this paper, we keep the point on the normal DNS name resolution process by focusing on obtaining legitimate NS records prior to sending DNS queries. Therefore, the case of a compromised DNS full resolver and authoritative DNS servers is beyond the scope of this paper and we omit the detailed discussion. Accordingly, we consider that if we can collect obtained legitimate NS records as well as the corresponding glue A records and check the destination IP addresses of all the DNS queries in the intranet, it will be possible to detect and block these types of DNS-based botnet communication.

2.2 Related Work

In recent years, many securities related types of the research reported that the the DNS protocol had been widely used for botnet communication as well as malware attacks [15], [16], [17], [18], [19], and we introduce some DNS based related solutions in the following.

In Ref. [15], the authors surveyed and categorized the existing researches about DNS-based botnet detection. The major topic of this paper is to introduce the detection method using a DNS mechanism, it also explains some existing researches about botnets that use the DNS protocol.

In Ref. [16], the authors focused on the fact that DNS TXT records might be used for botnet communication and categorized the uses of a DNS TXT record in order to differentiate the normal use of a DNS TXT record and the use of a DNS TXT record in a botnet communication. From the analysis results, obtaining all DNS traffic from a border router in our university, **Fig. 5** in Ref. [16] showed the number of malicious destination IP addresses in direct outbound DNS queries is 5 per day, that is, the average hit rate is about 8%. In addition, **Fig. 6** in Ref. [16] showed the rate of the malicious destination IP address in an indirect outbound DNS query is 22%. The authors confirmed that besides the malicious uses of a DNS TXT record, there were also several normal uses so that it was necessary to detect and block malicious uses effectively.

In Ref. [17], the authors analyzed 14 million DNS TXT queries and found the bot program named "Feederbot" which has to query the C&C server directly, that is to say, bypassing the DNS full resolver. However, none of the existing researches considered the role of DNS NS (Name Server) records in the name resolution process. Moreover, as all reports never discussed direct/indirect outbound DNS queries so far, it was difficult to detect and block only DNS-based botnet communications. Thus, we tend to create automatic detecting and blocking system for DNS-based botnet communications by constructing the legitimate NS records history database for monitoring all DNS queries.

In Ref. [18], the authors referred to DNS tunneling technology in which the malicious contents are included in the labels of domain names being used as a part of DNS queries. To detect malicious DNS tunneling traffic, they discussed two separated categories, payload analysis, and traffic analysis. In payload analysis, they analyzed tunnel indicators from the payload of a DNS query and response packets with focusing on Domain Generation Algorithms. In traffic analysis, they analyzed over time in DNS traffic.

In Ref. [19], the authors proposed a real-time detection method of tunneling of data over DNS using machine learning techniques. The authors also implemented and preliminarily evaluated the proposed system. Since this method is based on machine learning, the performance will heavily depend on the learning datasets.

As we can see from the above DNS based related researches, many of them only focused on DNS traffic analysis and failed to consider the DNS name resolution process. In this paper, we purpose to solve this issue in conventional solutions.

3. Proposed System

In order to detect and block DNS-based botnet communications, we first construct a database of whitelisted DNS NS and glue A records. This list includes the DNS NS records of domain names achieved from all received normal DNS responses such as domain names ".com" and ".net" and their corresponding glue A records. We also include other well-known IP addresses for specific applications such as update servers of anti-virus software and public DNS servers to which the internal computers may send DNS queries directly. When a client sends direct outbound DNS queries to the listed servers we consider them as normal, while to those not listed in the white list we will drop the packets and log them down for further investigations. We use Software Defined Network (SDN) technology, specifically OpenFlow switch and controller, for detecting and blocking the DNS traffic in the proposed system. When a DNS query packet is "packet in" from the switch to the controller, the controller queries the aforementioned white list database for the destination IP address. The controller will then send instructions to the switch to drop or allow the DNS query packet. **Figure 7** shows the brief workflow of the proposed system.

3.1 Design of NS Record History Database

Figure 8 illustrates the detailed flow chart of NS_DB operations. First, we construct the NS_DB from the corresponding DNS queries and responses. One of the simple methods is to obtain all DNS traffic from an intranet and pick out legitimate DNS NS records as well as the corresponding glue A records. For the simplicity, we create query_DB first for storing all normal DNS

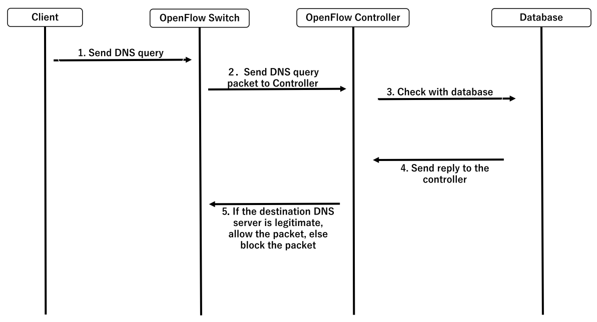
Client
OpenFlow Switch
OpenFlow Controller
Database
1. Send DNS query
2. Send DNS query packet to Controller
3. Check with database
4. Send reply to the controller
5. If the destination DNS server is legitimate, allow the packet, else block the packet

Fig. 7 Overview of the workflow in the proposed system.

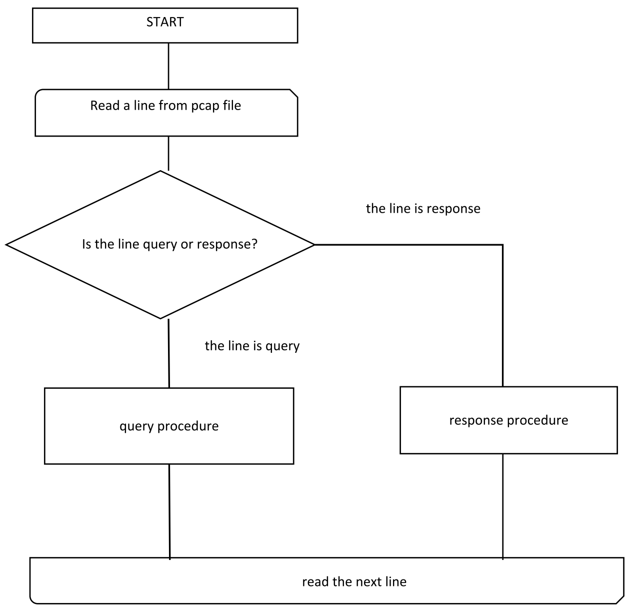

Fig. 8 The workflow of NS_DB creation program.

queries. Next, we construct the NS_DB using all the achieved normal DNS responses to the queries stored in the query_DB. We captured all DNS traffic in pcap format using tcpdump program and analyzed them in order to construct the NS_DB. Specifically, in the pcap file, if the line is a DNS query the analysis program performs the procedure for a DNS query otherwise for a DNS response. In the DNS query procedure, a new entry will be added to query_DB while in the DNS response procedure a new entry will be added to NS_DB. Note that some responses have NS records with the corresponding glue A records while some others have NS records only. In case of where the DNS responses only have NS records (such as when an out-of-bailiwick domain name is used for NS record), only the NS records will be registered to the NS_DB. Then the analysis program continuously monitoring DNS traffic. If DNS queries for the NS records are sent and the corresponding responses are received, the glue A records obtained from the responses will be registered as glue A records in NS_DB. Thus, we defined glue A record as glue A record in an additional section and A record for out-bailiwick NS record in an answer section. We set the following column of NS_DB.

- querytime: Received time of DNS query in milliseconds
- queryid: Message id of the query
- queryname: FQDN and record type of the query
- res_time: Received time of DNS response in milliseconds
- zname: Zone name of authoritative name server

Table 1 Configuration of MariaDB.

querytime	queryid	queryname	res_time	zname	nsttl	nsfqdn	glueAttl	glueA	del_time
decimal(20,0)	int	varchar(10000)	decimal(20,0)	varchar(1000)	decimal(20,0)	varchar(1000)	decimal(20,0)	varchar(1000)	decimal(20,0)

- nsttl: TTL of NS record
- nsfqdn: FQDN of authoritative name server
- glueAttl: TTL of glue A record
- del_time: Expected expire time in milliseconds calculated through nsttl or glueAttl

Table 1 showed as configuration of MariaDB.

3.2 Overview of SDN and Ryu

SDN technology [24], [25] refers to a new approach for network programmability by providing the capacity to initialize, control, change and manage network behavior dynamically via open interfaces. SDN introduces the abstraction of the data forwarding plane from the control plane. What this means is that SDN allows a network engineer to develop a program that can control, inspect and/or modify the flow of all network traffic. With this ability, we can check if the packets are sent from the client is a DNS query and how to handle them. SDN is able to perform this task by decoupling the standard routine of a switch or router into 3 separate planes: application, control, and data plane.

The lowest level, the data plane, has no logic. Logical decisions are made in the control plane. The data plane is in charge of sending and receiving frames based on instructions received from the control plane. The middle level, the control plane, is a black box that provides an interface from the application plane to the data plane. The key takeaway is that the control plane's logic can be logically centralized. This makes the management and scalability of an intranet easy. The highest level, the application plane, is where network engineers write a program to control the network traffic. We will write a program here to inspect, determine if the client's query has legitimacy and drops the packet if it is not normal by sending a dropped signal to the control plane. There are many SDN APIs available out on the Internet such as Floodlight [26]. In this paper, we write the controller program using Python [27] based SDN solution Ryu [28] which supports the OpenFlow protocol.

3.3 System Architecture

We use the following terminologies in the description of the proposed system:

- Query destination IP Address: The destination IP address to which the client sends a DNS query.
- Database: For storing the legitimate DNS NS and glue A records as well as normal public DNS servers.

Basically, we use SDN technologies for controlling the packets in the proposed system. All packets will be transferred to the OpenFlow switch and the OpenFlow controller decides whether or not pass through the specific packets. In the proposed system, the destination IP address of a packet will be checked in the NS record history database. **Figure 9** shows a simple system architecture of the proposed method by using SDN technologies. We describe the basic procedure of the proposed system using an example in which a client sends a direct outbound DNS query to the Internet in the following.

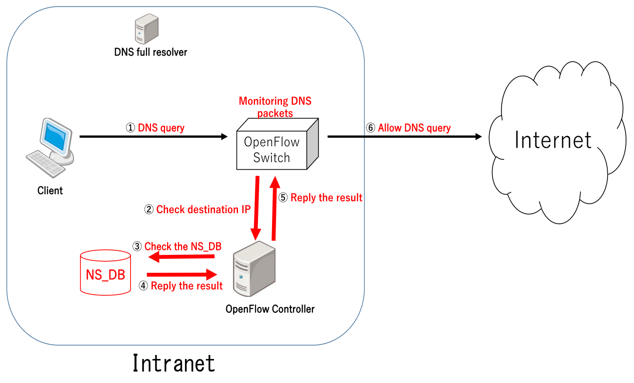

Fig. 9 The detailed procedure of the proposed system.

(1) A client sends a DNS query to the Internet directly and the packet will be transferred to the OpenFlow switch.
(2) The OpenFlow switch receives the DNS query and forwards the packet to the OpenFlow controller since the information about the destination IP address of the DNS query packet is not in the flow table.
(3) When the OpenFlow controller receives the DNS query packet, it inspects the packet to obtain the queryname and the destination IP address. The controller checks the destination IP address of the DNS query packet in the NS record history database.
(4) The database replies the corresponding entry (can be multiple entries) if the destination IP address is in the database.
(5) The OpenFlow controller decides whether or not the DNS query packet passes through based on the result. If the destination IP address of the DNS query packet is in the database, then the OpenFlow controller passes it as a normal DNS query, otherwise, the OpenFlow controller will drop the DNS query packet.
(6) In the case where the destination IP address of the DNS query packet is in the database, the DNS query packet will be allowed to be transferred to the Internet.

Based on the above procedure, the proposed system can detect and block abnormal DNS traffic which is supposed to be used in some types of DNS-based botnet communication.

4. Implementation

Based on our proposed system, we created two programs for the implementation of the prototype system: the program to create NS_DB and the controller program to control the OpenFlow switch. We describe the detailed contents in the following sections.

4.1 Construction of NS Record History Database

In order to create the NS record history database using DNS traffic in pcap format, we created a DNS query database named query_DB first and eventually created the database NS_DB.

Table 2 An example of a table entry in the NS record history database.

querytime	queryid	queryname	res_time	zname	nsttl	nsfqdn	glueAttl	glueA	del_time
1414782044594	31812	yahoo.co.uk	1414782044659	yahoo.co.uk	57954000	ns3.yahoo.com	51169000	203.84.221.53	1414833213659

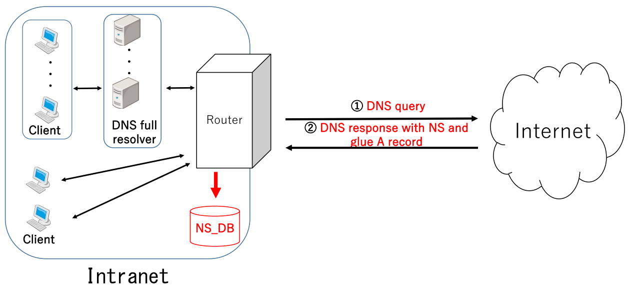

Fig. 10 DNS data collection in case of DNS response with NS and glue A record.

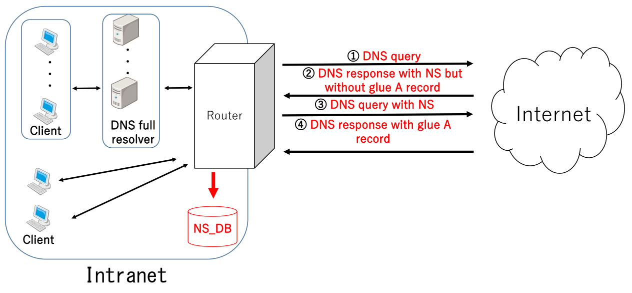

Fig. 11 DNS data collection in case of a DNS response with NS but without glue A record.

We used Python language in the program and imported the dpkt module for processing the pcap files, and finally we used MariaDB [29] as the database system.

Figures 10 and **11** illustrate the brief network topology we used to obtain DNS traffic in our campus network. All DNS queries (the arrows numbered 1 in Fig. 10) and DNS responses (the arrows numbered 2 in Fig. 10) are obtained from the border switch in pcap format. Then the analysis program analyzes the pcap file and stores the legitimate NS records and corresponding glue A records in the NS_DB. Note that we do not construct an empty database in the initial NS_DB, but register destination IP addresses of root server and TLD in the initial setup. **Table 2** shows an example of the entry in the NS_DB. Moreover, if the DNS response only includes NS records, that is, the corresponding glue A records are not included, the NS records will be registered without glue A records as Fig. 11 shows. After that, if the glue A records will be received continuously within two seconds the remaining information about the corresponding glue A records will be added in the NS_DB. Otherwise, the NS records will be deleted. Some destination IP addresses of the direct outbound queries are normal. For example, we need to register IP addresses of public DNS, of DNS full resolver, and of antivirus to glue A records, because those IP addresses are normal. Note that the DNS traffic of fixed interval (appropriately one month) is obtained, analyzed by setting in period of learning. Then, our system sets NS_DB in a real SDN network, obtains the DNS traffic, which update NS_DB, and runs a blocking direct outbound DNS query. The NS record and the corresponding glue A record is deleted in expiring time to live. The size of NS_DB can be restrained.

Table 3 Software versions used in the implementation.

Operating system, software and database	Version
CentOS	7.4
Open vSwitch	2.9
Ryu	4.23
Python	2.7
MariaDB	10.3

4.2 Detection and Blocking Features

In the prototype system, we used the SDN technology to realize the detection and blocking features, specifically OpenFlow solutions. We choose Open vSwitch [30] and the OpenFlow switch and Ryu program as the OpenFlow controller. The former is a software version of the OpenFlow switch program and the latter is a solution of OpenFlow controller program. We constructed the network environment using Kernel Virtual Machine (KVM) and used CentOS 7.4 operating system for both host and guest machines. **Table 3** shows the detailed version information for the operating system, SDN solutions and database system. In the prototype system, all packets will be transferred to the Open vSwitch and the Open vSwitch only checks DNS packets. When the Open vSwitch cannot find an entry for a DNS query packet, the "packet_in" function will send the DNS query packet to the Ryu controller. Then the Ryu controller checks the destination IP address of the DNS query packet in the NS_DB in order to determine how to process it. If the destination IP address is stored as a glue A record in the NS_DB, the DNS query packet will be passed through the Open vSwitch by the Ryu program. Otherwise, the DNS query packet will be blocked by the Ryu program. It should be noted that in the current system other traffic will be passed through the Open vSwitch.

5. Evaluation

In order to verify the feature and performance of the proposed system, we evaluated our system in a local testbed. In the feature evaluation, we mainly focused on the functionality of the controller program to check the ability of the system about detecting and blocking malicious DNS traffic. The evaluations include database construction part and malicious DNS traffic detection and blocking part. Next, to quantitatively evaluate the performance, we conduct the stress test by comparing the conventional and proposed systems.

5.1 Experimental Network Environment

To conduct the evaluation of the prototype system, we have set up a local experimental network environment consisting of an OpenFlow swtich (Open vSwitch), an OpenFlow controller (Ryu), a NS record history database (NS_DB) which is created from pcap files, a replica database to be accessed from OpenFlow controller, a client, C2Server to send a direct outbound DNS query, Web to download program with a direct outbound DNS query, and a DNS, which DNS application is BIND version 9.9.4. These are shown in **Fig. 12**. The OpenFlow switch was installed

Table 4 The results of feature evaluation.

Command	Result of checking the destination IP address by NS_DB	Behavior of OpenFlow switch
nslookup www.google.com 8.8.8.8	8.8.8.8 is unknown	blocked
nslookup www.google.com 8.8.4.4	8.8.4.4 is registered glueA record	passed
nslookup www.yahoo.co.uk 203.84.221.53	203.84.221.53 is registered glueA record	passed

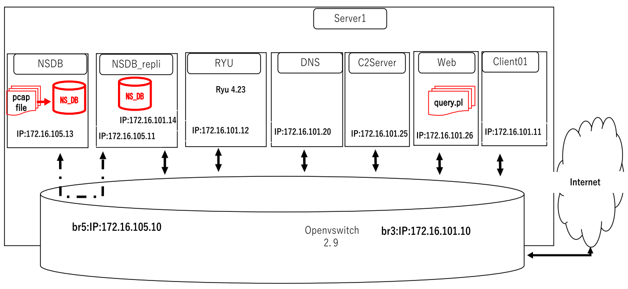

Fig. 12 The network topology used in evaluation.

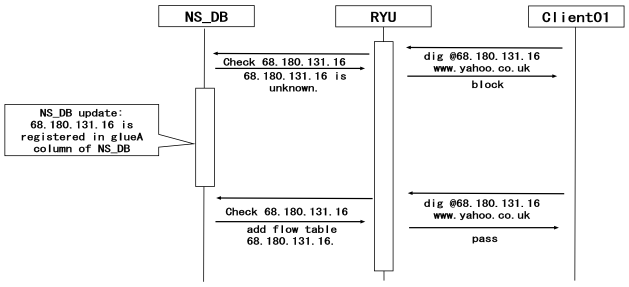
NS_DB
RYU
Client01
dig @68.180.131.16 www.yahoo.co.uk
Check 68.180.131.16
68.180.131.16 is unknown.
block
NS_DB update: 68.180.131.16 is registered in glueA column of NS_DB
dig @68.180.131.16 www.yahoo.co.uk
Check 68.180.131.16
add flow table 68.180.131.16.
pass

Fig. 13 Check registration of glue A in NS_DB.

on a host OS, whereas the controller (RYU), client (Client01), DNS (DNS), replica database (NSDB_replica), C2Server, Web, and NS record history database (NSDB) were installed as Kernel Virtual Machine (KVM) on host OSs. The controller, DNS, client, C2Server, Web, and replica database were configured to be connected to the same network in an OpenFlow switch and they could only communicate with the global Internet through the OpenFlow switch. The replica database and NS history database were connected to another network in the OpenFlow switch and synchronized by Galera Cluster, which is one of the most advanced software for multi-master synchronous replication [31]. Moreover, NS_DB can use index for improving search performance. We used NSDB_replica in order to avoid a deadlock caused by simultaneous update and reference for the same entry. That is NSDB_master will be used for maintaining the database while the NSDB_replica will be used for search and reference. More specifically, we created a replica database from the NS record history database using Galera Cluster. Note that records of the NS record history database will be deleted when del_time expires. Consequently, all traffic from the client and the DNS will pass through the OpenFlow switch and thus the OpenFlow controller can check the destination IP addresses as well as the result of packet filtering at the switch. In our previous study [32], we obtained many pcap files of DNS traffic from the border router of our campus network. In the evaluation, we used the pcap files for constructing the NS record history database. A pcap file of DNS traffic for about two hours approximately has a size of 200 MB and we used it in the evaluation. It took one hour to analyze the pcap file and to store the NS records history in the database. In particular, from the pcap formatted file, we matched the DNS queries and the responses. Next, we registered all NS records in the authority section and the corresponding glue A records in the additional section to the NS_DB. If the response only has NS records without glue A records like out-of-bailiwick domain name, we only registered the NS records in the NS_DB. Note that the normal destination IP addresses, such as Public DNS, were registered to NS_DB. As a result, 18,100 legitimate NS records were picked from the pcap file and stored in the NS record history database.

5.2 Feature Evaluation

After constructing the NS records history database in the experimental network environment, we evaluated the malicious NS traffic detection and blocking feature of the proposed system, which is mainly implemented in the controller program. Specifically, we attempt to check the destination IP address of a DNS query packet sent from the client with the glue A records stored in the NS record history database (NS_DB) and the OpenFlow switch will be instructed by the controller program to block the DNS query if there was no glue A record in the database.

To test the controller program, we sent DNS query from the client (172.16.101.11) through the command described in **Table 4**. When the OpenFlow switch receives a DNS query packet having a new query request, "packet_in" packet will be forwarded to the controller program, which will check whether the destination IP address is stored in the database. If the destination IP address is stored as a glue A record in the NS_DB, the controller program will create a log entry "<destination IP address> is registered as a glue A record" and the query has to be passed and passes the DNS query packet. Otherwise, it will log "<destination IP address> is unknown" and the query has to be blocked and blocks the DNS query packet.

We described the evaluation results in Table 4. We tested for three types of IP addresses as the destination IP address of the DNS query from the client; 8.8.8.8 which is an open DNS resolver provided by Google and we did not register it in NS_DB to be able to perform the test, 8.8.4.4 which is another open DNS resolver provided by Google and is registered in NS_DB, and 203.84.221.53 which is an authoritative DNS server of the domain name "yahoo.co.uk" and is registered in the NS_DB.

In addition, we checked the feature of the program to manipulate the NS_DB with the flow illustrated in **Fig. 13** in the following. Firstly, we made the NS_DB empty and then made the client send the DNS query using "dig @68.180.131.16 www.google.co.uk" to the Internet with running the RYU controller. The results showed that the RYU controller blocked the DNS query with "68.180.131.16 is unknown" message due to the destination IP address "68.180.131.16" does not exist in NS_DB. Next, we updated the NS_DB with adding the IP address "68.180.131.16" as a glue A record which eventually showed

Table 5 The results of query time performance using dig.

Command	Type of DNS resolver	Conventional method		Proposed method	
		Average [ms]	Standard deviation [ms]	Average [ms]	S. D. [ms]
dig @172.16.101.20 www.google.com (w/o cache)	Internal	515.5	92.13	530.7	69.42
dig @172.16.101.20 www.google.com (with cache)	Internal	0	0	0	0
dig @8.8.8.8 www.google.com	Public DNS (8.8.8.8)	40.9	2.33	42.9	22.85
dig @1.1.1.1 www.google.com	Public DNS (1.1.1.1)	4.6	2.00	4.0	1.25

Table 6 The results of query speed performance using dnsperf.

Command	Type of DNS Resolver	Conventional method			Proposed method		
		Average query speed [query/s]	S. D. of q. s. [q/s]	Limitation of max. q. s. [q/s]	Average q. s. [q/s]	S. D. of q. s. [q/s]	Limitation of m. q. s. [q/s]
dnsperf -s 172.16.101.20 domainlist (w/o cache)	Internal	80.13	5.81	108	40.14	0.23	42
dnsperf -s 172.16.101.20 domainlist (with cache)	Internal	2976.59	409.52	N.A.	2955.29	1014.65	N.A.

"68.180.131.16 is registered in glue A column of NS_DB" message. We checked that there is "68.180.131.16". Then we also manually checked the NS_DB and confirmed the IP address "68.180.131.16" had been added as a glue A record entry. After that, we made the client send the same DNS query using "dig @68.180.131.16 www.google.co.uk" again to the Internet. The results showed that the RYU controller passed the DNS query with "add flow table 68.180.131.16" message since the destination IP address "68.180.131.16" had been added in the NS_DB.

Moreover, we performed the additional feature evaluation in the following Fig. 12 as a tested environment again. The objective showed that our proposed system can block a direct outbound DNS query which was sent to a C&C server in a real complicated network environment. We constructed a virtual environment as well as a real network on host OSs, generated a direct outbound DNS query which was sent to C2server and checked whether the direct outbound DNS query was blocked. Firstly, the destination IP address of C2Server was not registered in NS_DB. In addition, we constructed a HTTP server including perl program "query.pl" which sends a direct outbound DNS query to a C2server on the Web. This is one process to be a bot-infected PC in a real environment. In this condition, we run the controller program. Client01 was able to download query.pl by "wget http://web01.exampl.com/query.pl" from Web. When client01 run "query.pl", we checked blocking the DNS query with a TXT record sent to C2Server by the tcpdump command in C2Server. This result showed that the system worked well as we expected. Note that we maintain the NS record and glue A record entries for all domains including the root domain and the top level domains, which prevents unnecessary blocks of the DNS queries.

5.3 Performance Evaluation

After completing the feature evaluation of NS_DB, we also evaluated the performance of the prototype using the experimental network shown in Fig. 12 in order to check the overhead of OpenFlow architecture in the proposed system. Note that 8.8.8.8 is registered in the NS_DB and the query has to be passed and passes the DNS query packet. The Client01 can send a DNS query to the DNS or a public DNS server on the Internet. We measured the latency of the DNS name resolution in various patterns and the detailed results are described in **Tables 5** and **6**.

First, we measured the latency of a single DNS query by using "dig" command from the Client01. As described in Table 5, when the Client01 used the internal DNS full resolver (172.16.101.20) the average latency in the conventional method (without using the prototype) for ten times was 515.5 ms while that in the prototype was 530.7 ms. By calculating the difference between the two types of latencies we confirmed the latency might be caused by the proposed system which was 15.2 ms (530.7–515.5). In this case, we restarted the DNS before sending each DNS query from the Client01 which is indicating that the results were not cached in the DNS. When we used the cache in the DNS, we can see that all the latencies are "0" since all the DNS responses were sent back from the DNS. On the other hand, we also measured the latency with the same scenario using public DNS servers (8.8.8.8 and 1.1.1.1) and we found that, in both public DNS servers, the latency of the prototype was only a little more than that of a conventional method. We consider the reason was that when we used the DNS all the flow entries were added in the OpenFlow Switch by checking the OpenFlow Controller with "packet_in" function thus the latency was much higher. Moreover, as we can see that the standard deviation was also high in all evaluation cases and we consider the reason was that the network conditions were changing dynamically on the Internet thus the latency of each DNS query was also changed.

Next, we conducted performance evaluations using a DNS performance measurement tool named dnsperf 2.1.0 [33] which was installed on the Client01. In addition, the parameters of dnsperf "-s" identified target DNS IP address. We used one thousand domain names obtained from the QuantCast Measure [34] which provides a list of web sites. In this case, we only used the DNS and measured the query speed in [query/s] which indicates the number of queries sent per second when there was no packet loss which means all the one thousand domain names were processed successfully with DNS name resolution. As we can see from the Table 6, when we did not use the cache of the DNS, the average query speed of the proposed method was 40.14 [query/s] which was much less than that of the conventional method with 80.13 [query/s]. Here, when we add the limitation of maximum query speed (query/s) with 108 in the conventional method, we found that all the domain names were processed successfully while the value was 42 in the proposed method. This means the latency in the proposed method was much higher than that in the conventional method which shows the same results as in Table 5. On the

Table 7 List of public DNS servers.

8.8.8.8, 8.8.4.4 (dns.google)
1.1.1.1, 1.0.0,1 (Cloudflare)
208.67.222.123 (opendns)
9.9.9.9 (quad9.net)
64.6.64.6, 64.6.65.6 (nstld.net)
216.146.35.35, 216.146.36.36 (dynect.net)
77.88.8.1, 77.88.8.2, 77.88.8.3, 77.88.8.88 (yandex.ru)
185.228.168.10, 185.228.168.11, 185.228.168.168, 185.228.168.169 (cleanbrowsing.org)
180.76.76.76 (baidu.com)
114.114.114.114 (114dns.com)
8.20.247.20 (dnsbycomodo.com)
161.97.219.84 (sourpuss.net)
104.168.144.17 (hostwindsdns.com)

Table 8 False positive rate of the proposed system.

	The number of destination IP addresses in pcap files (excluding registered glue A in NS_DB)	The number of destination IP addresses in pcap files	Ratio (%)
1st pcap file	1573	10773	14.60
2nd pcap file	1441	9599	15.01
3rd pcap file	1676	9694	17.28
4th pcap file	1647	9455	17.41
5th pcap file	1255	7095	17.68
6th pcap file	1395	9437	14.78
7th pcap file	1445	10170	14.20
8th pcap file	1460	10933	13.35
9th pcap file	1567	11239	13.94
10th pcap file	1544	10896	14.17
Avarage	N/A	N/A	15.25
Total	2479	26894	9.22

other hand, when we allow the cache function of the DNS, in this case we did not indicate the limitation of max query speed, we can see that the query speed of the proposed method was about a half of that of the conventional method. We consider the reason was that the overhead of the OpenFlow architecture in the proposed method caused the low performance. However, we also think that it is possible to improve the performance of the proposed method by tuning up the OpenFlow Controller program as well as the matching method of the "packet_in" function and we plan to address this problem in our future work. Finally, as we can see from Table 6 when we limited the max query speed, the standard deviation was small while when we did not set the limitation the standard deviation was large. We consider the reason was that when there was no limitation on query speed the network condition can be easily changed so that the query speed can also be easily affected.

So far, our proposed method is basically an off-line detection method. To construct an online and pseudo real-time detection method, we can adjust the interval to create a single pcap file. For example, in our campus network, a two hours interval is corresponding to approximately 200 MB of pcap file size. If a two hours interval does not satisfy a requirement from users, we can choose a smaller interval time, e.g., two minutes. Another important thing is the processing speed. Let's consider the following scenario with a two hours interval. In the first interval, the system records the first pcap file. In the second interval, the system updates NS_DB based on the first pcap file as well as it records the second pcap file. If the NS_DB update processing requires more than two hours, the system will not work as an online method. According to our experiments, the processing time becomes less than two hours, which is fast enough to construct an online system.

6. Discussion

In this paper, to identify the abnormal DNS traffic used in botnet communication, we have proposed a detection system based on the observation; in a typical anomalous DNS traffic, a client computer sends a DNS query to the Internet directly without using the DNS full resolver. To identify such anomalous DNS traffic, we maintain a database of NS record history. Based on this concept, we have designed, implemented a system and evaluated the feature and performance. One important issue in the proposed system is the database learning time (e.g., for one month), because the proposed system can decrease false positive/negative by storing many NS records and glue A records. Database entries will be updated dynamically, and if the database does not contain required NS record history entries, e.g., at the beginning phase of the system, then the false positive/negative may occur. To deal with this matter, we have to introduce the database learning phase.

(1) Just after the system started, no DNS queries will be immediately blocked.
(2) After the learning phase finished, anomalous DNS queries will be blocked.

We evaluated a metric in terms of the detection accuracy of our proposed. The evaluation metric was the false positive rate (FPR) in terms of malicious DNS queries, which is presented in **Table 8**. More specifically, we use the first ten pcap files in log data, which were obtained in our previous study [32]. First, we manually confirmed that the pcap files do not contain any malicious traffic. And, a pcap file of DNS traffic for about two hours approximately has a size of 200 MB. In our proposed method, the destination IP address of DNS query packets will be registered in NS_DB if the query was legitimate. So, the destination IP address not registered in NS_DB was a false positive. In Table 8, we show the FPR of ten individual pcap files as well as the total FPR which was calculated using concatenated ten pcap files. As a result, the ratio was 9.22% (=2479/26894) in the total. 2479 IP addresses may include public DNS, and other legitimate use cases including the software update protocol. Note that the famous public DNS servers described in **Table 7** were registered in NS_DB for decreasing the FPR, however other legitimate use cases may exist. Another source of the false positive is the division of multiple pcap files, i.e., after a change in saved pcap files, glue A record happens, so that the NS record has not been registered in NS DB. This is the reason why the total FPR is much larger than the individual FPRs. We can decrease this type of false positive to shorten the interval of pcap file saving.

Another way to decrease the false positive/negative rates is to introduce a monitoring system; the network administrator will cooperate with the automatic detection system. The design and performance evaluation of the learning phase and the monitoring system will be future work.

7. Conclusion

According to many security reports and researches, several types of DNS-based botnet use direct outbound queries and responses (Q&R), which are different processes form the normal DNS name resolutions. Based on this observation, in this paper, we proposed a detection and blocking system of DNS-based botnet communication using a NS record database. In order to differentiate the normal and abnormal DNS queries, we created a database which stores NS records and the corresponding glue A records. In the normal use cases, NS records and the corresponding glue A records will be obtained prior to sending DNS queries to them, whereas some types of DNS-based botnet communication will not confirm the principle necessarily. Therefore, by using our proposed system, the destination IP address of a DNS query in a normal DNS name resolution will be stored in the database and the packet will be passed through as is while the destination IP address of malicious DNS traffic will not be included in the database so that the traffic will be blocked.

Based on our proposed system, we also implemented a prototype system using SDN technologies and performed feature evaluations and a preliminary performance evaluation. The prototype system consists of a NS record history database (NS_DB), a replica of NS_DB, an Open vSwitch and a Ryu controller. We used MariaDB for the database system and customized the Ryu controller using Python programing language. Based on the feature evaluation results using the prototype system on a local network environment, we confirmed that our proposed system worked as we designed and it was expectable to detect and block some types of DNS-based botnet communication. Moreover, based on the preliminary performance results we consider that although the results did not show a good performance it is possible to improve it by tuning the Ryu program, and finally, the proposed system can also be deployed in a real network environment.

For the future work, we obtain a NS record and need to analyze the corresponding glue AAAA record. Thus, we plan to set IPv6 in an OS or a network environment. we plan to tune up the Ryu controller program in order to improve the performance of the proposed system and feature evaluation as well as performance test on a real network environment. Furthermore, we also plan to expand the proposed system to apply for other types of DNS-based botnet communication as well as malicious DNS traffic.

References

[1] Ichise, H., Jin, Y., Iida, K. and Takai, Y.: Detection and Blocking of Anomaly DNS Traffic by Analyzing Achieved NS Record History, *Proc. Asia-Pasific Signal and Information Processing Association, Annual Summit and Conference 2018* (*APSIPA-ASC2018*), pp.1586–1590 (2018) (online), available from ⟨http://www.apsipa.org/proceedings/2018/pdfs/0001586.pdf⟩.

[2] Li, S., Jin, Y. and Iida, K.: Detection and Control of DNS-based Botnet Communications by using SDN-Ryu Solution, *IEICE Technical Report*, Vol.115, No.482, pp.73–78 (2016).

[3] Ichise, H., Jin, Y. and Iida, K.: Design and Implementation of NS Record History Database for Detecting DNS-based Botnet Communication, *IEICE Technical Report*, Vol.117, No.299, pp.7–11 (2017).

[4] Khattak, S., Ramay, N.R., Khan, K.R., Syed, A.A. and Khayam, S.A.: A Taxonomy of Botnet Behavior, Detection, and Defense, *IEEE Commun. Surveys & Tutorials*, Vol.12, No.2, pp.898–924 (online), DOI: 10.1109/SURV.2013.091213.00134 (2013).

[5] Amine, A., Mohamed, O.A. and Benatallah, B. (Eds.): *Network Security Technologies: Design and Applications*, Binsalleeh, H.: Botnets: Analysis, Detection, and Mitigation, pp.204–223, IGI Global (online), DOI: 10.4018/978-1-4666-4789-3.ch012 (2013).

[6] Soltani, S., Seno, S.A.H., Nezhadkamali, M. and Budiarto, R.: A Survey on Real World Botnets and Detection Mechanisms, *Int'l Journal of Information and Network Security*, Vol.3, No.2, pp.116–127 (2014).

[7] McAfee Labs: Threats Report (online), available from ⟨https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-mar-2018.pdf⟩ (accessed 2018-12-10).

[8] Mockapetris, P.: Domain Names: Concepts and Facilities, IETF RFC1034 (1987).

[9] Mockapetris, P.: Domain Names: Implementation and Specification, IETF RFC1035 (1987).

[10] Feily, M., Shahrestani, A. and Ramadass, S.: A Survey of Botnet and Botnet Detection, *Proc. IEEE Int'l Conference on Emerging Security Information, Systems and Technologies*, pp.268–273 (online), DOI: 10.1109/SECURWARE.2009.48 (2009).

[11] Bromberger, S.: DNS as a Covert Channel within Protected Networks (online), available from ⟨http://energy.gov/oe/downloads/dns-covert-channel-within-protected-networks⟩ (accessed 2018-12-10).

[12] Hands, N.M., Yang, B. and Hansen, R.A.: A Study on Botnets Utilizing DNS, *Proc. ACM Conference on Research in Information Technology* (*RIIT'15*), pp.23–28, ACM (2015).

[13] Xu, K., Butler, P., Saha, S. and Yao, D.: DNS for Massive-scale Command and Control, *IEEE Trans. Dependable and Secure Computing*, Vol.10, No.3, pp.143–153 (online), DOI: 10.1109/TDSC.2013.10 (2013).

[14] Ichise, H., Jin, Y. and Iida, K.: Analysis of Via-resolver DNS TXT Queries and Detection Possibility of Botnet Communications, *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing* (*PACRIM2015*), pp.216–221 (online), DOI: 10.1109/PACRIM.2015.7334837 (2015).

[15] Singh, M., Singh, M. and Kaur, S.: Issues and Challenges in DNS based Botnet Detection: A Survey, *Computers & Security*, Vol.12, pp.28–52, Elsevier (online), DOI: 10.1016/j.cose.2019.05.019 (2019).

[16] Ichise, H., Jin, Y. and Iida, K.: Analysis of DNS TXT Record Usage and Consideration of Botnet Communication Detection, *IEICE Trans. Commun.*, Vol.E101-B, No.1, pp.70–79 (online), DOI: 10.1587/transcom.2017ITP0009 (2018).

[17] Dietrich, C.J., Rossow, C., Freiling, F.C., Bos, H., Steen, M. and Pohlmann, N.: On Botnets that use DNS for Command and Control, *Proc. IEEE European Conference on Computer Network Defence* (*EC2ND'11*), pp.9–16 (online), DOI: 10.1109/EC2ND.2011.16 (2011).

[18] Farnham, G.: Detecting DNS tunneling (online), available from ⟨https://www.sans.org/reading-room/whitepapers/dns/paper/34152⟩ (accessed 2018-12-10).

[19] Ahmed, J., Gharakheili, H., Raza, Q., Russell, C. and Sivaraman, V.: Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks, *Proc. IFIP/IEEE Symposium on Integrated Network and Service Management 2019* (*IM2019*), pp.649–653 (2019) (online), available from ⟨http://dl.ifip.org/db/conf/im/im2019short/188679.pdf⟩.

[20] Google: Introduction to Google Public DNS (online), available from ⟨https://developers.google.com/speed/public-dns/docs/intro⟩ (accessed 2018-12-10).

[21] Cloudflare: Introduction to Cloudflare Public DNS (online), available from ⟨https://www.cloudflare.com/learning/dns/what-is-1.1.1.1/⟩ (accessed 2018-12-10).

[22] OpenDNS: The Role of DNS in Botnet Command and Control (online), available from ⟨http://info.opendns.com/rs/opendns/images/WB-Security-Talk-Role-Of-DNS-Slides.pdf⟩ (accessed 2018-12-10).

[23] Mullaney, C.: Morto Worm Sets a (DNS) Record (online), available from ⟨http://www.symantec.com/connect/blogs/morto-worm-sets-dns-record⟩ (accessed 2018-12-10).

[24] Open Networking Foundation: SDN Architecture (online), available from ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf⟩ (accessed 2018-12-10).

[25] Denazis, S. et al.: Software-Defined Networking (SDN): Layers and Architecture Terminology, IRTF RFC7426 (2015).

[26] Project Floodlight: Floodlight (online), available from ⟨http://www.projectfloodlight.org⟩ (accessed 2018-12-10).

[27] Python: Python (online), available from ⟨https://www.python.org⟩ (accessed 2018-12-10).

[28] NTT: Ryu: Getting Started (online), available from ⟨https://osrg.github.io/ryu-book/en/html/⟩ (accessed 2018-12-10).

[29] MariaDB: MariaDB (online), available from ⟨https://mariadb.org⟩ (accessed 2018-12-10).

[30] Open vSwitch: Open vSwitch (online), available from ⟨https://www.openvswitch.org⟩ (accessed 2018-12-10).
[31] Galera Cluster: Galera Cluster (online), available from ⟨http://galeracluster.com⟩ (accessed 2018-12-10).
[32] Jin, Y., Ichise, H. and Iida, K.: Design of Detecting Botnet Communication by Monitoring Direct Outbound DNS Queries, *Proc. IEEE Int'l Confrerence on Cyber Security and Cloud Computing* (*CSCloud2015*), pp.37–41 (online), DOI: 10.1109/CSCloud.2015.53 (2015).
[33] Akamai: Measurement Tools (online), available from ⟨https://www.akamai.com/us/en/products/network-operator/measurement-tools.jsp⟩ (accessed 2018-12-10).
[34] QuantCast: Measure Audiences (online), available from ⟨https://www.quantcast.com⟩ (accessed 2018-12-10).

Hikaru Ichise received his B.S. degree in mathematics from Kwansei Gakuin University, Sanda, Japan in 2008. Currently, he is a technical staff in Tokyo Institue of Technology, Tokyo, Japan and student in Graduate School of Information Science and Technology, Hokkaido University, Japan. His research interest is of detecting botnet communciations using DNS protocol.

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 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, traffic engineering, network security and Internet technology. He is a member of IEICE and IEEE.

Katsuyoshi Iida received B.E., M.E. and Ph.D. degrees in, respectively, Computer Science and Systems Engineering from Kyushu Institute of Technology (KIT), Iizuka, Japan in 1996, Information Science from Nara Institute of Science and Technology (NAIST), Ikoma, Japan in 1998, and Computer Science and Systems Engineering from KIT in 2001. Currently, he is an Associate Professor in the Information Initiative Center, Hokkaido University, Sapporo, Japan. His research interests include network systems engineering such as network architecture, performance evaluation, QoS, and mobile networks. He is a member of the WIDE project and IEEE. He received the 18th TELECOM System Technology Award and the Tokyo Tech Young Researcher's Award in 2003 and 2010, respectively.

Yoshiaki Takai was born in 1960. He is currently a Professor and Director of Information Initiative Center, Hokkaido University. He is an Aide to CIO of Hokkaido University. From 1988 to 1989, he was a Research Associate of the Faculty of Science, the University of Tokyo. In 1989, He joined the Faculty of Engineering, Hokkaido University. His research interests include parallel computer architecture, parallel processing, distributed processing, mobile agents applications, computer networks, augmented reality, virtual reality, physically-based modeling, and computer graphics applications. He received B.Eng. in electronic engineering in 1983, and M.Eng. and D.Eng. in information engineering in 1985 and 1988 from Tohoku University. He is a member of IPSJ, IEICE, and IEEE Computer Society.