An Optimal Selection Mechanism Using OpenFlow from Multiple IP Addresses in DNS Response

ZHENGXU JIN[1,a)] YONG JIN[2,b)] NARIYOSHI YAMAI[1,c)] REI NAKAGAWA[1,d)]

Abstract: Load balancing technology not only provides fault tolerance functionality but also can distribute access requests from the end clients to multiple web servers. An end client may receive multiple IP addresses for a web server during the DNS based domain name resolution which will be performed in prior to accessing the web server. In general, the end client will use the first IP address in the list to access the web server even it may cause high latency due to the long distance between the end client and the physical web server or the web server is in high workload. As a result, the Internet service providers cannot provide effective Internet services to the Internet users. In this paper, we propose an optimal selection mechanism from multiple IP addresses which can make the end client access the most proper web server in terms of causing low latency or being in low workload using Software Defined Network (SDN) technology. In the proposed mechanism, by using OpenFlow protocol, the OpenFlow controller makes the OpenFlow switch access the multiple IP addresses simultaneously when the end client attempts to access the first IP address and only replies the response from the most proper web server to the end client. Consequently, the end client will access the most proper web server transparently even there are multiple IP addresses available for the domain name of the web server. We implemented a prototype system for the proposed mechanism and evaluated the features on a local experimental network. The evaluation results confirmed that the prototype system worked correctly as designed and the performance was acceptable with reasonable overhead.

Keywords: Load balancing, route selection, DNS, SDN, openflow

1. Introduction

The Internet has become one of the indispensable social infrastructures and with the dramatic increase of the Internet users the stable and effective Internet services are required. In order to reduce the latency of responses to the requests from the end clients and improve the quality of Web service, the distributed and parallel systems are configured at the Web server side. As a well-known solution, load balancing method by using multiple application servers is very popular especially on web-based Internet services. In this case, the domain name of a web server may have multiple IP addresses since there are multiple web servers providing the same service. In the literature, there have been many approaches proposed in order to make end clients select the optimal web servers such as DNS Round Robin [1], Dispatcher of Web servers and proxies which manages the incoming access requests and are responsible for selecting the optimal connection based on the state information of each of web servers. In principle, an end client performs DNS [2][3] based name resolution in prior to accessing a web server and receives multiple IP addresses as the result when there are mulpible web servers for a domain name. In DNS Round Robin technology, in general, an end client always uses the first one when there are multiple IP addresses for the domain name of a web server. Accordingly, even if the web server of the first IP addresses is in high workload status or it is more far away from the end client which may cause a high latency, the end client will still access the first web server based on the DNS Round Robin load balancing technology. It becomes a critical issue in the Internet especially with the situation of dramatic increase of Internet users.

On the other hand, in the solution of using dispatcher of web servers and proxies, high administrative cost for collecting and configuring the state information runtime is required. Since the web proxy which receives the access requests from the end clients needs to decide to which web server the requests should be forwarded. Therefore, some new features with more flexible mechanism are required on the network layer in order to obtain network conditions. However, it is difficult to add new features on the existed network protocol and deploy the update to all network facilities and computers. Software Defined Network (SDN) technology [4] which supports the flexibility and scalability is more low-cost and more effective for the innovation. The main concept of the SDN is decoupling of the control plane and data plane which provides the programmable environment for the network administrators. By practically using the SDN technology, in this thesis, I propose an optimal selection mechanism from multiple IP addresses from the DNS response which provides the end client the most optimal choice for accessing a web server.

[1] Tokyo University of Agriculture and Technology, Koganei, Tokyo, Japan
[2] Tokyo Institute of Technology, Meguro, Tokyo, Japan
[a)] zxjin@net.cs.tuat.ac.jp
[b)] yongj@gsic.titech.ac.jp
[c)] nyamai@cc.tuat.ac.jp
[d)] rnakagawa@net.cs.tuat.ac.jp

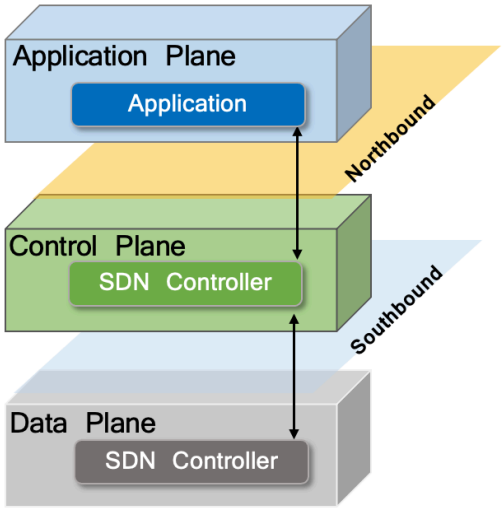

Fig. 1 SDN infrastructure

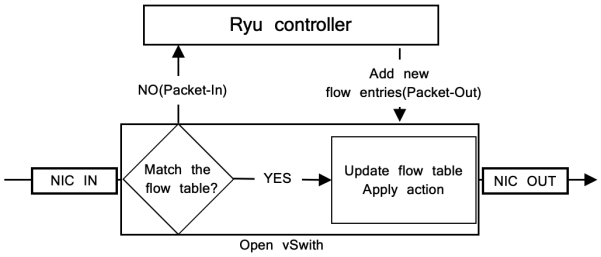

Fig. 2 The process of handling packets

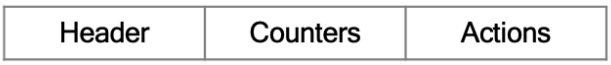

Fig. 3 Flow entry format

2. Load balancing and SDN technologies

2.1 Load Balancing

DNS-based load balancing technology [1] is one of the well-used solutions for distributing the access requests from the end clients to multiple servers and separate the workload of web servers in to different machines. This technology also can provide fault tolerance functionality so that when one of the web servers is down others can provide the same service. One of the well-used DNS-based load balancing technology is named DNS Round Robin, in which, multiple IP addresses are configured for a web server and they will be replied to the end clients in an alternative order during the DNS name resolution. In general, an end client uses the first IP address to access the web server even if it is in high workload or far away from the end client. Therefore, in DNS Round Robin, the end client cannot select the proper IP address based on the network condition and web server status.

2.2 Software Defined Networking (SDN)

The concept of SDN technology is defined by the Open Networking Foundation (ONF) [5] which is the most accepted SDN standard worldwide. The main character of SDN technology is that the decoupling the control plane from data plane. The SDN framework is illustrated in Figure 1. The data plane is responsible for forwarding packet according to the flow table. The control plane makes use of the objects provided by data plane that not matched with flow table to make the instructions and populates them to data plane for updating the flow table. The types of instructions include modification, transferring and discarding. The control plane communicates with the data plane through standardized interfaces which are named as southbound interfaces. The application plane which brings the new features in the SDN configuration has the global view of network. The control plane communicates with the application layer through another type of interface which is called by northbound interface.

2.2.1 OpenFlow Protocol

The communication protocol used in SDN technology is OpenFlow [6] which has been standardized by the ONF. Based on the OpenFlow protocol, two components consist the SDN architecture, one is the control plane which is referred as OpenFlow controller and the other is responsible for processing the packets which is referred as OpenFlow switch. The OpenFlow switch receives or forwards packets according to the flow table. The OpenFlow controller constructs and populates the flow tables using OpenFlow protocol. The flow table consists of flow entries each of which has three parts and the brief mechanism of packet processing in OpenFlow switch is illustrated in Figure 2. When an OpenFlow switch receives a packet, the header of the packet is parsed and then checked against the flow table. If the packet is matched to one of the flow entries, the subsequence of the operation that the counter of the flow entry will be updated. At the end, the OpenFlow switch handles the packet based on the specified actions. If there are no any mached flow engries, the OpenFlow switch sends the packet to the OpenFlow controller using Packet-In method. When the OpenFlow controller receives the Packet-In message, it sends an appropriate insturction to the OpenFlow switch using Packet-Out method in order to make the OpenFlow switch complete the packet processing such as forwarding the packet via the proper port.

2.2.2 OpenFlow controller and switch

There have been more than 30 different OpenFlow controllers developed by different groups in order to precisely control packet fordarding and contruct flexible instructions on a flow entry [7]. To ensure that the programmable control plane is more compatible and applicable to different administrators, it is written in a variety of programming languages. It is not only the Southbound API that needd to be focused on, but also the aspect of Northbound API needs to be considered.

The Open vSwitch (OVS), one of OpenFlow switch, has been developed in recent years. There are two main components of processing packets in Open vSwitch. One is ovs-vswitchd that is same as from one operating system and operating environment to another. The other is datapath kernel module which is used for the performance of operating system [8].

Figure 4 illustrates two major components of Open vSwitch

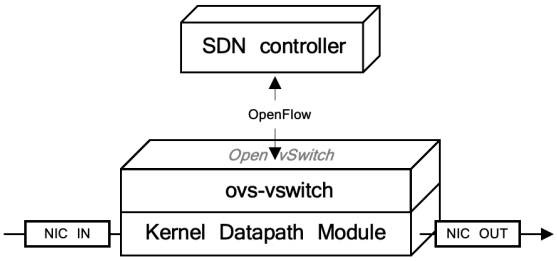

Fig. 4 Open vSwitch infrastructure

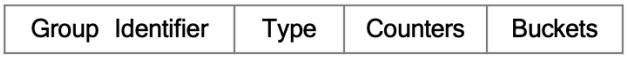

Fig. 5 Group table format

to handle packets. At first, when the kernel datapath module receives a packet from the NIC, it processes the packet under ovs-vswitchd instructions. There are two kinds of conditions, one is that set up instructions in advance by the ovs-vswitchd. Another one is that there are no prescribed instructions about how to handle the received packets, the kernel datapath module sends it to ovs-vswitchd and then the new instruction established and populated from ovs-vswitchd. The OVS and OpenFlow controller communicates through OpenFlow protocol which allows the OpenFlow controller to add, remove, update, monitor, and obtain statistics of flow tables and their flow entries, as well as to divert selected packets to the OpenFlow controller and to inject packets from the OpenFlow controller into the OpenFlow switch [8]. For the datapath kernel, there are two types of caching flows which called as Microflow caching and Megaflow caching. In the design of the first one, the cache entries are extremely fine-grained and match at most packets of a single transport connection. And the second one, it supports caching forwarding decisions for large aggregates of traffic.

2.2.3 Group Table

In this paper, we practically use a special fow table on the Openflow switch named group table. The format of a group table is showed in Figure 5. The group identifier is a 32-bit unsigned integer uniquely identifying the group. The group type determines which type of group table will be used. For now, there are four types defined by ONF. The types of ALL and INDIRECT are marked as “Required” and the types of SELECT and FAST FAILOVER are marked “Optional” which switch is not required to support [5]. In the proposed mechanism, the type of ALL is utilized for cloning and sending queries to all associated Web servers. When the Open vSwitch handles the packet following the rules of ALL type group table, the packet is effectively cloned for each bucket in the group, if a bucket contains the actions which is instructed directly to forward the packet to the ingress port, then this packet clone is dropped. The counters are updated when packets are processed by a group. The actions buckets contain a list of buckets, and each bucket contains an array of actions that are to be executed based on the associated parameters.

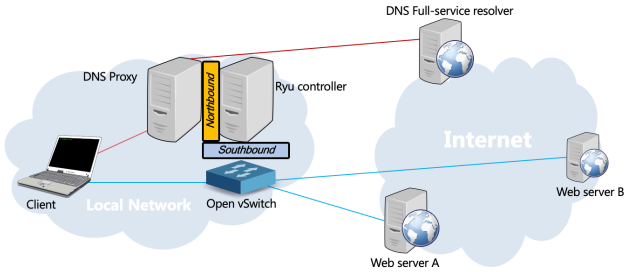

Fig. 6 Topology of the proposed mechanism

3. Proposed Mechanism

3.1 Architecture design

There are three main components in the proposed mechanism.

The first component is DNS proxy which is responsible for forwarding DNS queries and DNS responses. When the DNS proxy receives the DNS responses, a list of IP addresses, from the DNS full-service resolver which may cache the associated mapping addresses, it performs two processes. The first process is sending information which consists of Group ID, end client IP address and the list of IP addresses from resource records to the Ryu controller by northbound channel. The second process is selecting one IP address which is the top of the list to ensure end client which of IP address to access and sending it to end client as the DNS response. The second component is RESTful API. At the RESTful API, Ryu controller defines the URL to receive the information from the DNS proxy in the Ryu controller.The Ryu controller builds the flow entries and populates them to OVS.

The second component is RESTful API. In the RESTful API, the Ryu controller defines the URL to receive the information from the DNS proxy in the Ryu controller. The Ryu controller builds the flow entries and populates them to OVS.Two different kinds of flow entries will be populated to OVS. In the first type, the match filed consists of TCP SYN flag, source IP address (end client IP address) and destination IP address (top of the IP list) and the actions field points to the the associated group table. The group table consists of two parts. The first part is the type field in which the ALL type is assigned. As described in section 2.2.3, the ALL type provides the clone function. The second part is bucket field which consists of a set of bucket actions. Depending on the IP list, one of the bucket actions consist of rewriting destination IP address and transferring the packets to the corresponding web server. The second kind of flow entries which instruct OVS how to handle when receive packets from web servers. The match field consists of TCP SYN and ACK flags, source IP address (Web server) and destination IP address (end client). The action field instructs OVS to transfer the received packets to Ryu controller by Packet-In method. After building flow entries, Ryu controller sends them to the OVS. Then OVS updates the flow table based on these instructions.

The third component is a function of Ryu controller that sends RST packets to the Web servers which spend more time responding to the end client. The Ryu controller whether or not to send RST packet to web server based on the following two steps. The

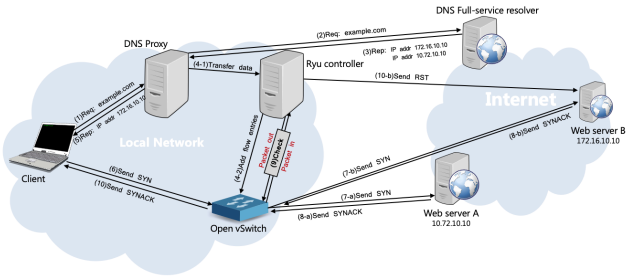

Fig. 7 Detail of mechanism

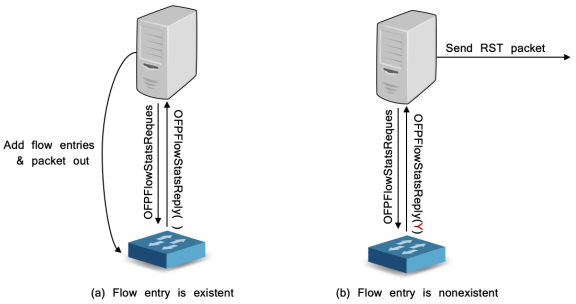

Fig. 8 Flow entry state

first step is that Ryu controller requests OVS to check whether the flow entry exists or not at the flow table according to source IP address, destination IP address, TCP source and destination port numbers. If the answer is empty which means there is no associated flow entry exists, then Ryu controller build flow entries for next packet forwarding which transferring between end client and Web server to OVS by Packet-Out method. If the answer meaning there is associated flow entry exists, then Ryu controller builds RST packet according to the received packet from OVS which transferred by Packet-In method.

Figure 6 shows the network topology of the proposed mechanism. In order to reduce the latency of data transfer in Northbound and Southbound, we configure DNS proxy, Ryu controller and OVS in the same local network. The end client requests and receives DNS query and response directly from the DNS proxy as shown with the red line. The blue line is the process of accessing to the web server after end client received the IP address of the web server from DNS proxy. As shown in Figure 6 , the distance between web server A and end client is shorter than web server B which means the response times from the web server B is longer than web server A. In the following procedure description, we always assume the web server B causes long latency.

3.2 Procedure of the proposed mechanism

Figure 7 illustrates the procedure of the proposed mechanism. The end client directly sends DNS query to the DNS proxy (step 1), and the DNS proxy forwards it to the DNS full-service resolver without any processes (step 2), and as described in section 3.1, the DNS proxy does not provide cache function. The DNS full-service resolver answers the DNS query to the DNS proxy (step 3). When the DNS proxy receives the DNS response, it performs two procedures. The first one is that the DNS proxy sends the information of the received DNS response to the Ryu controller using RESTful API (step 4-1). Then the Ryu controller creates the flow entries based on the informaion received from the DNS proxy and add them to the OVS (step 4-2). The other procedure is that the DNS proxy selects the IP address at the top of the IP addresses list and sends it to the end client (step 5). Then the end client accesses to the web server based on the received IP address (step 6). And then the OVS processes packets from the end client and web servers based on the flow table and instructions from Ryu controller. The detailed of the mechanism about how to judge the fastest responses from multiple web servers is explained in the following.

First of all, when the OVS receives a SYN packet from the end client which is the first step of establishing the TCP connection between the end client and the web server, the OVS clones the packet and modifies the destination IP address according to the actions defined in the group table. Then the OVS transfers the cloned and modified SYN packets to the destination web servers (step 7-a and step 7-b).

Then the OVS receives the corresponding SYNACK packet from both of the two web servers but the one from the the web server A is faster than that of the web server B as expected (step 8-a) which are replied to the end client from the two web servers. Then the OVS encapsulates the headers of the packets and sends to the Ryu Controller by Packet-In method (step 9). Based on the information of the packet headers, the Ryu controller confirms whether the related flow entry has already added on the flow table by sending requests to the OVS with OFPFlowStatsRequest method. This method requests the source IP address, destination IP address, TCP source port, TCP destination port and cookie. Due to this SYNACK packet is being received from web server side for the first time, the reply must be empty, as showed in the Figure 8 (a), then Ryu controller makes the flow entries and populates them to the OVS for transferring the next packets from end clients and Web server and instructs OVS to transfer this SYNACK packet to the end client.

For the other SYNACK packet which is received from the web server B, as showed in the Figure 8 (b), the Ryu controller conducts a different process. In order to prevent the end client from receiving duplicate ACK packets due to the implementation of sending SYN packet to all associated web servers, the Ryu controller sends a RST packet to the web server based on the SYNACK packet (step 10-b). Finally, the OVS transfers the packets to end client (step 10) and only transfers the packets from the web server A which responds faster to end client.

4. Implementation and evaluations

4.1 Experimental local netowrk construction

We implemented a prototype system based on the proposed mechanism. For the DNS proxy, we used the Net::DNS::Dynamic::Proxyserver module (version 1.2) [9] which is a dynamic DNS proxy server and its main task is to receive the DNS queries from the end clients and forwards them to the DNS full-service resolver. Then the DNS proxy also receives DNS responses from the DNS full-service resolver and replies them back to the end clients. For the DNS full-service

resolver, we used Berkeley Internet Name Domain (BIND) [10] for the DNS server software.

For the web server construction, we used the Apache HTTP Server [11] which is an open-source cross-platform web server software. We set up two web servers with a simple web page and the end clients can access them via the OpenFlow switch.

For the OpenFlow switch, we used OVS 2.13.3 and for the Ryu controller we used python 3.7. We used Raspberry Pi 3 models [12] in the experimental network for all the components with 4GB flash memory and 1.2GHz CPU.

4.2 Experiments with latency generation

In order to generate extra latency between the end client and a specific web server, we used tc tool [13] to set 50ms latency between the end client and web server B. The tc tool is used to control the traffic in the Linux kernel. The main parameters which we configured for the web server B are ‘qdisc’ and ‘netem’. The ‘qdisc’ is short for ‘queueing discipline’ and it is the elementary to understand the traffic control. When the kernel needs to send a packet to an interface, it is enqueued to the qdisc configured for that interface. Immediately afterwards, the kernel tries to get as many packets as possible from the qdisc for transferring them to the network adaptor driver. The ‘netem’, which means Network Emulator, is an enhancement of the Linux traffic control facilities that allow to add delay, packet loss, duplication and more other characteristics to the packets outgoing from a selected network interface. In the experiment, we added 50 ms delay to the packets outgoing from the web server B.

With the above exter latency added to web server B, we conducted two experiments on the local experimental network in order to evaluate the prototype system of the proposed mechanism. In the first experiment, we conducted 1000 times of HTTP accesses from the end client to the web servers under the DNS Round Robin and the proposed mechanism conditions respectively. In order to make the end client access the web servers as much as possible with performing DNS name resolution each time, we set the TTL (Time To Live) of the DNS Resource Records in the DNS proxy and used wget tool [14] for conducting web accesses on the end client. The wget tool is a simple program that retrieves the content from the web servers and it also provides the function of downloading the web contents via HTTP protocol from the web servers. The wget tool repeats the processing of content download from the web server, if there is a network problem which makes the download incompletely, until the whole content has been retrieved. In this experiment, we issued 1000 requests in sequence and counted the total execution time in different environments.

In the second experiment, we used the httperf tool [15] to evaluate the stability and measure the throughout for the prototype system of the proposed mechanism per second. The httperf tool provides a variety of workload generators based on the HTTP protocol. Once the httperf tool is applied, it keeps track of several performance metrics that are summarized in the form of statistics that are printed at the end of a test run. We issued 1000 requests for establishing the connections between the end client and the web servers at a rate from 15 requests per second to 24 requests per second. In addition, the parameter of timeout is set to 5 second to wait for the responses.

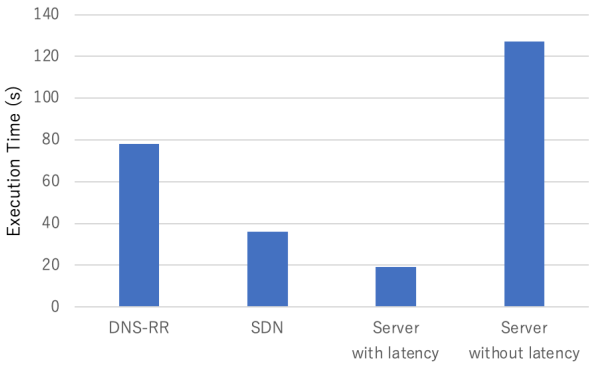

Fig. 9 Result of wget execution

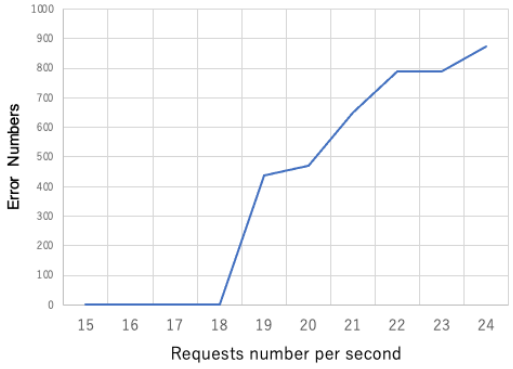

Fig. 10 Result of httperf execution

The evaluation results of the above two experiments are described in the following section.

4.3 Evaluation results

In the evaluations, we focus on the communication overhead between the Ryu controller and the OVS in order to check the performance of the prototype system of the proposed mechanism.

First, Figure 9 shows the total execution time of wget tool run under different conditions. The leftmost bar chart shows the total execution time of the wget tool when DNS Round Robin was adoppted for load balancing. The second bar chart from the left illustrates the execution time of the wget tool when the prototype system of the proposed mechanism is used. From the evaluation results, we can confirm that comparing to DNS Round Robin load balancing, the protytpe system of the proposed mechanism needed low execution time in the same network condition.

In addition, in order to confirm the performance of the prototype system of the proposed mechanism, we conducted the same execution that accessing to the two web servers with and without adding extra latency respectively under the same condition. The results are shown on the right half of Figure 9. Although the prototype system of the proposed mechanism spends more execution time than the measurement, it spends less time than the DNS Round Robin load balancing observably.

Moreover, we also measured the throughput of the prototype

system of the proposed mechanism using httperf tool and Figure 10 shows the evaluation results. As shown in Figure 10, when we sent web access requests with the rate from 15 per second to 18 per second from the end client, there was no any erros or timeout occured. However, as the rate of the web access requests increases, the bottleneck of the OVS which is responsible for replicating SYN packets, rewriting headers of the IP addresses for providing optimal choice to end client and checking related flow entries are available at the flow table is revealed. The hardware specs of the components used in the local experimental network may be the reason of the low throughput and it can be improved by using high spec hardware compnents.

Based on the above evaluation results, we can confirm that the protype of the proposed mechanism worked correctly as designed and the overhead and throughput is acceptable with reasonable overhead.

5. Conclusion

In this paper, we proposed an optimal selection mechanism from multiple IP addresses of a web server for end clients using OpenFow in order to reduce the latency of web access and improve the performance of load balancing for web servers. In the proposed mechanism, when an end client attempts to access a web server with multiple IP addresses the OpenFlow controller makes the OpenFlow switch access the multiple IP addresses simultaneously during the DNS name resolution stage and let the end client access the web server which is near to the end client or under a low workload. We implemented a prototype system of the proposed mechanism using Ryu application and Open vSwitch which are one of the OpenFlow applications and conducted the evaluations. Based on the evaluation results, we confirmed that the prototype system worked correctly as designed and the performance was acceptable with reasonable overhead. In the future work, we consider to add some extra functionalities in order to support DNSSEC [16] and design a redundant network topology to solve the single point of failure issue. Moreover, performance improvement and evaluation in a real network environment are also included in the future plan.

References

[1] T. Brisco, “DNS support for load balancing,” RFC1794, IETF, April 1995.
[2] P. Mockapetris, “Domain names – concepts and facilities.” RFC1034, IETF, November 1987.
[3] P. Mockapetris, “Domain names – implementation and specification.” RFC1035, IETF, November 1987.
[4] Open Networking Foundation, “Software-Defined Networking (SDN) Definitaion” (online), available from https://opennetworking.org/sdn-definition/.
[5] Open Networking Foundation (online), available from ⟨https://opennetworking.org⟩
[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. “OpenFlow: enabling innovation in campus networks.” SIGCOMM Comput. Commun. Rev. 38, 2 (April 2008), 69–74. DOI:https://doi.org/10.1145/1355734.1355746
[7] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, “Advanced study of SDN/OpenFlow controllers,” In Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR ’13). Association for Computing Machinery, New York, NY, USA, Article 1, 1–6. DOI:https://doi.org/10.1145/2556610.2556621
[8] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, “The design and implementation of open vSwitch,” In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI’15). USENIX Association, USA, 117–130.
[9] Jakobs, M. S.: Net::DNS::Dynamic::Proxyserver - A dynamic DNS proxy-server, CPAN (online), available from ⟨https://metacpan.org/pod/Net::DNS::Dynamic::Proxyserver⟩
[10] Internet Systems Consortium, Inc. “BIND 9 - Versatile, classic, complete name server software” (online), available from https://www.isc.org/bind/.
[11] The Apache Software Foundation, Apache HTTP server project(online),available from ⟨https://httpd.apache.org⟩
[12] Raspberry Pi, “What is a Raspberry Pi?” (online), available from https://www.raspberrypi.com/products/raspberry-pi-3-model-b/.
[13] Repogitory, U. M.: tc - show/manipulate traffic control settings(online), available from ⟨https://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html⟩
[14] Free Software Foundation, I.: Wget - The non-interactive network downloader(online), available from ⟨https://linux.die.net/man/1/wget⟩
[15] Mosberger, D. and Jin, T.: httperf — A tool for measuring web server performance, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 26, No. 3, pp. 31–37.
[16] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, “DNS security introduction and requirements,” IETF, RFC4033, March 2005.

