Semi-reactive Switch Based Proxy ARP in SDN

Fauzi Dwi Setiawan Sumadi
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
fauzisumadi@umm.ac.id

Diah Risqiwati
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
risqiwati@umm.ac.id

Syaifuddin
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
saifuddin@umm.ac.id

***Abstract*— In order to achieve high scalability during the network discovery process in software-defined networking (SDN), an extensive method for generating switch-based proxy is essential. This paper investigated the semi reactive solution for guiding the controller to build an OFPT_FLOW_MOD message that allowed SDN switch to reply an Address Resolution Protocol (ARP) request directly by deploying the semi-reactive switch-based proxy ARP application in northbound application programming interface (API). We conduct the experiment by using Open Networking Operating System (ONOS) an open-source SDN controller simulated in Mininet environment. As can be seen from the evaluation result, the installed application can reduce the ARP reaction time up to 95% calculated from the sender host. The final result also indicates that our approach can decrease the controller's loads significantly.**

Keywords— SDN, Proxy ARP, Semi-reactive

I. INTRODUCTION

Recently, SDN has gained a proper concern among researchers because it boosts the innovation's rate in a networking environment. The main concept of SDN is the separation between the control layer known as controller and the forwarding function which can communicate each other regulated by the southbound API [1]. The network administrator can easily determine the controller's functionality by installing an application directly through the northbound API. This mechanism maintains the network scalability since the proprietary issues originated from the forwarding devices are omitted.

In term of the network discovery process in the IPv4 environment, the controller and the SDN switch may suffer because of the abstraction of centralized networking control and management. The controller will be forced to provide a service to forward the ARP requests which are going to be flooded by the forwarding devices. This problem has been reduced by implementing the reactive based application. It can directly generate an ARP reply packet. However, if there are a lot of end host devices that transmit an ARP request at the same time, the controller still receives the impact for crafting and delivering an ARP reply for every incoming request.

Our proposed research focused on developing a semi-reactive application that can response the ARP request by partially offloading the capability to generate the response into the SDN switch. It can be achieved by defining a single flow rule through an OFPT_FLOW_MOD message which can filter all of the incoming requests. We conduct several scenarios to investigate the effects by calculating the ARP response time and the controller's CPU usage.

The remaining section of this paper is organized as follows. Section II is concerned about the relevant literature review of OpenFlow network and the details of ARP processing in SDN. In section III, we discuss the related works that have been performed before, either in proactive or reactive approach. We illustrate the application workflow and the evaluation scenario in brief in section IV, while section V contains the comprehensive analysis of the proposed simulation. We conclude the paper in section VI.

II. BACKGROUNDS

This section expounds the details of the relevant information relating to the paper's topic.

A. OpenFlow Network

One of the prominent southbound API that has been widely utilized in SDN is OpenFlow [2]. It provides a programmatic approach for the controller so it can directly configure the switch's behaviour by defining flow rule for filtering each packet that enters the forwarding device. All of the designed flow rules are stored in the flow table by using OFPT_FLOW_MOD message which also can be used for deleting a particular rule. Each of the specific rules has a traffic selector and also a traffic treatment. If there is no rule that can handle the inbound packet, the SDN switch will encapsulate it by using OFPT_PACKET_IN message then transmits the packet directly to the controller for further action such as the network initialization process. In response, the controller will send OFPT_PACKET_OUT message to the switch for performing the specified action such as broadcast or multi-cast the inbound packet if the controller does not has a legal route between the source and destination or deliver OFPT_FLOW_MOD message for commanding the switch to install the defined flow rule.

B. ARP Processing in SDN

Generally, the ARP processing in SDN is conducted in a reactive manner. The controller installs an application called ProxyARP on its northbound API. Through the default flow rule deployed in SDN switch, this application can collect OFPT_PACKET_IN message contained the incoming ARP request packet and easily break it down by

978-1-5386-8402-3/18/$31.00 ©2018 IEEE

extracting the essential information including the target protocol address (TPA), target hardware address (THA), sender protocol address (SPA), sender hardware address (SHA), VLAN identification (ID), device ID, and the incoming port's ID. Then the ProxyARP will try to check whether the TPA is mentioned on its ARP cache. If the TPA is listed, the controller will directly craft an ARP reply based on the extracted information enclosed in OFPT_PACKET_OUT message which instructs the switch to send the ARP replay directly. However, if there is no information available, the controller will send the ARP request packet encapsulated in OFPT_PACKET_OUT. Subsequently, the switch will broadcast the ARP request, then the targeted host will reply. After the SDN switch receives the ARP reply, it will pass the packet to the controller by using OFPT_PACKET_IN. Therefore, the controller has a complete route between sender and destination host. Then the controller will send an OFPT_PACKET_OUT that intends to tell the switch to forward the packet within the message (ARP replay) back to the sender host. The SDN switch may perform a proactive mechanism for handling the ARP request. The main difference from the previous concept is the processing point. Forwarding device does not have to pass the incoming request since there exists a flow rule that is defined by the network administrator.

III. Related Works

Previously, there were several researchers that focused their efforts on implementing either reactive or proactive Proxy ARP in SDN intended to reduce the ARP broadcast storm. [3]-[7] proposed a reactive based Proxy ARP involving the controller for generating ARP reply. Similarly, to the default ProxyARP in ONOS, the controller will store the details of the available host in a specific table or service by obtaining the incoming OFPT_PACKET_IN message's information that may contain an ARP request or the Link Layer Discovery Protocol (LLDP) packet. If the controller has a specific information regarding the destination address of the current ARP request, it will directly craft an ARP reply without involving the targeted host.

FSDM [3] initiated a caching scheme for storing the incoming DHCP and ARP information which were used by the controller for crafting both DHCP and ARP reply directly based on the cache information without flooding the request extensively on the network. SEASDN [4] provided a hashing mechanism for storing the request packet's information then responded by sending OFPT_PACKET_OUT message containing the reply packet. In the same manner, as FSDM and SEASDN, the authors of [5], [6], and [7] also implemented a reactive approach for handling the ARP request by administering the controller as the ARP proxy. The result from all of the papers indicates that the reactive application can successfully provide scalability to the network. However, this method may exhaust the controller's resource upon receiving a huge amount of requests since it should examine the incoming packets one by one.

Another type of Proxy ARP in SDN is introduced by [8],[9] which deploy proactive scheme for resolving ARP reply generation. [8] extends OpenFlow enabled switch's capability for creating an ARP reply independently while [9] perform offloading mechanism by injecting several flow rules for generating ARP reply packet manually (hard-coded). This method is restricted by the OpenFlow since it cannot determine the TPA and THA of ARP reply packet. In consequent, [9] applies the broadcast address as the default value. A significant problem that can possibly occur is when a new host joins the network causing the network administrator to install the corresponding rule manually.

IV. Proposed Method

Our method intended to create semi reactive mechanism. The proposed approach extended the capability of the current reactive proxyARP application illustrated in figure 1. It could be achieved by partially storing the ARP request information and eventually generating flow rule for the SDN switch. This single rule could handle any incoming ARP request because it could construct the ARP reply's TPA and THA extracted from the incoming ARP request's SPA and SHA. Therefore, the ARP processing would be completely handled by the SDN switch based on the flow rule's selector. The main difference between our method and the proactive Proxy ARP could be clearly pointed during the ARP reply's flow rule generation process.

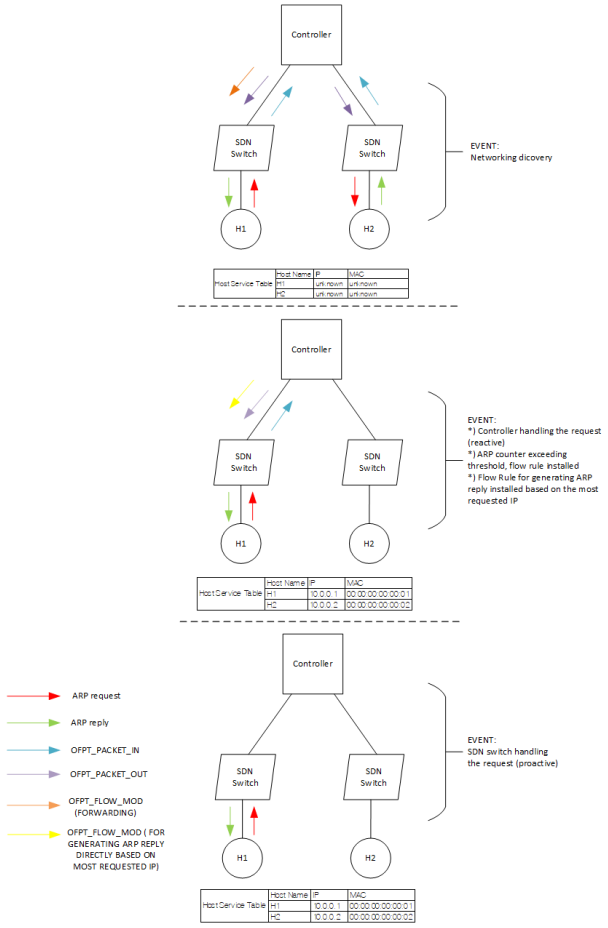

Fig. 1. Semi-reactive proxy ARP scheme

Network administrator requires to manually determine a flow rule for each available host on the network when the proactive proxyARP is implemented. Conversely, the semi reactive approach will assign the flow rule's generation to the application directly which allows it to construct a

single rule based on the information of the most requested IP address during the ARP broadcasting.

V. Simulation Scenario

Our experiment was emulated using Mininet [10] version 2.2.2 which specifically implemented tree topology in OpenFlow network (version 1.3). It consisted of 3 SDN switch which used Open vSwitch [11] (OVS) version 2.5.4, 4 virtual hosts separated on each switch and ONOS [12] version 1.13.0 as the SDN controller. We assumed that ONOS applied the default applications including Forwarding, ProxyARP, Host-Provider, Optical-model, Drivers, Mobility, LLDPprovider, OpenFlow, and OpenFlow-base. The simulation was performed on a single Ubuntu 16.04 PC that had the following specification Intel Core CPU i5 3210M Processor 4 GB DDR3 1600 MHz SDRAM.

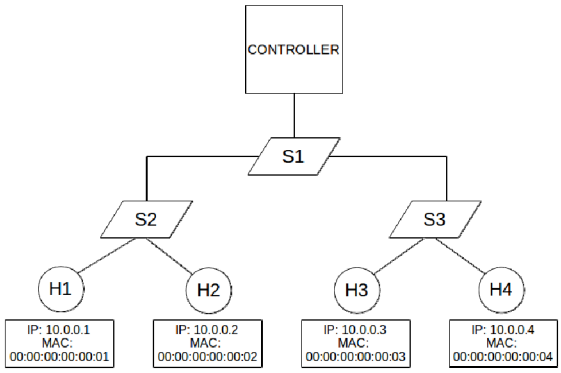

Fig. 2. Simulation topology

We conducted two different scenarios for the experiment. In order to imitate real network environment, the first scenario performed UDP transmission between H2 and H4 that had distinct bandwidth varieties including 2, 50, 100, 500, and 800 Mbps then H1 generated an ARP request to H4 by sending one packet each second. During the UDP transmission, H1 generated a constant ARP request directed to H4 each second. Therefore, the response time between the reply and request could be calculated.

The other emulation scenario performed different ARP request sending rates from H1 to H3 which varied from 500, 1000, 1500, 2000, 2500 packets per second without implementing UDP transmission. The ARP request was originated from H1 which each of the packets consisted of random source IP and MAC address generated by scapy [13] library. The ARP response time could be extracted by subtracting the time when H1 receiving the ARP replay from H3 and H1 sending ARP request to H3. There were two variables that were calculated during evaluation including the controller CPU usage for mapping the controller overhead and the ARP response time for each different sending rates. All of the packets were sent using Tcpreplay [14].

In term of the application's workflow, the controller injected particular flow rule for instructing the SDN switch to transmit incoming ARP packet encapsulated in OFPT_PACKET_IN message. Packet service module in ONOS was used to generate the proposed flow rule and also regulate its priority for only having the standard reactive variable (5). Whenever the SDN switch receiving ARP request packet, it would directly investigate its TPA and THA. If there is a match event between TPA and flow rule's traffic selector originated from the semi-reactive switch proxy application, the SDN switch will craft an ARP reply packet associated with the destination address of the ARP request which is built by imitating the traffic treatment. Therefore, the sender can receive the ARP reply almost real-time without involving the reactive proxyARP application since the semi-reactive application's rule has higher priority.

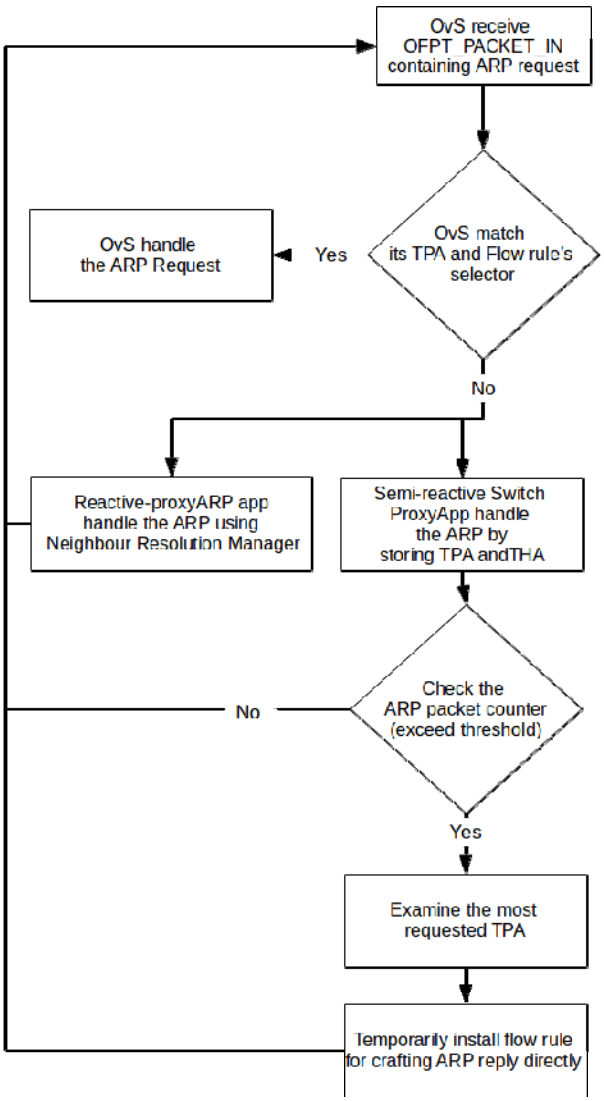

Fig. 3. Workflow of semi-reactive switch proxy application

In contrast, if the TPA doesn't correspond with all of the traffic selector specified in the flow table, OpenFlow switch automatically forward the ARP request to the ONOS controller enclosed in OFPT_PACKET_IN message which is handled by reactive proxyARP and semi-reactive switch-based proxy application. Then the NeighbourPacketManager in ONOS will directly send OFPT_PACKET_OUT message as described in the 'ARP Processing in SDN' section. In addition, semi-reactive switch proxy application breaks down the packet in message for extracting the TPA and THA which are then stored in HashMap for statistic purpose. If the ARP

counter exceeds the defined value by the application, it will inspect the most requested TPA then temporarily installs the corresponding flow rule with the most occurred TPA as the traffic selector and extends the flow for having the highest priority. The details of the installed flow rule are expounded on table 1.

TABLE I. FLOW RULE SCHEME

Traffic Selector	
ARP OpCode	1
ARP TPA	The most requested TPA
Traffic Treatment	
ARP OpCode	2
ETH Source	The most requested THA
ETH Destination	ETH of incoming ARP request
ARP TPA	IP of incoming ARP request
ARP THA	ETH of incoming ARP request
ARP SPA	The most requested TPA
ARP SHA	The most requested THA
Output Interface	Incoming packet's port

Traffic selector is set to accommodate TPA of the ARP request packet. This method will filter only an ARP packet with OpCode 1, specifically targeting the specified TPA. SDN switch responds the match event by generating an ARP reply packet based on the traffic treatment which includes transferring both source MAC and IP address of incoming packet into the destination address of the reply packet (THA, TPA, and ETH destination) and changing the source address with the TPA and THA address (SHA, SPA, and ETH source) extracted from the hashmap which has the most occurrence among other TPA. The HostProvider service should have identified the most requested address. Subsequently, the ARP reply is transmitted back to the sender host via incoming packet's port. This scheme will allow the SDN switch for filtering every ARP request destined to particular host only by a single flow rule.

VI. SIMULATION RESULTS AND ANALYSIS

In term of the effectiveness of the semi-reactive approach, we performed an in-depth analysis between our proposed method and the reactive proxyARP. Therefore, we could infer the ideal scheme for reducing ARP response time significantly. As expected, the results experiment depicted in figure 4 clearly showed a slight difference in the response time of ARP processing between two conditions including without installing the semi-reactive switch application or not. Since the reactive proxyARP maintained the ARP processing, the controller responded rapidly proved by the blue bar exactly at 2.61 ms. It could happen because the Host-provider service in ONOS had already mapped the targeted host after receiving the first ARP request. This method allowed the controller directly to generate ARP replay without sending OFPT_PACKET_OUT that contained ARP request for creating a link between the source and the destination. In contrast, after implementing the proposed application, the ARP response time was dramatically decreased approximately at 0.14 ms indicating that the SDN switch successfully created ARP replay without involving the controller.

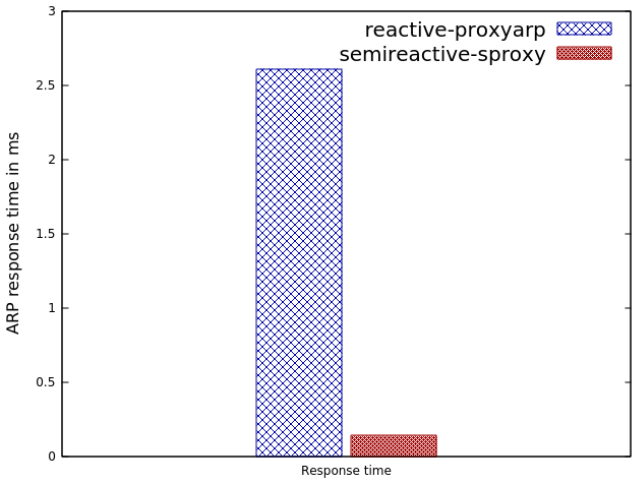

Fig. 4. ARP response time

As far as the first scenario was concerned, the emulation's result during the UDP transmission between H2 and H4 depicted in figure 5, illustrated that the reactive proxyARP still maintained the capability for generating ARP reply encapsulated in OFPT_PACKET_OUT message pointed within 1.5 ms and 2 ms.

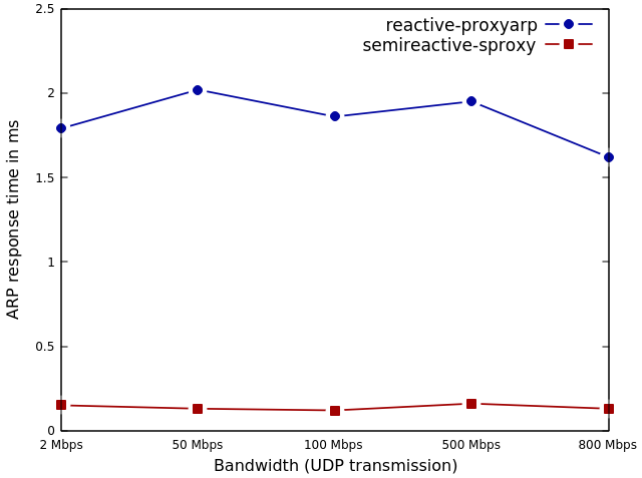

Fig. 5. ARP response time during UDP transmission

Although there was a huge traffics in H4's link, the controller could easily send the replay message to H1 because it had a topological map of the network. On the other hand, the response time of semi-reactive application still pointed in 0.16 ms which still proved that ARP reply was crafted locally by the switch depicted in figure 5. A significant result was shown during the second evaluation scenario. Flooding the controller by using a huge amount of ARP requests which had a different source MAC and IP address could exhaust the controller's resources since it should generate a massive amount of ARP replies within a second interval. The rising trend of ARP response time was described by the figure 6. Along with the growth of the packet sending rate, the response time variable before installing the semi-reactive application showed a gradual rise, which could reach more than 40000ms or 40s.

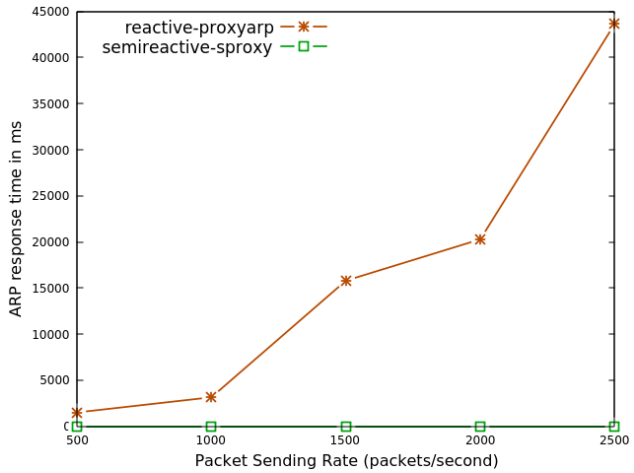

Fig. 6. ARP response time on different packet sending rate

This condition could lead to packet loss event or even worse when the controller didn't have the topological map of the network. In a real network case, a single port on the SDN switch can possibly contain a large number of end-hosts that directly connected in a multi-access network. For instance, a network address 172.16.0.0/22 may contain 1022 hosts on a single subnet. In a worst-case scenario, all of the registered hosts try to contact a single address by sending ARP request. This circumstance will overwhelm the controller which can exhaust the available resources. This problem can be solved by implementing the semi-reactive switch-based proxy application which can handle the problem by directly forwarding the ARP reply packet proven by the result data, approximately below 0.15 ms on each distinct packet sending rate.

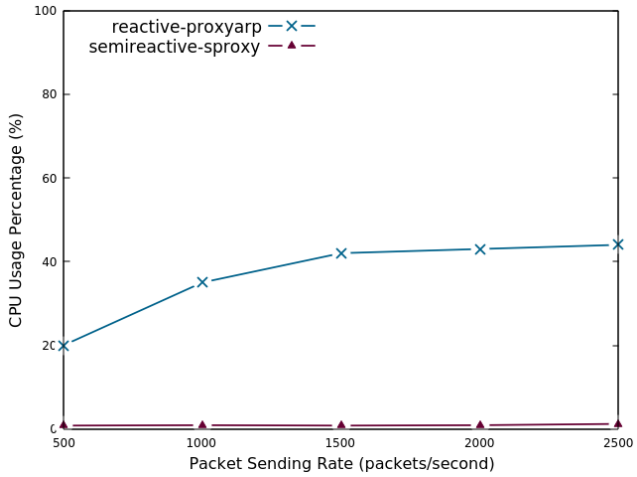

Fig. 7. Controller's CPU usage when receiving a different amount of ARP request

Similarly, a large amount of ARP request could significantly increase the CPU usage of the controller described in figure 7, because it must extract the information of incoming packet as well as transmitting the response packet. The largest packet sending rate could consume more than 40% of CPU usage. This problem might bring the controller into unpredictable state and caused the sender host not to detect an appropriate link to the targeted host (packet loss). However, after the switch-based proxy rule was installed, the CPU usage stayed below 2% indicating that all of the random ARP requests could be responded accurately by the SDN switch.

VII. Conclusions

According to the experiment result, the semi-reactive switch-based proxy application can effectively reduce ARP processing overhead by partially offloading the reply mechanism without involving the controller. This method provides a better option rather than fully hands over the task to the switch. If the network administrator implements the proactive method, whenever a new host entering the network, manual configuration needs to be performed in order to accommodate the request of the new hosts.

Acknowledgment

This work was supported by the Universitas Muhammadiyah Malang.

References

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine,* vol. 51, no. 2, pp. 114-119, 2013.
[2] OpenFlow Switch Specification (Version 1.5.0), O. N. Foundation, 2014.
[3] W. Jian, Z. Weichen, Y. Shouren, L. Jiang, H. Tao, and L. Yunjie, "FSDM: Floodless service discovery model based on Software-Defined Network," in *2013 IEEE International Conference on Communications Workshops (ICC)*, 2013, pp. 230-234.
[4] N. Jehan and A. M. Haneef, "Scalable Ethernet Architecture Using SDN by Suppressing Broadcast Traffic," in *2015 Fifth International Conference on Advances in Computing and Communications (ICACC)*, 2015, pp. 24-27.
[5] C. Hyunjeong, K. Saehoon, and L. Younghee, "Centralized ARP proxy server over SDN controller to cut down ARP broadcast in large-scale data center networks," in *2015 International Conference on Information Networking (ICOIN)*, 2015, pp. 301-306.
[6] L. Jun, G. Zeping, R. Yongmao, W. Haibo, and S. ShanShan, "A Software-Defined Address Resolution Proxy," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017, pp. 404-410.
[7] R. d. Lallo, G. Lospoto, M. Rimondini, and G. D. Battista, "How to handle ARP in a software-defined network," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 63-67.
[8] F. Schneider, R. Bifulco, and A. Matsiuk, "Better ARP handling with InSPired SDN switches," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2016, pp. 1-6.
[9] T. Alharbi and M. Portmann, "SProxy ARP - efficient ARP handling in SDN," in *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*, 2016, pp. 179-184.
[10] Mininet. (Online). Available: Available: http://mininet.org
[11] O. vSwitch. (Online). Available: http://openvswitch.org
[12] ONOS. (Online). Available: http://onosproject.org
[13] Scapy. (Online). Available: http://www.secdev.org/projects/scapy
[14] Tcpreplay. (Online). Available: http://tcpreplay.synfin.net