Hybrid NDP Proxy in OpenFlow Network

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

Ade Rega Susanto
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
aderega12@gmail.com

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

Didih Rizki Chandranegara
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
didihrizki@umm.ac.id

***Abstract*—The implementation of Neighbor Discovery Protocol (NDP) in OpenFlow network was handled specifically by three different methods including the learning switch, reactive proxy, and proactive mechanism. All of the specified approaches still raised problems either consuming the controller's resources or forcing manual adjustment by the network administrator. This paper combined both reactive and proactive manner by deploying an application called Hybrid NDP proxy (HNDPP). The controller stored all information from Neighbor Solicitation (NS) request for generating OFPT_FLOW_MOD message as the guideline for the OpenFlow switch to response and craft the Neighbor Advertisement (NA) based on the defined selector and treatment. The research's results showed the effectiveness of the HNDPP by reducing the number of NS requests that entering the controller up to 20 packets on average. In addition, the waiting time for receiving NA packet lessened approximately below 0.5 ms.**

***Index Terms*—SDN, Hybrid Proxy, NDP, OpenFlow**

I. INTRODUCTION

SDN emerges as an alternative solution for resolving the scalability problem occurred in the traditional network [1]. Current networking technology is restricted by vendor lock proprietary which has distinct hardware configuration. The separation approach on networking control and forwarding function into a different part of layer provides an easiness for the network administrator to manage and administer the traffic based on the defined application on the control plane which is adjusted programmatically using Application Programming Interface (API). The southbound API connects the control plane layer known as controller with the forwarding devices by defining communication standards which allows the controller to define the treatment action for each of the filtered packets such as transmitting to a certain port, generating a reply packet, restricting the network bandwidth, metering the port's connection, or even dropping specific packet based on the filtered headers. In term of the NDP process in IPv6, the controller acts as the centre of path learning for providing the appropriate path between the host that sends NS and its destination. Even though the NDP mechanism does not apply the broadcast communication, the other hosts which already implement the IPv6 will also acquire the impact from the NS request. Therefore, upon receiving a vast amount of request all of the available nodes on the network including the controller, the forwarding device, and the hosts which deploy IPv6 will suffer from the request flooding. There were several methods that have been investigated. The common approach for handling networking discovery in SDN imitates the learning switch application which based on MAC learning. The process for comprehending the MAC address of the destined host forces the NS packet to be multicasted across the network since the controller does not have a capability to generate or reply each of the new incoming requests. Therefore the resource overhead is extremely high when a vast amount of requests entering the network. The other approach is known as the reactive proxy. The controller stores the incoming request information as well as the reply packet in order to directly generate the relevant response through OFPT_PACKET_OUT message. Therefore the other hosts do not have to deliver the NA packet which also can reduce the request's waiting time for the sender host. This method may possibly bring the controller to use all of the available resources when a huge amount of NS packets are transmitted by the SDN switch. The last method called proactive proxy is deployed by manually adjusting the flow table for directly generating the NA packet. However, this method obligates the network administrator to append new flow rule for every host joining to the network which systematically similar to the traditional network (manual configuration). The authors propose a new method that combines both reactive and proactive mechanism which called HNDPP for managing the NS requests effectively without overwhelming the controller's resource as well as reducing the response waiting time. This method was conducted by installing an application on the controller northbound API. HNDPP has a capability to store NS's header information which then generates a response to produce OFPT_FLOW_MOD containing a flow rule. The defined flow rule was used by the SDN switch to reply the

This paper supported by the Universitas Muhammdiyah Malang.

filtered NS.

II. BACKGROUND

A. Southbound API OpenFlow

One of the well known southbound API that has been widely implemented in SDN is OpenFlow [2]. This standard provides guidelines for the controller to communicate with the forwarding devices [openflow standard]. There are some message's types that are important for the networking control including the OFPT_PACKET_IN, OFPT_PACKET_OUT, and OFPT_FLOW_MOD. Each of the mentioned packets has a particular purpose. The packet in message is utilized by the SDN switch for redirecting the incoming packet that is not filtered by any available flow rule on its flow table. Subsequently, the controller will inspect the packet's header for specific matter corresponded to the installed application on the controller such as the Intrusion Prevention System (IPS), Intrusion Detection System (IDS), routing, switching, load balancing, and another network managerial application. In response after the packet prepossessing, the controller produces a message for instructing the forwarding devices to follow the specified action that is encapsulated either in packet out or flow mod message. The OFPT_PACKET_OUT message is used by the controller for commanding the SDN switch to forward the defined packet directly through the kernel space of SDN switch using the flood action. The other response packet called OFPT_FLOW_MOD describes the flow rule structure that should be installed by the forwarding device on its flow table. The treatment part of the defined flow rule may contain some action such as forwarded to the specific port, flooded, or even generated a particular packet. The forwarding devices can also send a packet message for informing the controller about a particular event that may need appropriate concern through OFPT_ERROR_MSG containing specific type and code for different error source. For example, the error code OFPFMFC_ALL_TABLES_FULL indicates a fully-loaded flow table's capacity of the forwarding device.

B. NDP Comprehension in SDN

Generally, there are three main methods for resolving network discovery in OpenFlow environment. Whenever a new host intends to recognize the destination MAC address of the other node it directly sends the NS packet for this purpose. It is destined to the multicast address which is a combination of the last 24 bits of the targeted host and the solicited-node multicast address [3]. While the MAC destination address contains the IPv6 multicast MAC address based on Internet Assigned Numbers Authority (IANA) standard. The crafted packet will be disseminated to all hosts that implement IPv6. Furthermore, the NDP processing in SDN is expounded below.

1) Learning Switch: Generally, there are several methods for handling the NS request in SDN environment. The prominent approach was designed to imitate the traditional switch capability for leaning the MAC address of available hosts on the network through MAC learning stored on its table. As can be seen from the Fig 1, the controller is functioned as the centre of MAC learning instead of involving the forwarding devices to learn the networking path. The controller by default installs the learning switch application. It has a particular data structure for comprehending and storing the packet header's information such as the MAC address, the Datapath Identifier (DPID) as well as the Datapath's port of SDN switch connected to the sender host, and Internet Protocol (IP) address. All of those variables are processed whenever the controller receives MAC learning packet such as the Address Resolution Protocol (ARP) and NS/NA packet. In term of the NDP in IPv6 environment, when the sender host transmits the NS request for learning the destined host MAC address, the controller will directly receive the incoming packet encapsulated in OFPT_PACKET_IN message. It comprehends the packet's DPID, switch's port, and MAC address respectively saved on its MAC learning table using a hash table or dictionary. Subsequently, the controller transmits an OFPT_PACKET_OUT message containing the NS packet which then is forwarded to the destined host from the Datapath's kernel space in a multicast manner. After the targeted host receives the NS packet, it will craft a NA reply which is being redirected to the controller for learning purpose. All in all, the controller will have a complete networking map between the two communicated hosts saved on its local database. In response, the controller will send an OFPT_PACKET_OUT message containing the NA reply to the sender host as well as instruct the SDN switch for installing the OFPF_FLOW_MOD message. The installed flow rule will be used by the forwarding devices for transmitting the packet based on the MAC address filter to the destination.

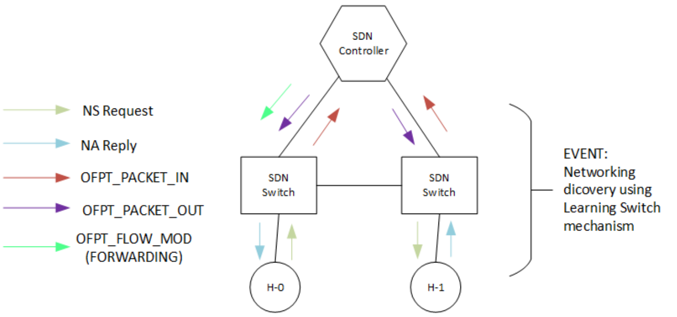

Fig. 1. Learning switch application in SDN [9]

2) Reactive Proxy NDP: The major difference between learning switch application and the reactive based proxy is the NA generation process. In order to minimize the networking path as well as shortening the waiting time for receiving the NA packet, the reactive application offers a mechanism where the controller can directly produce the reply packet which then is being sent by using OFPT_PACKET_OUT message to the sender host described by the Fig 2. The mentioned operation above is conducted after the controller has the access link mapping between the host and the destination. Therefore, it can specify the target host's MAC and IP address. This method will reduce the resource usage of the devices residing in the control and data plane layer since it implements unicast communication instead of multicast.

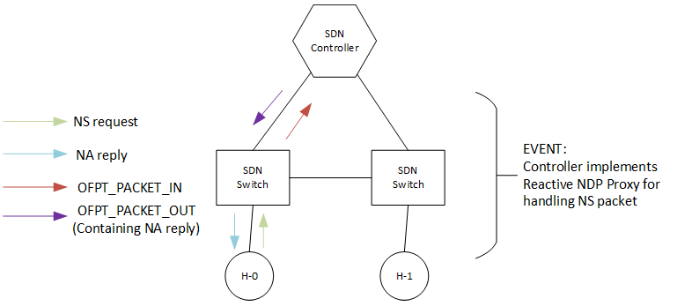
SDN Controller
NS request
NA reply
OFPT_PACKET_IN
OFPT_PACKET_OUT (Containing NA reply)
SDN Switch
SDN Switch
EVENT: Controller implements Reactive NDP Proxy for handling NS packet
H-0
H-1

Fig. 2. Reactive NDP proxy in SDN [9]

3) Proactive Proxy NDP: The third method which still imitates the traditional network configuration, is called the proactive mechanism. The network administrator performs manual adjustment by accessing the console panel of the OpenFlow switch to insert a particular flow rule. The installed flow will directly filter specific packet based on the destination IPv6 and MAC address. Subsequently, the forwarding device crafts the NA packet containing the exact structure of the reply packet generated by the destined hosts. Finally, the SDN switch sends the packet through its kernel space back to the sender host. In the OpenFlow network, this approach can be performed by executing 'ovs-ofctl add-flow' command. The standard example of 'ovs-ofctl' command for IP fe80::200:ff:fe00:2 is illustrated in Table 1. The flow match members include the match of ICMPv6 packet, the destination MAC, the destination IPv6, and ICMPv6 type. While the flow action section contains the command for setting up the source MAC, the ICMPv6 type, the ICMPv6 code, the source IPv6, moving the source MAC to the destination MAC, IPv6 source to the IPv6 target address, IPv6 destination to the IPv6 source address, and send the packet directly from the input port of the SDN switch.

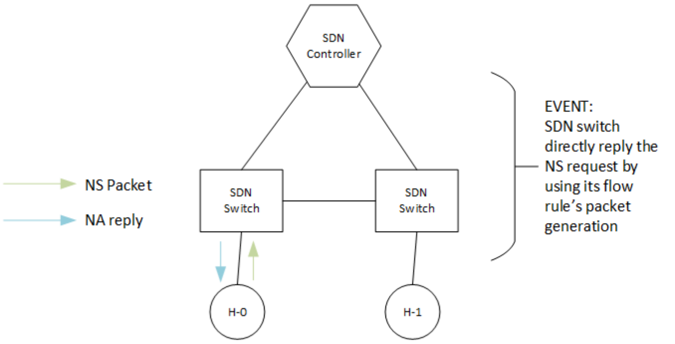
SDN Controller
NS Packet
NA reply
SDN Switch
SDN Switch
EVENT: SDN switch directly reply the NS request by using its flow rule's packet generation
H-0
H-1

Fig. 3. Proactive NDP proxy in SDN [9]

TABLE I
PROACTIVE RULE'S STRUCTURE

	Flow-Match
1	icmp
2	icmp6
3	eth_dst=33:33:ff:00:00:02
4	ipv6_dst=ff02::1:ff00:2
5	icmpv6_type=135

	Flow-Action
1	move:NXM_OF_ETH_SRC[ ]->NXM_OF_ETH_DST[ ]
2	set_field:00:00:00:00:00:02->eth_src
3	set_field:136->icmpv6_type
4	move:NXM_NX_IPV6_DST[ ]->NXM_NX_ND_TARGET[ ]
5	set_field:0->icmpv6_code
6	move:NXM_NX_IPV6_SRC[ ]->NXM_NX_IPV6_DST[ ]
7	set_field:fe80::200:ff:fe00:2->ipv6_src
8	IN_PORT

*Using the Nicira Entended Match (NXM)

III. RELATED WORKS

Previously, there were several related types of research that performed analytical experiment for implementing either reactive or proactive scheme in order to handle the Address Resolution Protocol (ARP) as well as NDP processing. The common approach intended to shorten the packet networking journey for comprehending the destination MAC address reactively. This method were experimented in DR-ARP [4], SEASDN [5], Centralized ARP proxy [6], SDARP [7], and ARP-aware SDN controller [8]. Each of the mentioned related works installed an application which acquired capabilities to identify and store topological map of the network based on the MAC address resolution from both request and the reply packet. The application would comprehend the MAC address, the DPID of SDN switch which directly connected to the requester, and also the SDN switch ingress port. Therefore, upon receiving the similar request, the controller could reactively response by deploying an OFPT_PACKET_OUT message containing the ARP reply packet to the source node. In term of the IPv6 environment, there was also similar research which deployed controller based NDP proxy [9] for resolving the NDP. However, the implemented method still caused problematic event whenever a vast amount of requests processed by the controller. The controller would be overwhelmed for using all of its resources to generate the reply packet which already stated in [9]. In response, Sproxy [10] was introduced by its authors for solving the centralized logic control in SDN. The authors proposed a proactive method by commanding the SDN switch to install the appropriate flow rule which could be used by the SDN switch to directly replay the request packet. Similarly, the scalability problem related to the manual configuration or standalone application outside the controller's application layer in the proactive approach still may possibly be occurred when a new host tries to join the network which forces the network administrator to provide appropriate rule for the new host or install the proactive application on each available programmatic SDN switch (Open vSwitch).

To address the specified problem occurred in both reactive and proactive mechanism, this paper tried to combine those approaches reactively by deploying a modified proxy application that had a capability for storing the MAC information extracted from either ARP, NS, or NA packet. Therefore the controller would have a thorough view where exactly the position of each of the active hosts. The modified method was built in python based environment. There were two main components in HNDPP application including (i) the MAC learning section and (ii) the proactive response depicted in Fig 4. Section (i) in HNDPP was extending the NS processing in a reactive manner which then statistically examined the most requested address from the incoming NS packets within a certain amount of time (10s) through comprehending the MAC address extracted from both NS and NA packet. During the comprehension process, the HNDPP performed traditional learning process by sending out OFPT_PACKET_OUT message containing the request packet to its destination in order to obtain the networking path as well as the MAC address of the targeted node. Therefore, the controller could define the appropriate NA structure. Upon receiving the same request for the most wanted MAC address, the proactive response section (ii) produced an OFPT_FLOW_MOD message containing the flow rule scheme for matching NS packet based on the most requested address as well as generating action to craft the NA packet based on the specified packet's structure. After successfully installed the defined flow rule, the SDN switch could independently reply the most requested MAC address using the packet generation scheme determined by the HNDPP based on the OpenFlow standard. This method could significantly reduce the controller overhead upon receiving a vast amount of NS request within a short period of time. The scalability problem that occurred on the others approach for exhausting the networking node resource was resolved by locally answering the NS request from each of the connected SDN switch.

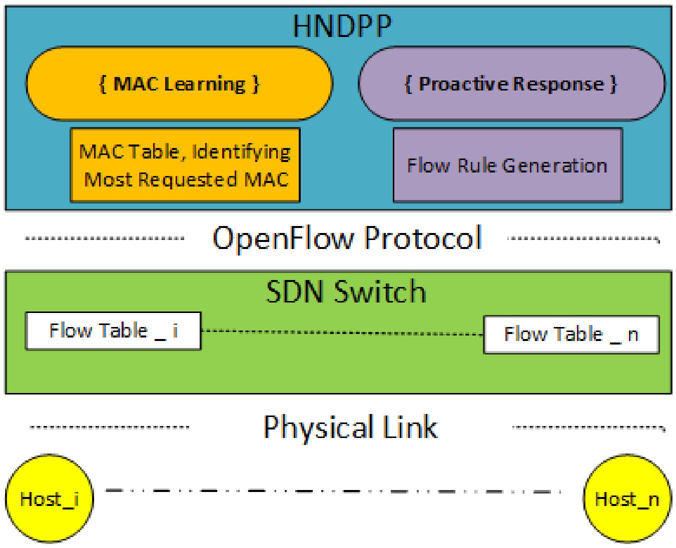

Fig. 4. Block diagram of SDN after implementing HNDPP application

IV. EVALUATION SCENARIO

The experiment was emulated using Mininet [12] version 2.2.1 network emulator generated in Linux based OS using the following PC specification:

- CPU: Intel Core i5 3210M
- Memory: DDR3 1600 MHz SDRAM, 4 GB

In order to create a vast amount of NS request, Tcpreplay [16] was used for maintaining the packet sending rates which varies on a different level. The packet generation was done by utilizing Scapy [15]. In term of the network topology, Fig 5 explains the details of networking nodes. The network was implemented in tree topology consisted of four SDN switch deployed virtually using Open vSwitch [13] (OvS 2.11) with three active hosts attached and the controller node which emulated using RYU [14] 4.3. All of the listed hosts were IPv6 enabled devices which required NDP mechanism for resolving the others MAC address through the multicast transmission. The experiment scenario was conducted for analyzing both controller's overhead and the NDP comprehension process after received distinct amounts of NS request varying from 500, 1000, and 2000 packets per second. The NS request originated from H-1 to H-4. Each of the request packets consisted of randomly generated source IP and MAC address in order to make the SDN switch recognizing the incoming packet as the newly distinct packet.

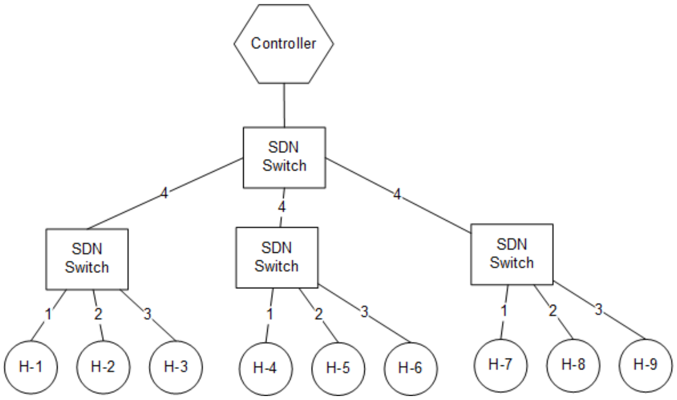

Fig. 5. Experiment's topology

V. EXPERIMENT RESULT

An analytic experiment was performed for investigating the impact of NS request varies between 500-2000 packets per second in RYU controller. The evaluation scenario compared the results before and after implementing the HNDPP which included an analysis of learning switch application, reactive based Proxy NDP, and the HNDPP, therefore, the most effective method could be inferred.

The number of OFPT_PACKET_IN during the simulation which contained the NS packet depicted on Fig 6. The blue line represented the outcome of NS request while only deployed the learning switch application. The significant rise illustrated during the 1000 NS packet request per second which produced more than 450000 OFPT_PACKET_IN in total. However, along with the growth of packet sending rate, the number of packet in message was slightly decreased indicating that the RYU controller was actively processed the incoming NS request by generating the OFPT_PACKET_OUT message contained the incoming NS packets directed to the destined host. The similar pattern displayed on the red line after installing the reactive proxy NDP. Since the reactive approach could truncate the NS networking journey by directly replying the incoming NS using NA packet encapsulated in OFPT_PACKET_OUT message, the total number of OFPT_PACKET_IN that penetrated the controller was reduced approximately about 88%. During the 2000 NS request per second, the reactive proxy NDP could not handle all of the incoming OFPT_PACKET_IN message because the controller should reply by deploying OFPT_PACKET_OUT. Conversely, the HNDPP could suppress the number of packet in message about 20 packets in average for all scenarios by partially dividing the NA responding mechanism to the SDN switch which supported the OpenFlow protocol. The southbound protocol allowed the forwarding devices which directly connected to the originated NS's Host for crafting the NA packet based on the specified treatment generated by the HNDPP application.

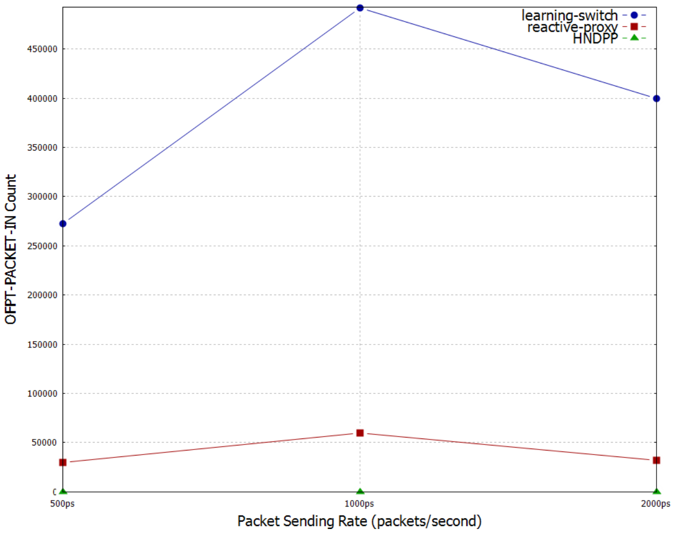

Fig. 6. Total number of packet in message during the simulation

In term of the controller's overhead, the CPU usage of RYU controller during the simulation was also being investigated. Although the CPU percentage did not show a significant rise upon receiving a huge number of NS request, the results showed a similar trend for each of the proposed research's scenarios depicted in Fig 7. The highest value generated by the learning switch application upon receiving NS request for all sending rates types which resided at 25% on average. Since the RYU controller did not implement high modularity for its code documentation [SDN Controllers: A Comparative Study], the vast number of NS request could not significantly affect the CPU usage even though the rates reaching 2000 packets per second. However, the other controller's vendor (ONOS and OpenDayligth) which supports distributed mechanism may receive comprehensive impact throughout all of the provided services leading to the increase of its CPU usage. The reactive based application could restrain the CPU percentage during the 500 packets per second requests. In spite of that, an enormous number of requests still overloaded the CPU utilization proving that the controller focused for responding all of the incoming requests using all of the available resources. Instead of using the controller as corresponding node for generating a reply, the HNDPP application directly commanded the SDN switch for crafting the NA packet which could highly reduced the CPU usage of the controller validated by the controller's CPU usage value remained below 3%.

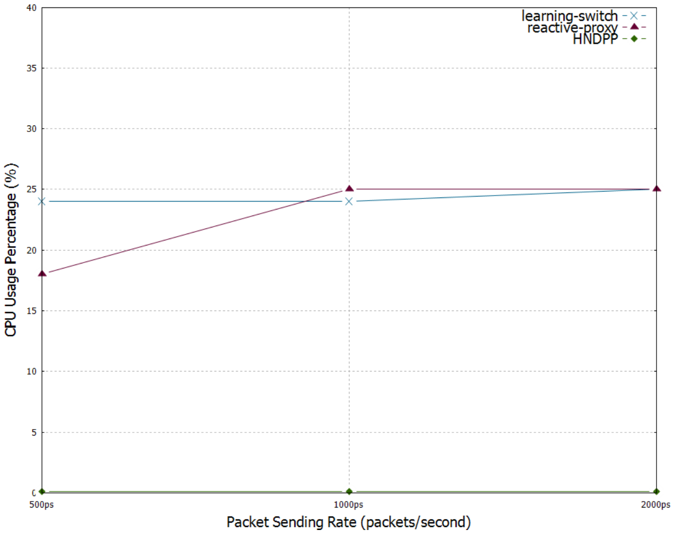

Fig. 7. Controller's CPU usage during the simulation

As far as the NDP process is concerned, the experiment also quantitatively calculates the effects of implementing the HNDPP application in term of the replying action for the incoming NS requests. As can be seen from the Fig 8, the graph describes the time of NS receiving its reply which is the NA Packet either directly from the destined host, the controller, or the SDN switch based on the installed application. The learning switch application from the controller produced the longest time for obtaining the NA packet since the application forced the targeted host for responding the NS request demanding a long networking trip as well as overloading the controller for prepossessing both NS and the NA packet. The results pointed at more than 350000 ms at the time when the controller obtaining 2000 NS packets request per second. The deployment of the reactive proxy application was shortening the networking route along with the time for the sender host receiving the NA packet. Similarly, the vast amount of NS request about 2000 packets per second overwhelmed the controller for either managing the NS packet or crafting and sending the corresponding NA packet proven by the graph pointed at more than 34000 ms. The effectiveness of installing HNDPP application was also being described in Fig 8. The application instructed the SDN switch for generating the reply packet using the OFPF_FLOW_MOD message upon receiving the NS from the sender host. This method successfully minimized the waiting time for each of the generated requests to receive the response confirmed by

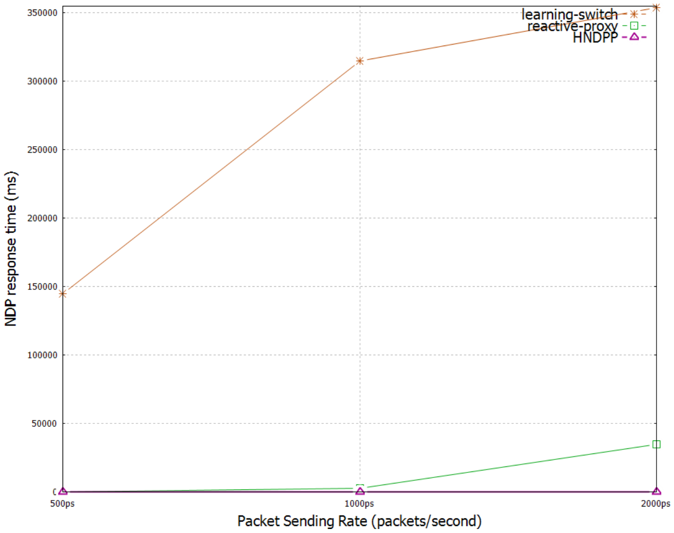

Fig. 8. The waiting time for the requester host to receive NA

the average response time pointed below 0.5 ms since the SDN switch can reply the incoming NS packet locally without involving the controller. In addition, the HNDPP approach

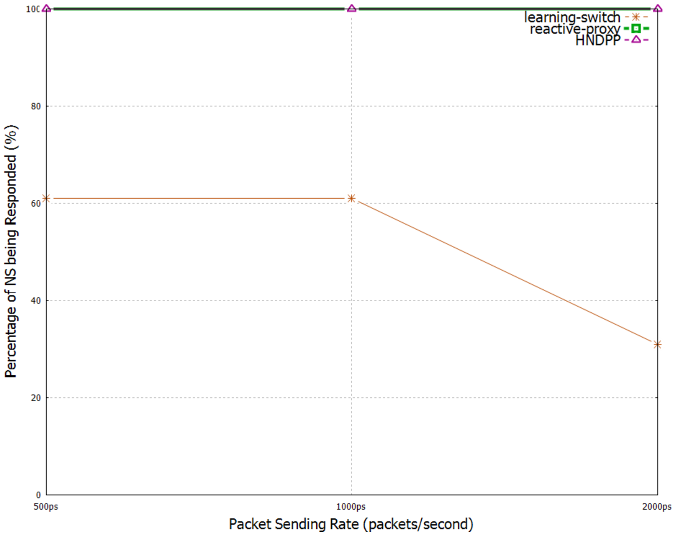

Fig. 9. The number of NS request being replied in percentage

could be beneficial for the network initialization process for all available hosts. Moreover, the percentage of the transmitted NS packet also was also being analyzed. The learning switch application could not produce maximum percentage since there were a lot of requests generated within a second period which caused the controller being overwhelmed for comprehending the incoming NS originated from either the originated host or the targeted host. This circumstance created congestion on a link between the controller and the SDN switch which caused the dropping number of NS packet being responded in percentage pointed at below 35% in average during 2000 NS packet requests per second. The implementation learning switch and HNDPP application successfully answered all of the incoming requests along with the growth of the request's rate even though there was a vast difference regarding its waiting time.

VI. CONCLUSION

In conclusion, the HNDPP application could resolve the scalability and centralized control problems occurred during the network initialization in IPv6 environment by transferring the reply process directly to the SDN switch, therefore, it would craft the NA packet locally. The future milestone for this project will extend the capability of SDN switch OpenFlow enabled protocol to allow another development of unspecified IPv6 packet's header on the current OpenFlow version.

REFERENCES

[1] N. Feamster, J. Rexford, and E. Zegura, ”The road to SDN: an intellectual history of programmable networks,” SIGCOMM Comput. Commun. Rev., vol. 44, no. 2, pp. 87-98, 2014.

[2] OpenFlow Switch Specification (Version 1.5.0), O. N. Foundation, 2014.

[3] Deering, S. and R. Hinden, ”Internet Protocol, Version 6 (IPv6) Specification”, RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>

[4] M. Matties, ”Distributed responder ARP: Using SDN to re-engineer ARP from within the network,” in 2017 International Conference on Computing, Networking and Communications (ICNC), 2017, pp. 678-683.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] F. D. S. Sumadi and D. R. Chandranegara, ”Controller Based Proxy for Handling NDP in OpenFlow Network,” 2018, Reactive application; SDN; NDP; IPv6 p. 8, 2018-11-10 2018.

[10] 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.

[11] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, ”SDN controllers: A comparative study,” in 2016 18th Mediterranean Electrotechnical Conference (MELECON), 2016, pp. 1-6.

[12] Mininet. (Online). Available: Available: http://mininet.org

[13] O. vSwitch. (Online). Available: http://openvswitch.org

[14] RYU. (Online). Available: https://osrg.github.io/ryu/

[15] Scapy. (Online). Available: http://www.secdev.org/projects/scapy

[16] Tcpreplay. (Online). Available: http://tcpreplay.synfin.net