**KINETIK**, Vol. 4, No. 1, February 2019, Pp. 55-62
ISSN : 2503-2259
E-ISSN : 2503-2267

# Controller Based Proxy for Handling NDP in OpenFlow Network

**Fauzi Dwi Setiawan Sumadi*[1], Didih Rizki Chandranegara[2]**
[1,2]Universitas Muhammadiyah Malang
fauzisumadi@umm.ac.id *[1], didihrizki@umm.ac.id[2]

***Abstract***

*A significant method should be deployed in OpenFlow environment for reducing the complexity during the implementation of IPv6 neighbor discovery protocol (NDP) in multicast manner. This paper was performed for deploying reactive-based application in controller's northbound layer for handling as well as cutting the Neighbor solicitation packet's journey. The application had a capability for storing each of the incoming Neighbor Solicitation (NS) and Neighbor Advertisement (NA) packet information. Therefore, the controller could reply the NS packet directly by using OFPT_PACKET_OUT message that contained the NA packet extracted from the reactive application. The experiment's result showed that the proposed approach could reduce the NS response time up to 71% than the normal result produced by the traditional/learning switch application.*

***Keywords:*** *IPv6, Neighbor Discovery Protocol, Reactive Application, Software Defined Networking*

## 1. Introduction

Software Defined Networking (SDN) is considered to be a networking paradigm that overcomes the traditional network's problems by separating the controlling and the forwarding functions into two distinct layers [1]. The separation approach is implemented as an alternative for extending the innovation rate and the networking scalability without bounded by the proprietary vendor which becomes the main problem in the traditional network. Vendor lock between either intermediate or forwarding node restricts its capacity for delivering the proposed data seamlessly, because each of the device's vendor has a unique configuration scheme that differs each other. A significant problem may occur when there is a link-down event. Network administrator should response in extensive manner by manually configurating the broken link or device. Along with the growth of networking problems, the possibility of misconfiguration/human error is inevitable. Therefore, SDN emerges as the fundamental solution which can diminish the stated problem by implementing centered control system resided in control layer.

Systematically, in SDN, the controlling function is handled by the controller while the forwarding devices including SDN enabled switch and router behaves for transmitting the processed data based on the specified rule. The two functional layers can communicate to each other through Southbound Application Programming Interface (API). It regulates the standard and procedure in order to maintain the complexity between them. One of the most prominent Southbound API that have been widely utilized among well-known vendor is OpenFlow [2]. The OpenFlow enabled forwarding devices will automatically support the flow rules initiation which consist of the controller's specified command for giving the forwarding devices a capability for filtering the incoming packet based on the selector's condition and performing the corresponding action such as being forwarded to another port, blocked, dropped, or even creating the reply packet.

In term of the networking discovery implemented in IPv6 environment, the controller functionates as the main node for generating the extensive network's topological map as well as the gateway or shortcut for locating the destined host's address. This method was regulated by NDP which started by sending an initiation packet called NS from the sender host to the destination host intended to comprehend the Media Access Control (MAC) address of the targeted host. The network layer address of the destination host specified from the sender host, is either a combination of the multicast address and the network layer address of the targeted host or not [3]. This mechanism restricts the IPv6 enabled device not to disseminate the whole network by using NS packet in term of mapping the network. Practically, in SDN, the default

*Receive November 01, 2018; Revise November 08, 2018; Accepted November 09, 2018*

application installed in the control plane’s Northbound Application Programming Interface (API) is a learning switch application which regulates the forwarding devices for behaving as a regular traditional switch. Upon receiving the NS packet, this application will learn and store the incoming NS information then gives an access to the forwarding device for transmitting the NS to its destination through OFPT_PACKET_OUT message within its multicast group. Subsequently, the targeted host will response by sending NA packet which consists of the targeted host’s detailed information including its MAC address. Then the controller will have a clear map between host and the destination which is used to determine MAC based flow rule for handling the packet transmission process. Therefore, the controller does not have to receive the NS packets which have a similar characteristic as the flow rule selector. This approach will be automatically repeated as long as there is no flow rule that can filter the incoming NS.

Based on the defined problem above, the proposed research is focused for creating a controller-based application or the reactive mechanism, that can reduce the multicasting process significantly. The authors offer a notion for installing controller-based proxy NDP which allows the controller to reply the NS packet directly without commanding the forwarding devices to multicast the NS. This method will reduce the networking loads that may occur during the NDP process as well as decrease the NS response time generated from the sender host. The proxy NDP will automatically response the NS if there is a thorough information on its MAC table, which is used for crafting the NA packet. Previously, there wasn’t any similar research topic that has been performed. However, the reactive method has been implemented in the IPv4 environment which aimed for decreasing the network load occurred from the Address Resolution Protocol (ARP) broadcasting. Generally, all of the former research including SEASDN [4], FSDM [5], SDARP [6], [7], [8], [9], [10], [11], [12], [13], and [14] intended to implement reactive based approach in SDN for handling ARP broadcasting by deploying an application that can store any information of the incoming ARP request into an ARP table that maps between MAC and IP (Internet Protocol) address. If there exists an information regarding to the ARP request, the controller will directly reply by sending an artificial ARP reply packet which has similar detail as the packet originated from the targeted host through an OFPT_PACKET_OUT message. Our method also implements reactive based proxy in IPv6 environment for handling the NDP. We extend the fundamental capability of previous research for handling ARP broadcast by having a capability for storing information corresponding to the forwarding devices therefore the controller can specifically pinpoint the devices that has a direct link to the host which generates NS packet.

The structure of this paper is expounded as follows, section 2 concerned about the detail of the research method that is used during the experiment. Section 3 will mainly discuss about the experiment’s results as well as perform an in-depth analysis while section 4 contains of the research’s conclusion and the possible recommendation for future experiment.

## 2. Research Method

Generally, the default forwarding application that is utilized by the controller is the traditional switch. It performs a basic learning switch mechanism by multicasting the incoming NS if there is no appropriate MAC saved on switch’s MAC table. The detail of its work flow is described in Figure 1. If there is a host trying to learn MAC address of another end node in IPv6 environment, it will directly send the NS packet to the SDN-switch. Subsequently, the forwarding device will check whether the incoming NS’s header matched with the available flow rule’s selector or not. If there is a matched event the switch/router performs the specified action derived from the flow rule. In contrast, when there is no traffic selector that can filter the NS’s header, the SDN-switch forwards the NS packet to the controller for saving the NS’s header information specifically stored in MAC table encapsulated in OFPT_PACKET_IN message. Upon receiving the packet in message, the controller will save the appropriate information then forward the NS packet encapsulated in OFPT_PACKET_OUT in multicast manner. It can be received only by the available hosts that implement IPv6.

The targeted host response the NS packet by using NA that contains of its corresponding MAC address which then being forwarded by the switch/router to the controller using packet in message. Therefore, the controller will have an information of the networking map between sender and the destination host then sends two responses immediately, including OFPT_PACKET_OUT that contains NA packet and OFPT_FLOW_MOD that is used for installing a flow rule at the forwarding devices based on MAC address. This flow rule is deployed for handling the following packet originated from the sender host and destined to the targeted host.

A significant problem that may occur is the NS response time as well as the controller resources. In a worst-case scenario, when there are a lot of end nodes need to comprehend each other's MAC address, the SDN switch and the controller will be overwhelmed by the NS packets which may lead worse the controller into unstable condition or even worse the packet loss event.

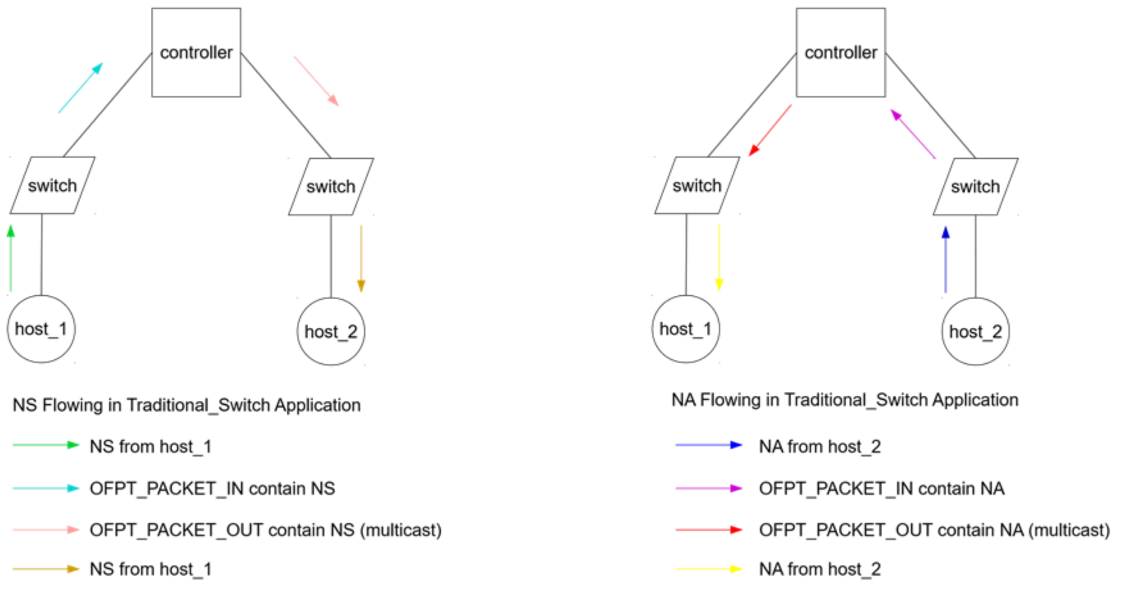

controller
switch
switch
host_1
host_2
NS Flowing in Traditional_Switch Application
NS from host_1
OFPT_PACKET_IN contain NS
OFPT_PACKET_OUT contain NS (multicast)
NS from host_1
controller
switch
switch
host_1
host_2
NA Flowing in Traditional_Switch Application
NA from host_2
OFPT_PACKET_IN contain NA
OFPT_PACKET_OUT contain NA (multicast)
NA from host_2

*Figure 1. The Workflow of Traditional Switch in SDN*

The paper came out with a notion for building an alternative solution reactively by deploying the reactive application in controller's northbound API. Instead of forwarding the NS packet, the application would try to reply the request directly based on the stored information of the incoming NS request by following the OpenFlow standard. The Figure 2 below illustrates the reactive application work flow that has been used during the simulation.

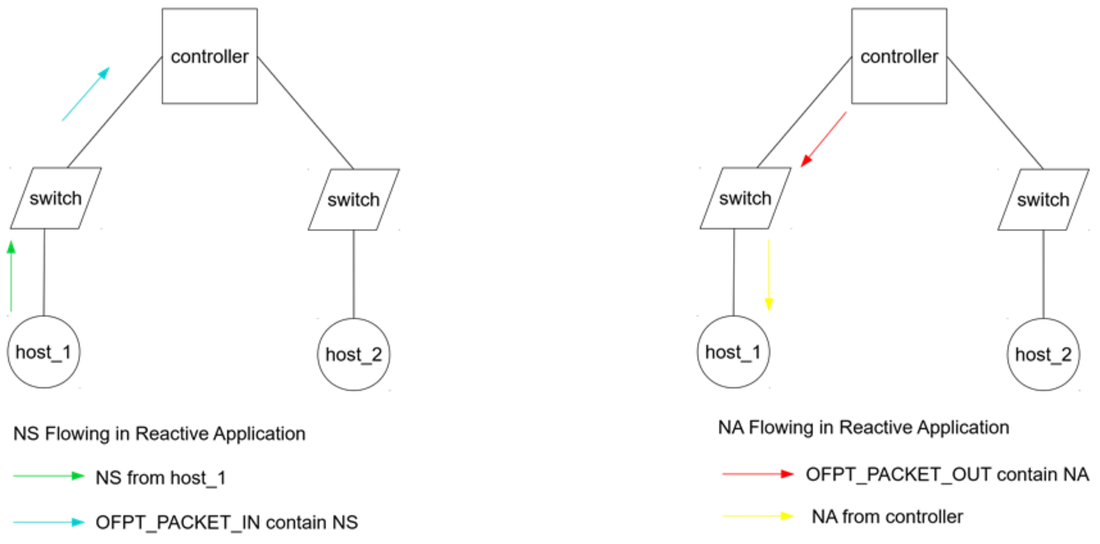

controller
switch
switch
host_1
host_2
NS Flowing in Reactive Application
NS from host_1
OFPT_PACKET_IN contain NS
controller
switch
switch
host_1
host_2
NA Flowing in Reactive Application
OFPT_PACKET_OUT contain NA
NA from controller

*Figure 2. The Workflow of Reactive Proxy NPD in SDN*

As can be seen from the Figure 2, the controller will shorten the networking flow from the previous traditional switch mechanism. It could be achieved by implementing two major components that was used for crafting the NA directly including the host's MAC to host's IPv6 address table and host's IPv6 to switch Datapath Identification (DPID). Upon receiving the first

NS packet from the sender host, the controller would behave the same as the traditional switch. It stored both sender and destination host's information including MAC and IPv6 address as well as their corresponding switch's DPID. After receiving the similar NS packet, the controller had already stored the NS's header information as well as its targeted host's information. Therefore, the controller could directly craft the NA packet based on the available information stored in the specified table. The details of the NA packet are expounded on the Table 1.

*Table 1. The Structure of NA Packet*

| Packet's Structure | Type | Details |
|---|---|---|
| Ethernet | Source MAC | MAC of targeted host |
| | Destination MAC | MAC of sender host |
| IPv6 | Source IPv6 | IPv6 of targeted host |
| | Destination IPv6 | IPv6 of sender host |
| ICMPv6 | Destination ICMPv6 | ICMPv6 of sender host |
| | ICMPv6 type | 136 |
| Switch | DPID | DPID of switch that directly connected to sender host |
| | DPID's Port | Port of sender host at switch |

We conducted the experiment using Mininet [16][17], which simulated in tree topology. It consisted of 7 virtual switch using Open Virtual Switch (OvS) [18], 8 virtual hosts, and 1 controller using Ryu [19]. The experiment was emulated on a single Ubuntu 16.04 PC specifically consisted of Intel Core CPU i5 3210M Processor and 4 GB DDR3 1600 MHz SDRAM. The emulation's topology is described by Figure 3.

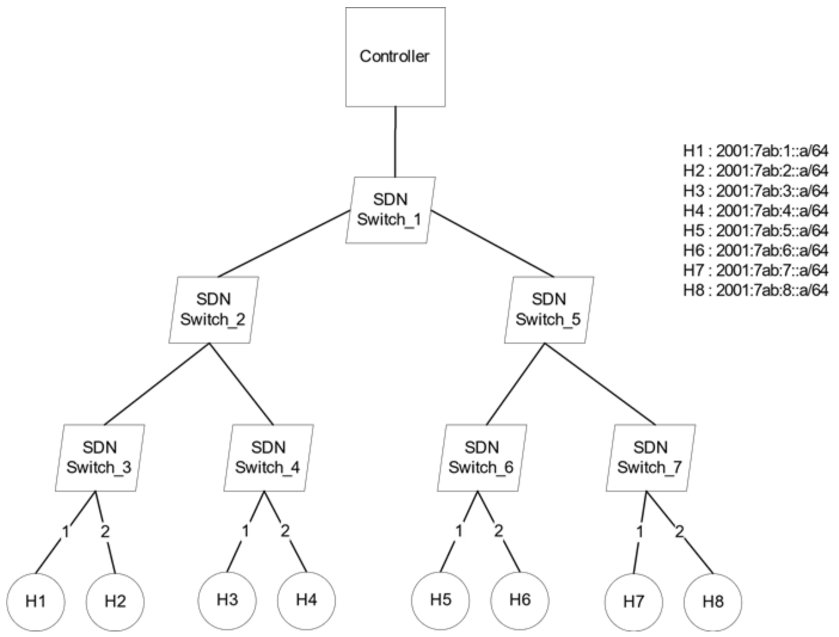


*Figure 3. The Simulation's Topology*

The research performed two distinct simulation scenarios for comprehending the impact of the reactive application. The randomly generated NS consisted of both random IPv6 and MAC address were sent from H1 to H7 on the first scenario using Scapy [20]. The sending rate varied from 500, 1000, and 2000 NS packets/second sent by using Tcpreplay [21]. Time difference from NA and the corresponding NS was analyzed specifically for justifying the capability of reactive application compared to the traditional switch. The packet loss ratio was also being investigated for comprehending the possible source of network congestion. The second scenario involved H1 and H7 as well. However, in term of mimicking the real network environment, H1 performed UDP transmission to H7 using Iperf [22]. Then H1 transmitted randomly generated NS packet to H7. During the specified event, the authors also analyzed the same variable as the first scenario.

## 3. Results and Discussion

As far as the experiment's results is concerned, this section expounds all of the details which have been performed during the simulation. It could be analyzed if the application successfully reducing the NS's response time which indicated the controller responding the request directly proven by the Figure 4. The Figure 4 illustrated the NS response time that extracted from the traditional switch and the reactive application for handling NDP process. As could be seen, the traditional switch method produced slower response time at approximately 7 ms than the controller-based proxy resided at 2 ms. This could be happened because the reactive method shortened the NS networking journey. The controller directly replied the NS packet by using OFPT_PACKET_OUT message contained the NA packet which extracted based on the MAC to IP table stored in the controlled based application on the NDP process. In converse, the traditional switch application did not give the controller a direct access for storing the NS and NA information. Therefore, the controller forwarded the NS packet through the network using multicast method. Subsequently, the targeted host responded by sending the NA packet then the response time was calculated.

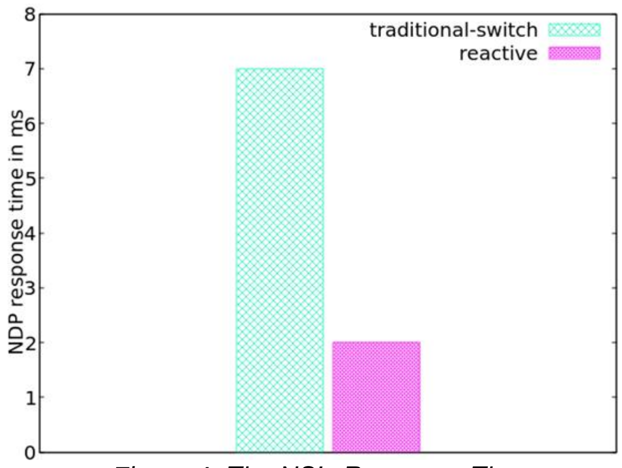
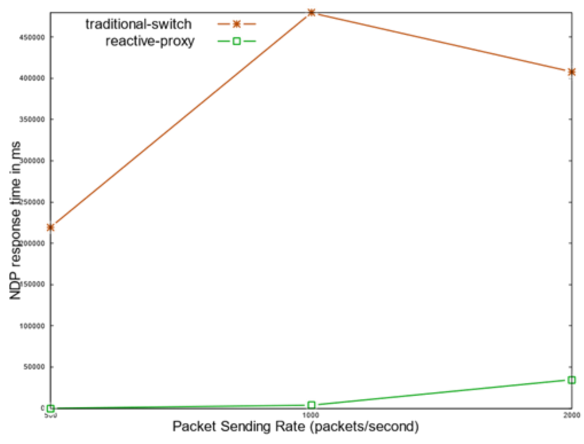


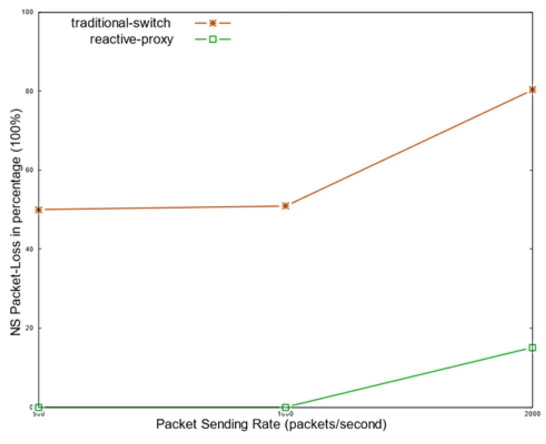
*Figure 4. The NS's Response Time*

A significant result was deduced from the first scenario illustrated in Figure 5. The response time calculated during 1000 NS packet/second in traditional switch, was pointed approximately more than 455000 ms or 7.5 minutes which indicated that the controller was overwhelmed. The anomaly was shown during 2000 NS packet/second. The response time variable was shortly faster than the previous sending rate because of the impact of packet loss event. Since only 20% of the NS packet originated from the sender host was successfully responded by the targeted host, therefore the average response time was accumulated without the remaining 80%. In term of the percentage of the packet loss in traditional switch, the result showed, along with the growth of the NS packet sending rate, the percentage's value also incrementing in the same manner. This could be happened because the link between controller and the SDN-switch was congested by the NS originated from the sender host and the NA packet from the destined host.

Conversely, after implementing the controlled based proxy NDP application, the response was dramatically reduced proven by the variable's value resided below 40000 ms because the controller directly replied the incoming NS request. The similar result also described in packet loss percentage. The packet loss value significantly decreased almost 50% than the normal results produced by the traditional switch application which indicated that the controller was not being congested by the NA packet sent by the targeted host.

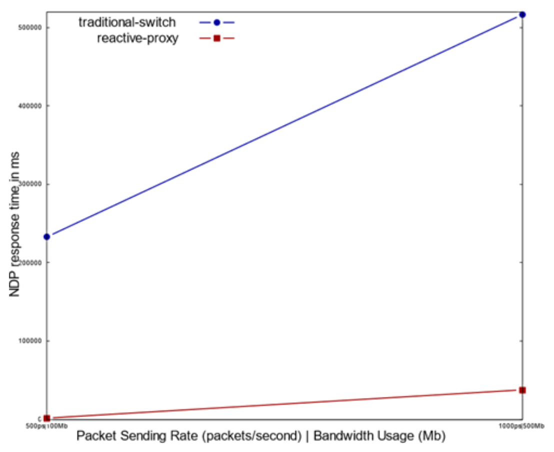
Similarly, during the second scenario depicted in Figure 6, the response time and the packet loss variable showed a dramatical rise in traditional switch application. However, the results were slightly higher than the first scenario since the link between the SDN-switch and either H1 or H7 was congested by the normal UDP transmission that had a bandwidth ranged in either 500 and 1000 Mb/s. In term of the controller's CPU usage in both scenarios depicted in Figure 7, the traditional switch had slightly more value than the controller-based application. CPU usage resulted from both scenario was resided below 50% since Ryu did not implement modularity [15].

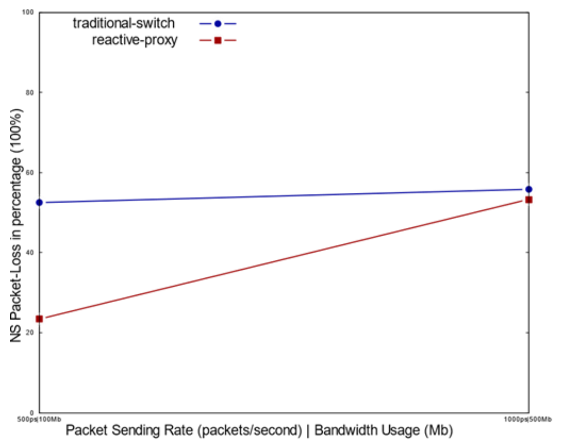




*a) NS Response Time in ms* *b) NS Packet-Loss Percentage*

*Figure 5. The NS's Response Time and Packet Loss Percentage During the First Scenario*





*a) NS Response Time in ms* *b) NS Packet-Loss Percentage*

*Figure 6. The NS's Response Time and Packet Loss Percentage During the Second Scenario*

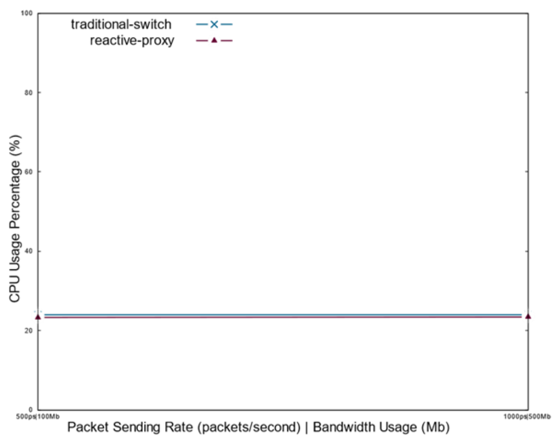


*Figure 7. The Controller's CPU Usage During the Second Scenario*

## 4. Conclusion

Based on the results and discussion, it could be concluded that the controller-based proxy NDP application successfully reduced the NS response time by shortening NS networking journey proven by its value which pointed below 50000 ms or 0.8 minutes in average. However, this trend may not behave in the same manner if the NS request is exponentially increase within a second interval possibly bring the controller into unpredictable state. This circumstance could be occurred since the controller was exhausted for processing all of the incoming NS packet due to the centralized logic control in SDN. To address the specified problem, an extensive method called proactive approach should be included for resolving the controller's overhead problem in the future research and development of current method. Therefore, the controller's load can be offloaded to the forwarding device.

## 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 1.3.0),"* O. N. Foundation, 2014.
[3] R. Hinden and S. Deering, *"Internet Protocol Version 6 (IPv6) Addressing Architecture,"* RFC Editor, 2003.
[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), Pp. 24-27, 2015.
[5] 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), Pp. 230-234, 2013.
[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), Pp. 404-410, 2017.
[7] 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), Pp. 301-306, 2015.
[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), Pp. 63-67, 2016.
[9] 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), Pp. 1-6, 2016.
[10] H. A. Naqvi, S. N. Hertiana, and R. M. Negara, *"Enabling Multipath Routing for Unicast Traffic in Ethernet Network,"* in 2015 3rd International Conference on Information and Communication Technology (ICoICT), Pp. 245-250, 2015.
[11] C. Kim, M. Caesar, and J. Rexford, *"Floodless in Seattle: a Scalable Ethernet Architecture for Large Enterprises,"* SIGCOMM Computer Communication Review, Vol. 38, No. 4, Pp. 3-14, 2008.
[12] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, *"Scalability of ONOS Reactive Forwarding Applications in ISP Networks,"* Computer Communications, Vol. 102, No. C, Pp. 130-138, 2017.
[13] K. Kataoka, N. Agarwal, and A. V. Kamath, *"Scaling A Broadcast Domain of Ethernet: Extensible Transparent Filter Using SDN,"* in 2014 23rd International Conference on Computer Communication and Networks (ICCCN), Pp. 1-8, 2014.
[14] Y. Xu, X. Lu, and T. Zhang, *"A Novel Efficient SDN Based Broadcast Scheme."* 2015.
[15] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, *"SDN Controllers: A Comparative Study."* Pp. 1-6, 2016.
[16] R. L. S. d. Oliveira, C. M. Schweitzer, A. A. Shinoda, and P. Ligia Rodrigues, *"Using Mininet for Emulation and Prototyping Software-Defined Networks,"* in 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Pp. 1-6, 2014.
[17] Mininet. (Online). Available: Available: http://mininet.org/
[18] O. vSwitch. (Online). Available: http://openvswitch.org/
[19] RYU. (Online). Available: https://osrg.github.io/ryu/
[20] Scapy. (Online). Available: http://www.secdev.org/projects/scapy/
[21] Tcpreplay. (Online). Available: http://tcpreplay.synfin.net/
[22] Iperf. (Online). Available: https://iperf.fr/