Reducing Packet-In Messages in OpenFlow Networks

Chit Su Khin†, **Aye Thandar Kyaw**, **Myo Myint Maw**, and **May Zin Oo**, Non-members

ABSTRACT

In OpenFlow networks, several packet-in traffic messages are sent to the controller to request routes from any switch on the network. Executing these packet-in messages and replying to control messages may interfere with the controller performance. Therefore, a mechanism for lightening the load on the controller by reducing packet-in traffic between the controller and the switch is proposed in this research. The proposed system operates by responding to the two flow entries for each request and removing packet-in messages from some unused multicast traffic. The proposed system can thus not only avoid a third packet-in message but also some multicast packet-in traffic, thereby reducing the load and traffic in the network as well as packet loss. According to the evaluation results, the proposed system can improve network performance by significantly reducing packet-in overhead.

Keywords: OpenFlow, Packet-In Message, Multicast, Traffic Load, Software Defined Networking

1. INTRODUCTION

Software defined networking (SDN) provides better performance for the centralized management of network traffic compared to the conventional network model. SDN consists of a control layer and data plane layer for centralized management. The control layer is composed of the centralized network controller which manages all network behaviors, especially involving traffic. The data plane is composed of forwarding infrastructure devices such as switches and routers. The control plane directly manages the forwarding plane using the OpenFlow protocol [1, 2].

Every switch in the network sends the packet-in to request the control messages from the controller for all network behaviors such as packet forwarding, link changes, and so on. A large number of packet-in traffic is received and handled by the OpenFlow controller to reply to these requests. Such packet-in messages are accumulated by the controller, and thus bottlenecks are encountered in the OpenFlow networks. Many researchers have tried to reduce controller overheads using various methods. Pranata *et al.* [3] proposed a framework for reducing traffic between the controller and switches in SDN. This framework reduces the number of packet-in and packet-out messages during rule installation in the flow tables of OpenFlow switches. The switch sends the first packet of a flow to forward path determination and chooses switches for rule installation using stacked multiprotocol label switching (MPLS). However, using multiple MPLS in each packet may cause significant transmission overhead in large networks.

Guo *et al.* [4] proposed JumpFlow to achieve a minimum flow table size and eliminate bandwidth waste in the MPLS-based forwarding approach on switches. JumpFlow selects any unused field in the packet header and inserts forwarding instructions into it to accomplish the mission. Simulating both unicast and multicast scenarios, it performs better than baseline schemes.

A mechanism for reducing the packet-in messages by instructing the two flow entries to respond to each request is presented in [5]. A method for removing flooding of packet-in messages is proposed in the paper originally presented at the ECTI-CON 2020 [5]. This research extends the previous idea and proposes a method for reducing some packet-in messages without affecting the network function and losing the replies to the control messages from the controller. The proposed system reduces some packet-in messages by inserting double-flow rules for each new packet and removing the packet-in messages of some unused multicast traffic. The Mininet emulator and Ryu OpenFlow controller are used for performance evaluation.

The remainder of this paper is organized as follows. Section 2 expresses the basic knowledge of OpenFlow, while Section 3 describes the proposed method in detail. Section 4 presents the simulation work, while Section 5 explains the formula used for counting packet-in messages. Section 6 expresses the results of the simulation, while the conclusion is drawn in Section 7.

2. OPENFLOW PROTOCOL

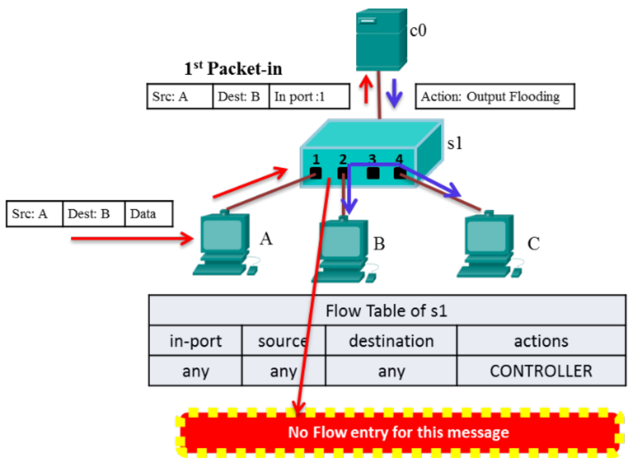
This section describes the theory of OpenFlow, separated into two subsections: packet forwarding and the challenges of OpenFlow networks.

Manuscript received on March 2, 2021; revised on July 8, 2021; accepted on August 19, 2021. This paper was recommended by Associate Editor Nattapong Kitsuwan.

The authors are with the Department of Computer Engineering and Information Technology, Mandalay Technological University, Mandalay, Myanmar.

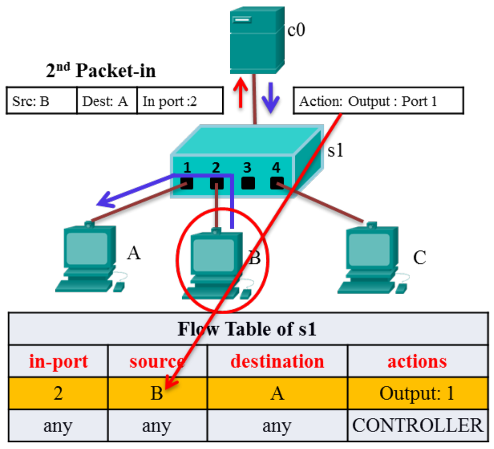
†Corresponding author: chitsukhin@mtu.edu.mm

Digital Object Identifier: 10.37936/ecti-eec.2022201.244944



Flow Table of s1			
in-port	source	destination	actions
any	any	any	CONTROLLER

Fig. 1: *The first packet-in message (host A to all nodes in the network).*



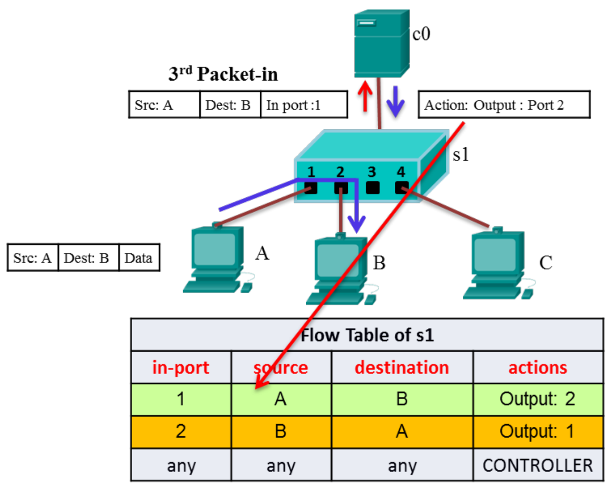
Flow Table of s1			
in-port	source	destination	actions
2	B	A	Output: 1
any	any	any	CONTROLLER

Fig. 2: *The second packet-in message (return host B to host A).*

2.1 Packet Forwarding in OpenFlow

In SDN, the controller centrally manages the data plane of the network via the southbound interfaces as OpenFlow protocol [6]. OpenFlow controls all forwarding processes programmatically and management of the network. When the data packet arrives at the OpenFlow switch for forwarding, the switch finds the flow entry that records this packet. The packet is then sent in accordance with action in the flow entry matching this packet. If the matched flow entry does not appear in the table, this flow is recognized as a table-miss entry, and the switch requests the flow rule from the controller by sending the packet-in messages and performs further actions in accordance with those of the flow entry [7].

Initially, the Ryu OpenFlow controller is enabled, and only one flow rule is added to the flow table of the switch. This flow entry is also known as a table-miss flow entry, and refers to “a packet from any source to any destination forwarded to the controller”. The switch must send at least three packet-in traffic messages to the controller to request the flow rule for a new packet flow [8]. The processing of the flow rule request which is



Flow Table of s1			
in-port	source	destination	actions
1	A	B	Output: 2
2	B	A	Output: 1
any	any	any	CONTROLLER

Fig. 3: *The third packet-in message (sending data host A to host B).*

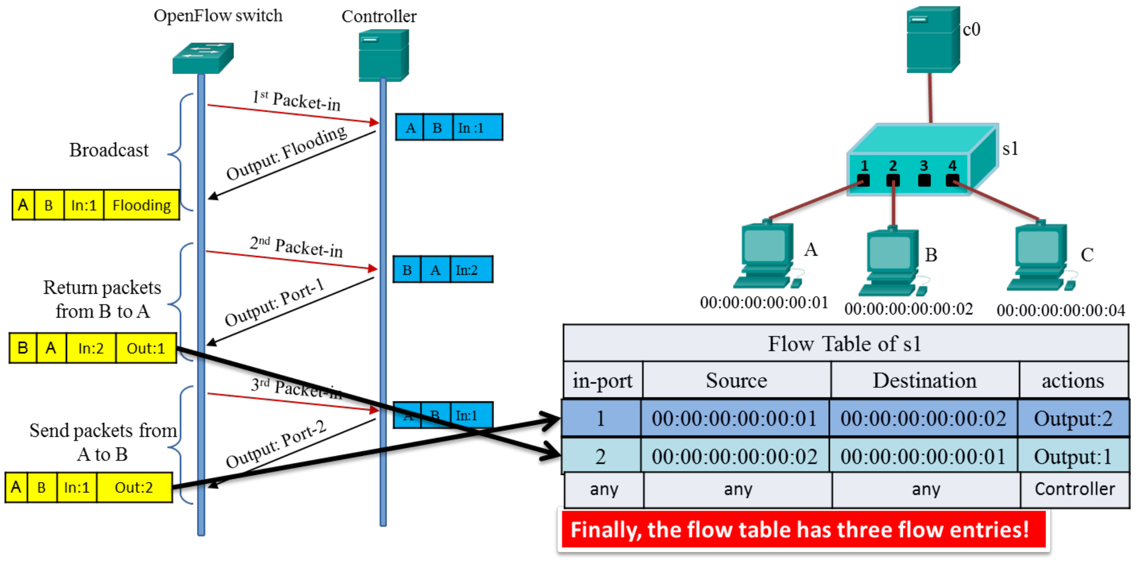
required to send three packet-in messages is illustrated in the following figures. Fig. 1 explains how to process the first packet-in, Fig. 2 second packet-in, and Fig. 3 the third packet-in. The process of sending the first packet-in by the switch and how to handle this packet-in by the controller is illustrated in Fig. 1. It is assumed that the three hosts A, B, and C are connected to ports 1, 2, and 4, respectively.

Assuming a scenario in which host A wants to send a packet to host B. On the first flow occasion, no flow rule exists in the flow table of the switch. To request such a flow rule, the switch sends the packet-in message to the controller. This time, the MAC address and number of post hosts A are maintained. The controller responds to the action by flooding this packet to other hosts in the network.

The second packet-in message is the returned message from host B to host A. Since the switch does not have the flow entry for host B to host A, it needs to send the next packet-in message for that flow. The controller responds to the flow rule with the action “forward to port 1” by forwarding the packet from host B to A. The switch adds that flow entry to the flow table and the packets from host B are then forwarded to port 1 by unicast, therefore, any packets will arrive at host C. This process is demonstrated in Fig. 2. Finally, the data packets from host A are sent to host B; the third packet-in message is also sent because the matched flow rule for that packet from host A to host B is not present in the flow table of the switch. The controller responds to flow rule with the action “forward to port 2”. That flow entry is recorded in the switch’s flow table and finally data packets are forwarded to port 2 which is connected to host B. Fig. 3 illustrates the process of sending the third packet-in message and receiving the data.

2.2 Challenges to OpenFlow

Central management is one of the strengths of OpenFlow. As the network grows, the controller handles more



Flow Table of s1

in-port	Source	Destination	actions
1	00:00:00:00:00:01	00:00:00:00:00:02	Output:2
2	00:00:00:00:00:02	00:00:00:00:00:01	Output:1
any	any	any	Controller

Fig. 4: *Event handling in OpenFlow.*

packet-in traffic. Many forms of packet-in messages are sent from the switch to the controller for various reasons such as requesting a flow rule for a table-miss entry, informing about changes in the network, sending multicast traffic, and so on.

At least three packet-in messages are sent merely for the purposes of requesting a flow rule. The first instance of packet-in flooding needs to be sent in accordance with the nodes connected to the network. The third type of packet-in messages can be sent as long as the bandwidth range is available. This is because the switch is sending the third packet-in messages in accordance with the data packets in the flow [8].

In this mechanism, the first type of packet-in flooding messages and the second type of flow rule requests must be transmitted from destination to source. The third type of flow rule request for sending packet-in message data from source to destination is performed on one occasion only for the first packet. Although this mechanism can reduce a substantial amount of traffic between the controller and switch, multicast packet-in messages tend to increase overheads.

When selecting which packet-in messages to omit, the first message cannot be ignored because the port number and MAC address of the destination are learned by the very first packet-in. The second and third packet-in messages can be merged because they are sent to request the route back and forth in a single path. Fig. 4 demonstrates the procedure for handling the packet-in messages for a new packet flow in the OpenFlow protocol.

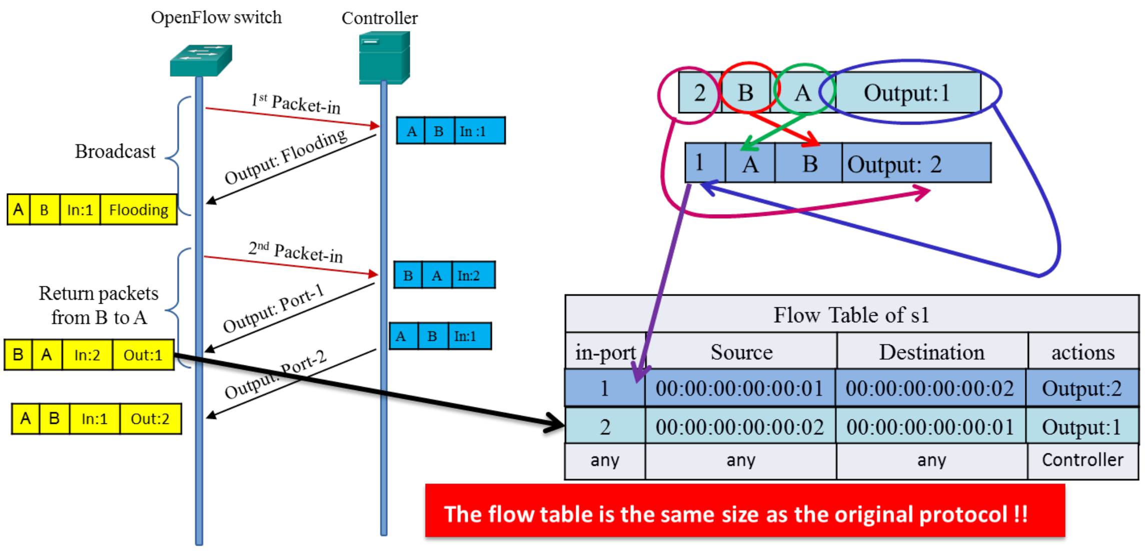
Another type of packet-in message that is not useful in OpenFlow is multicast flooding, especially the multicast dynamic name system (mDNS) address of 33:33:00:00:00:fb. The mDNS multicast traffic address is the most commonly used from the switch to the controller due to the RFC standard for a traditional network. Although it is not used for SDN/OpenFlow, many multicast packet-in traffic messages are being transmitted to the controller and the OpenFlow controller must respond by flooding. Since the controller needs to process these accumulating packet-in messages, it is overloaded. Kotani and Okabe [8] classified packet-in messages as important or unimportant, with unimportant packets filtered out to decrease CPU usage and control messages in the network. They classified important and unimportant loads based on heavy or light, with more packets being dropped in the case of heavy flow. Moreover, they used an extra flow table to record pending loads in the switches' memory, thereby potentially expanding resources and processing time.

3. THE PROPOSED SYSTEM

The original OpenFlow protocol used to send packet-in messages three times for a new data flow tends to increase the load on the controller. Processing the packet-in messages for multicast traffic also increases the load on the controller. Therefore, this research proposes reducing the controller overload by eliminating some of the ineffective packet-in messages.

The proposed system eliminates ineffective packet-in messages in two ways. Firstly, by eliminating the third packet-in message for forwarding a new flow of data by merging the second and third packet-in messages. The second method involves eliminating some multicast/broadcast packet-in messages by pre-installing the flow rule together with a table-miss flow entry for this multicast/broadcast flow of data as soon as the controller starts.

Eliminating the third packet-in message in the new flow of data is performed through the following steps. This procedure is illustrated in Fig. 5.



Flow Table of s1			
in-port	Source	Destination	actions
1	00:00:00:00:00:01	00:00:00:00:00:02	Output:2
2	00:00:00:00:00:02	00:00:00:00:00:01	Output:1
any	any	any	Controller

Fig. 5: Event handling of the proposed system.

3.1 Eliminating the Third Packet-in Message

The proposed system eliminates the third packet-in message using the following procedure.

- Request and respond to the first types of packet-in flooding as in the original OpenFlow.
- Merge the second and third packet-in traffic messages.
  - The first flow entry is recorded as the response of the second packet-in message from the controller.
  - The next flow entry is evaluated from the first flow entry by interchanging the addresses and port numbers of the source and destination.
- Record this flow entry as the next by reversing the first route.

The proposed system removes only the third packet-in traffic by combining it with the second. The second and third packet-in messages are used to request the route from destination to source and from source to destination, respectively. In the proposed system, the route from source to destination and the subsequent route from destination to source are received by using only the second packet-in message. Although the proposed system does not need to send the third packet-in message, the network behavior does not change. The loading and traffic of the third packet-in messages can be removed from the controller using the proposed method.

3.2 Eliminating Unused Multicast Packet-in Messages

Some multicast messages are transmitted by the networking devices according to the RFC standard. Since they are used in SDN architecture, they can cause controller overload. The proposed system eliminates some unused packet-in messages for multicast traffic and reducing the load on the controller by inserting a flow entry for this type of message.

In the original OpenFlow protocol, a flow entry only exists in the flow table of the OpenFlow switch when the controller starts working. This is the table-miss entry forwarded to the controller for any flow. In the proposed system, the next flow entry for the multicast traffic is added as soon as the controller starts.

Table 1: *Example flow table for the proposed system.*

Flow Table			
In-port	Source	Destination	Actions
Any	Any	Any	Controller
Any	Any	33:33:00:00:00:fb	Output: Flooding

The next flow entry is the flooding action for the multicast packets (from any source to the destination address of 33:33:00:00:00:fb). An example flow table for the proposed system is presented in Table 1. The first flow entry is recorded as the original OpenFlow while the second is added by the proposed system.

In the proposed system, the two flow entries for table-miss and multicast are added as soon as the controller starts. In the original protocol, the controller responds to the flooding action through this multicast packet-in message. By inserting the flooding action as the flow entry for the packets from this address, the proposed system eliminates the related packet-in messages.

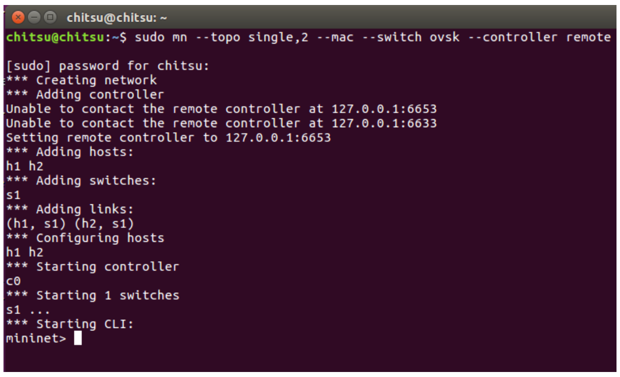
Even if it finds something useful, the network behavior does not change because the controller returns the same action. The proposed system can significantly reduce controller load without changing the network behavior because a large number of such packets are released when the controller starts.

4. SIMULATION SET UP

Simulation of the original OpenFlow and the proposed system are performed on a Mininet network emulator. The performance between the original protocol and proposed system is compared using the simulation

***Table 2:** Simulation network scenarios.*

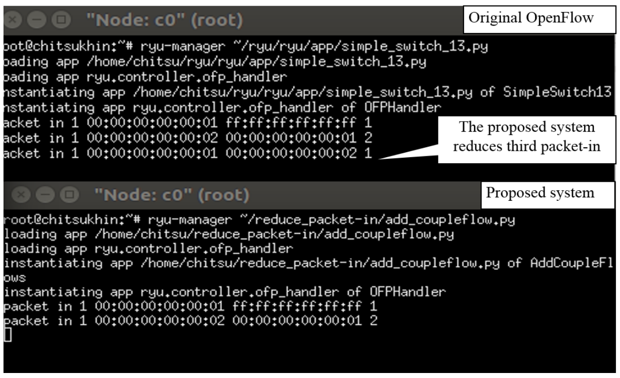
Network Parameters	Values
Number of hosts	5, 10, 15, 20
Number of switches	5, 10, 15, 20
Network topology	Single, linear
Bandwidth	1, 10, 20, 30, 40 Mbps
Transport protocol	UDP
OpenFlow protocol version	OpenFlow 1.3
Type of switch	Open vSwitch
Type of controller	Remote
OpenFlow controller	Ryu



```
chitsu@chitsu: ~
chitsu@chitsu:~$ sudo mn --topo single,2 --mac --switch ovsk --controller remote

[sudo] password for chitsu:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> 
```

***Fig. 6:** Creating a network on Mininet.*



```
"Node: c0" (root)
oot@chitsukhin:~# ryu-manager ~/ryu/ryu/app/simple_switch_13.py
oading app /home/chitsu/ryu/ryu/app/simple_switch_13.py
oading app ryu.controller.ofp_handler
nstantiating app /home/chitsu/ryu/ryu/app/simple_switch_13.py of SimpleSwitch13
nstantiating app ryu.controller.ofp_handler of OFPHandler
acket in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
acket in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
acket in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
```

```
"Node: c0" (root)
root@chitsukhin:~# ryu-manager ~/reduce_packet-in/add_coupleflow.py
loading app /home/chitsu/reduce_packet-in/add_coupleflow.py
loading app ryu.controller.ofp_handler
instantiating app /home/chitsu/reduce_packet-in/add_coupleflow.py of AddCoupleFl
ows
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
```

***Fig. 7:** Sending packet-in traffic for a new data flow.*

results. Performance is evaluated in the linear and single switch networks with different network parameters, as expressed in Table 2.

Mininet networks not only use real code, including applications of standard Unix/Linux network, but also the actual Linux kernel and network stack [9]. In the proposed system, the network infrastructure is implemented using the Mininet emulator, with Ryu employed to set the OpenFlow controller. The network topology for the Mininet emulator is created using the 'sudo mn' command. Other network scenarios can be constructed by typing the commands after "$ sudo mn".

Fig. 6 illustrates the creation of a network on Mininet. The example network is a single switch SDN with two hosts using a remote controller and the Open vSwitch. Fig. 7 compares the packet-in message handling process

***Table 3:** Counting the packet-in messages in the original OpenFlow.*

Number of Hosts	Traffic		Number of Packet-In Messages
2	$(1 \to 2)(2 \to 1)$	+ 1 flooding	2 + 1 = 3
3	$(1 \to 2)(2 \to 1)$ $(1 \to 3)(3 \to 1)$ $(2 \to 3)(3 \to 2)$	+ 3 flooding	6 + 3 = 9
4	$(1 \to 2)(2 \to 1)$ $(1 \to 3)(3 \to 1)$ $(1 \to 4)(4 \to 1)$ $(2 \to 3)(3 \to 2)$ $(2 \to 4)(4 \to 2)$ $(3 \to 4)(4 \to 3)$	+ 6 flooding	12 + 6 = 18

for a new data flow using the original OpenFlow protocol and the proposed system. The original OpenFlow uses three packet-in messages for a new data flow, while the proposed method requires only two.

5. COUNTING THE NUMBER OF PACKET-IN MESSAGES

In the original OpenFlow protocol, three packet-in messages are needed to obtain a new connection between two hosts. The first is a flooding packet-in message from the source to all hosts in the network, while the second is the response message from the destination host to the source host and the third message requests the route for sending data from the source host to the destination host.

The total number of packet-in messages used to communicate with all nodes in the entire network can be evaluated as shown in Table 3.

For a two-host network, the required packet-in messages to facilitate communication between hosts 1 and 2 are as follows:

- Host 1 sends the broadcast message (flooding) to facilitate communication from host 2.
- Host 2 responds the route to host 1, e.g., the first link $(2 \to 1)$.
- Host 1 requests the route to host 2, e.g., the second link $(1 \to 2)$.

Three packet-in messages must be used to request the route to the controller to build the connection between two hosts. To connect to all nodes in the entire network for a three-host network requires communication between hosts 1 and 2, hosts 2 and 3, and hosts 1 and 3, respectively. Nine packet-in messages are needed for a three-host network. The total packet-in messages required for a single switch network are as follows:

- For 2 hosts, $2 + 1 = 3$ packet-in messages.
- For 3 hosts, $6 + 3 = 9$ packet-in messages.
- For 4 hosts, $12 + 6 = 18$ packet-in messages.
- For 5 hosts, $20 + 10 = 30$ packet-in messages.
- For 6 hosts, $30 + 15 = 45$ packet-in messages.

The number of packet-in message pairs can be calculated using the following permutation

Table 4: *Counting the packet-in messages in the proposed system.*

Number of Hosts	Traffic	Number of Packet-In Messages
2	$(1 \to 2)$ + 1 flooding	1 + 1 = 2
3	$(1 \to 2)$ $(1 \to 3)$ + 3 flooding $(2 \to 3)$	3 + 3 = 6
4	$(1 \to 2)$ $(1 \to 3)$ $(1 \to 4)$ + 6 flooding $(2 \to 3)$ $(2 \to 4)$ $(3 \to 4)$	6 + 6 = 12

$$\text{no.packet\_in}_{\text{OpenFlow}} = 1.5 \cdot {}^{n}P_2 = 1.5 \cdot \frac{n!}{(n-2)!} \quad (1)$$

where n means the total number of hosts in the entire network. Since the linkage is between two hosts, the 2 number combination is used. The coefficient 1.5 means $1 \cdot {}^{n}P_2$ for a pair of messages, while $0.5 \cdot {}^{n}P_2$ refers to flooding traffic. As the number of switches increases, so does the number of packet-in messages. The number of packet-in messages will be the multiple number of switches because all packet-in messages are sent repeatedly for each switch

$$\text{no.packet\_in}_{\text{OpenFlow}} \times \text{no.switch} \quad (2)$$

In the proposed system, the third packet-in message is eliminated to communicate the new flow. The packet-in messages in the proposed system are shown in Table 4. As can be observed, the proposed system can reduce the packet-in messages by one-third compared to the original OpenFlow protocol. The number of packet-in messages in the proposed system remains at two-thirds of the original protocol.

The number of packet-in messages for the proposed system can be evaluated as in Eqs. (3) and (4):

$$\text{no.packet\_in}_{\text{Proposed\_sys}} = \frac{2}{3} \cdot \text{no.packet\_in}_{\text{OpenFlow}} \quad (3)$$

$$\text{no.packet\_in}_{\text{Proposed\_sys}} = \frac{2}{3} \cdot 1.5 \cdot {}^{n}P_2 = \frac{n!}{(n-2)!} \quad (4)$$

The above formula for the number of packet-in messages is evaluated to facilitate the communication of all nodes in the entire network without considering the broadcast/multicast packet-in traffic.

6. SIMULATION RESULTS

The purpose of this research is to reduce the controller overload caused by packet-in messages. Packet-in messages are crucial to the communication between the controller and forwarding plane in SDN. However, the switch sends packet-in messages for every operation in the network. These packet-in messages are usually overloaded at the controller. Therefore, this research proposes a method for reducing the flow of packet-in messages without affecting network behavior.

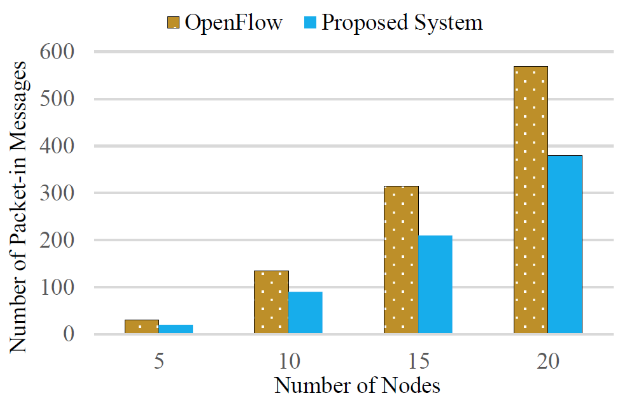


Fig. 8: *Number of packet-in messages in a single switch network.*

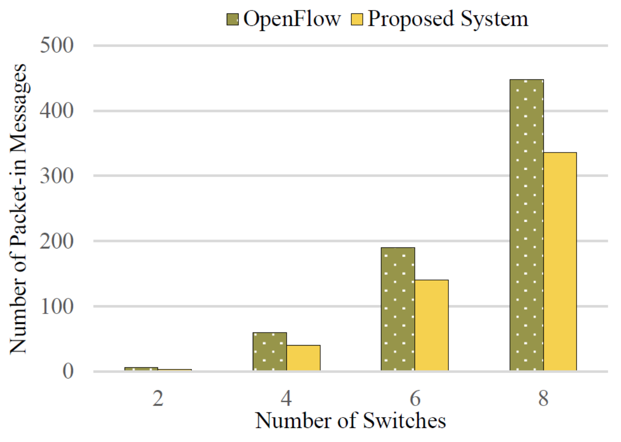


Fig. 9: *Number of packet-in messages in linear switch networks.*

The proposed method is implemented on a Mininet network simulator to compare the performance with that of the original OpenFlow. Firstly, the packet-in messages for accessing all nodes in the network are evaluated depending on the different number of nodes, switches, and network topologies. The packet losses and CPU usage are then measured to evaluate network performance.

Fig. 8 shows the number of packet-in messages in single switch networks. The packet-in messages are counted using the 'pingall' command in the network with different numbers of nodes. Fig. 9 shows the number of packet-in messages on linear networks with different numbers of switches.

According to the results in Figs. 8 and 9, the proposed system can reduce the packet-in messages by one-third without even counting the multicast and broadcast traffic. These messages are difficult to count because they are sent continuously. The proposed system can also reduce some of the multicast packet-in messages.

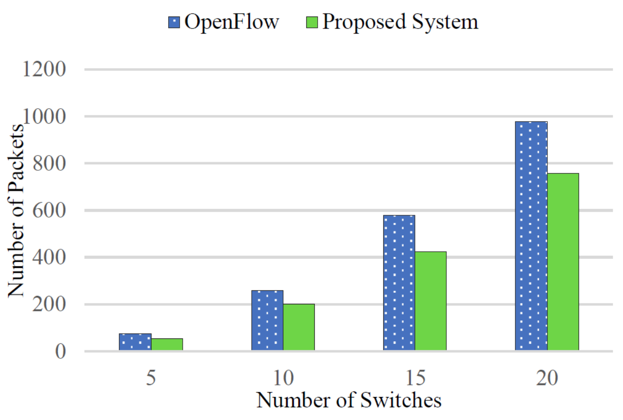


Fig. 10: *Number of packets crossing a switch.*

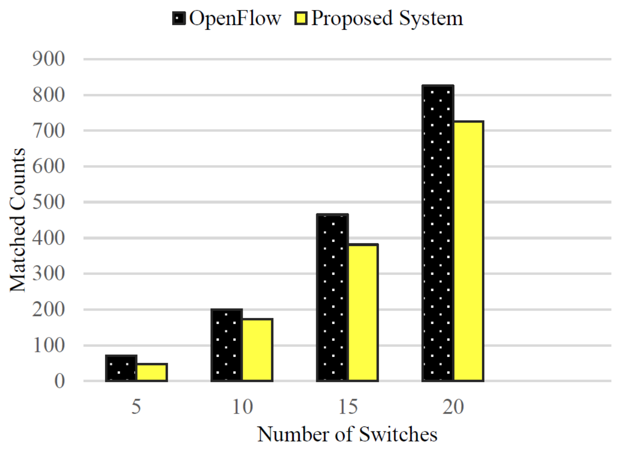


Fig. 11: *Number of matched counted on a switch.*

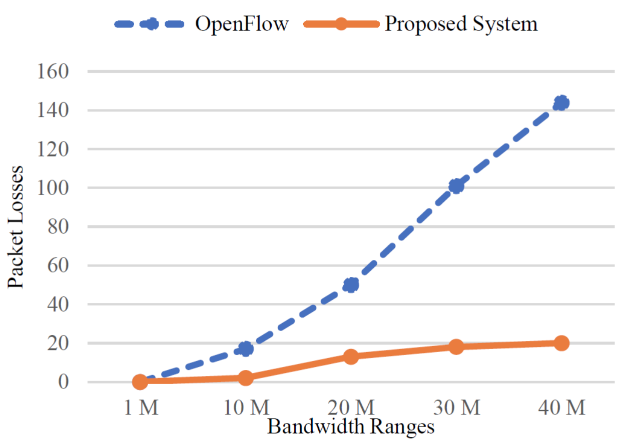


Fig. 12: *Number of packet losses in a single switch network.*

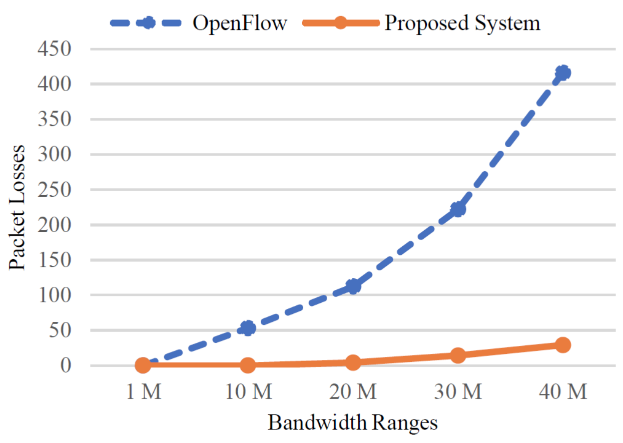


Fig. 13: *Packet loss in a linear switch network.*

Pranata *et al.* [3] were able to reduce about 30% of packet-in traffic by sending only data relating to the first packet-in message. The proposed system will have already received the flow rule from the first packet-in, so no further packet flow is required. The proposed system can eliminate secondary and subsequent packet-in messages as well as multicast packet-in traffic as flow_mod messages and packet-out messages. The proposed system can reduce the load caused by the first flow of packet-in messages on the controller by at least one-third as well as the load in the second and subsequent packet-in and multicast traffic.

Since the proposed system can reduce third packet-in messages for every new flow of packet, the number of packets crossing the network can also be reduced without affecting the network behavior. When pinging all nodes in the network, the number of packets crossing a switch are monitored. Fig. 10 shows the number of packets crossing a switch on five-switch linear network. According to the results of Fig. 10, the proposed system can reduce about 22% of packets across the switch.

Not only the load of the controller but also the load of the switch can be reduced by the proposed system. Every incoming packet that enters the OpenFlow switch is matched by the flow rules in the flow table. The proposed system reduces the number of packets, which tends to reduce the number of matched times slightly. Fig. 11 shows the match count on a switch to connect all the nodes in the network. To obtain this result, the matched count was tested by changing the number of different switches on the linear network.

To measure network performance, packet loss and CPU usage are evaluated by sending UDP traffic between two hosts. To detect packet loss, 10 MB of UDP data traffic is sent between two hosts with different bandwidth ranges in both single and linear switch networks. Fig. 12 expresses the number of packet losses in a single switch network, while Fig. 13 presents the figures for a linear switch network. According to the results, the proposed system can reduce packet loss by about 96% compared to the original OpenFlow. For all network types, packet loss can be significantly reduced by the proposed method. The original OpenFlow increases traffic congestion at the controller, so more packets may be lost.

The percentage of CPU usage in the 10-host linear switch network is presented in Fig. 14. The proposed system can reduce CPU usage by around 9% compared to the original protocol. Kotani and Okabe [8] classified data packets from a source host to the switch as important or unimportant, dropping the latter, which tends to reduce CPU load in both the SDN controller

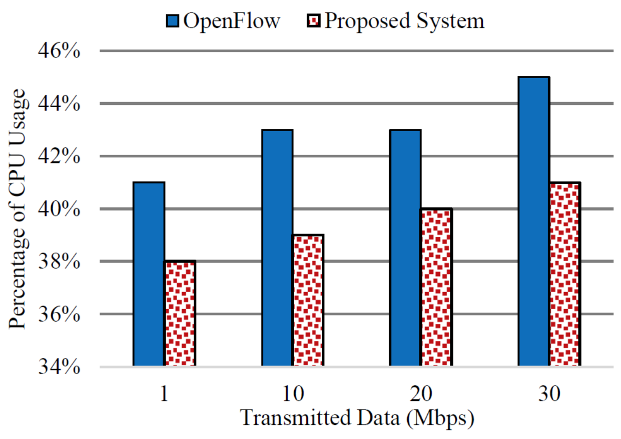


Fig. 14: *Percentage of CPU usage in a 10-host linear switch network.*

and switches. However, defining important loads as heavy and light loads as unimportant, may result in more packets being dropped in the event heavy flows occur inside the network. Another side effect may involve higher demand for processing resources and time spent in the OpenFlow switches, since a pending flow table is added to the inside. Since the proposed method can reduce packet-in traffic, the reduction in bottlenecks and packet loss is significant.

7. CONCLUSION

This research proposes a method for reducing the controller load caused by packet-in messages in OpenFlow networks. The proposed system reduces traffic by adding double-flow rules for each request and removing packet-in messages from some unused multicast traffic. The proposed system can reduce packet-in messages by 33% for each new data flow, as well as for some multicast traffic. The number of packets generated by the switch and the matching count in the switch can also be reduced by nearly 22% compared to the original OpenFlow protocol. Packet loss for the data traffic can be significantly reduced in both single switch and linear networks. The proposed system also reduces nearly 96% of packet loss compared to the original protocol. The downside of the proposed system is that one more flow entry would need to be retained for every switch in the network. However, there is no harm in having an extra flow entry. In conclusion, the proposed system can reduce network load and significantly improve performance.

REFERENCES

[1] H. A. Eissa, K. A. Bozed, and H. Younis, "Software Defined Networking," in *2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, Sousse, Tunisia, 2019, pp. 620–625.

[2] A. Meshinchi, "QoS-Aware and Status-Aware Adaptive Resource Allocation Framework in SDN-Based IoT Middleware," M.S. thesis, École Polytechnique, Montreal, Canada, May 2018.

[3] A. A. Pranata, T. S. Jun, and D. S. Kim, "Overhead reduction scheme for SDN-based data center networks," *Computer Standards & Interfaces*, vol. 63, pp. 1–15, Mar. 2019.

[4] Z. Guo, Y. Xu, M. Cello, J. Zhang, Z. Wang, M. Liu, and H. J. Chao, "JumpFlow: Reducing flow table usage in software-defined networks," *Computer Networks*, vol. 92, pp. 300–315, Dec. 2015.

[5] C. S. Khin, M. Zin Oo, and A. T. Kyaw, "Packet-in Messages Handling Scheme to Reduce Controller Bottlenecks in OpenFlow Networks," in *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, Phuket, Thailand, 2020, pp. 502–505.

[6] D. Kotani and Y. Okabe, "Packet-In Message Control for Reducing CPU Load and Control Traffic in OpenFlow Switches," in *2012 European Workshop on Software Defined Networking*, Darmstadt, Germany, 2012, pp. 42–47.

[7] "Mininet Overview." http://mininet.org/overview/

[8] D. Kotani and Y. Okabe, "A packet-in message filtering mechanism for protection of control plane in openflow networks," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Marina del Rey, CA, USA, 2014, pp. 29–40.

[9] D. Gao, Z. Liu, Y. Liu, C. H. Foh, T. Zhi, and H.-C. Chao, "Defending against Packet-In messages flooding attack under SDN context," *Soft Computing*, vol. 22, pp. 6797–6809, 2018.

Chit Su Khin received her B.E. (IT) degree from Technological University (Kyaukse), Myanmar. Afterward she received a M.E. (IT) degree from Mandalay Technological University (MTU), Myanmar. She is currently doing her Ph.D under the Department of Computer Engineering and Information Technology, Mandalay Technological University (MTU), Myanmar. Her interesting area is protocols in software defined network.

Aye Thandar Kyaw received her Bachelor and Master Degrees in Information Technology at Technological University (Kyaukse), Myanmar. She is currently doing her Ph.D under the Department of Computer Engineering and Information Technology, Mandalay Technological University (MTU), Myanmar. Her interesting area is Security in Software Defined Network and Wireless Network.

Myo Myint Maw received the B.E and M.E degrees in Information Technology from Department of Information Technology, Mandalay Technological University (MTU), Myanmar, in 2005 and 2007, respectively, and the D.Eng. degree in Electrical Engineering from the King Mongkut's Institute of Technology Ladkrabang (KMITL), Thailand, in 2015. Since 2007, she has worked with the Department of Computer Engineering, and Information Technology, where she is currently a professor in Department of Computer Engineering, and Information Technology, MTU, Mandalay, Myanmar. Her research interests include ICT, IoT, UAV, channel modeling and propagations, machine learning algorithms and wireless communications.

May Zin Oo was a former Hubert H. Humphrey fellow at Pennsylvania State University, USA. She was a professor of Computer Engineering and Information Technology at Mandalay Technological University, Myanmar. She received her Bachelor and Master Degrees in Information Technology at Mandalay and Yangon Technological Universities. Later she received her Ph.D. in wireless network at the University of Malaya, Malaysia. Her research interest includes mobile ad hoc and sensor network and software defined network.