International Journal of Future Generation Communication and Networking
Vol. 9, No. 8 (2016), pp. 9-22
http://dx.doi.org/10.14257/ijfgcn.2016.9.8.02

Proactive-LRU: An Efficient Flow Table Management Scheme in SDNs

Dongryeol Kim and Byoung-Dai Lee*

Kyonggi University, Department of Computer Science, Suwon, Republic of Korea
{kdr, blee}@kyonggi.ac.kr

Abstract

A Software Defined Network (SDN) separates an existing network structure into a control plane and a data plane and centrally manages the network at the control plane. SDN is able to control the network in detail and operate efficiently by managing the traffic by flow unit. However, if the performance of the switches comprising the SDN varies greatly from one to another, it has a problem of delayed handling of traffic due to the bottleneck caused by an overhead concentration at the controller when the message exchanges between the low performance switch and the controller for flow table replacement increase in number. This research paper suggests a flow table management scheme, called Proactive-LRU, to alleviate this problem and compares its performance to that of the existing replacement algorithms such as LRU, LFU, and FIFO. Proactive-LRU scheme is a flow table management scheme that deletes the flow from all switches available within the forwarding path when a flow is deleted and reduces the number of message exchanges and the controller overhead. In order to compare the performance of the suggested scheme against that of the existing algorithms (FIFO, LRU, LFU), video traffic and web browsing traffic have been mixed in certain ratios in an SDN environment for the experiment. The analysis of the results showed that the number of message exchanges as well as the overhead and table overflow of switches in the suggested scheme were significantly reduced.

***Keywords**: SDN, Table Overflow, Controller Overhead, Flow table Replacement*

1. Introduction

Software defined networking is a technology that provides flexibility to the network by separating the existing network structure into a control plane handling the network intelligence and a data plane handling data transfer, and by centralizing the control of the overall network in the control plane. [1][2]. Packets are managed by flow unit in an SDN and information of the flow is saved and managed in the flow table at the switch level. When a packet arrives at a switch, the switch searches the flow table, finds a matching flow for the packet and takes a relevant action (transfer, modify, terminate) for that flow or creates a new flow for the packet through communication with the controller. The flow table is created and processed within the limited memory capacity of the switch and this capacity is different from one switch vendor to another. If the scale of a network becomes large, the switches used can be from various vendors and this may create a problem of packet transmission delay and increase of the controller overhead because of the switches with relatively poor performance used in the packet transfer process. This can result in a performance drop in the overall network. For example, flow table replacements occur frequently in a network consisting of switches with different performances, when the number of flows is greater than the flow table size of the switch. The time required for message exchanges and for the controller to determine which flow to delete when

* Corresponding Author

changing a flow table can increase the network's overhead and hence result in a performance drop of the entire network [3]. Therefore, it is very important to reduce the number of flow replacements and the controller's overhead.

Hence, a flow table management scheme to reduce the overhead of the controller as well as the number of flow replacements is suggested in this paper. This paper is structured as follows: Section 2 introduces the flow management scheme and a number of related studies in reducing the overhead of the controller. Section 3 explains the problems occurring from the performance differences between the switches that comprise the network environment with SDN technology applied. Section 4 suggests the flow table management scheme that can minimize the packet transfer delay caused by the problems mentioned in Section 3. Section 5 compares the performance of the suggested flow table management scheme to that of a number of representative replacement algorithms like FIFO, LRU and LFU and conducts a performance review. Finally, Section 6 describes the conclusions and future research directions.

2. Related Work

The network where SDN technology is applied may have large overheads within many processes such as packet handling or switch management of the controller. [4] describes a framework utilizing a flow-based monitoring scheme and it uses a wild card scheme to process newly arrived packets without *PacketIn* messages to the controller. When there exists a flow matching the packet arrived at the switch, the flow counter increases in number and when this count reaches the pre-set threshold, it is categorized as an elephant flow and is sent to the controller for flow processing. This process has reduced the number of *PacketIn* message transfers from the switch to the controller as well as the overhead. However, using a wild card has the disadvantage that the limited ternary content-addressable memory (TCAM) of the switch can be used up thoughtlessly. [5] shows how the switches managed by a controller with large overhead are moved to other controllers to reduce the overhead and balance out. [6] suggests a Kandoo framework. This is a method to reduce the overhead of the controller caused by frequent network events. In this method, the control plane is separated into a root controller and multiple sub-controllers, where any request regarding the overall network condition is handled by the root controller whereas the other events are handled by the sub-controllers, thereby distributing the overhead of the controller. [7] suggests a NOX Controller application that aims at a distributed platform by distributing the controllers and letting each controller manage the network locally. This method has achieved an overhead distribution by spreading the controllers over the network, but is not practical due to handover problems between multiple controllers and an absence of a standard east-west bound API that is the interface between the controllers. [8] suggests a tag-in-tag scheme. This is a method to reduce the flow table size by routing multiple flows via specific paths. [9] describes a similar method where a busy core network is loosened up through label switching and the number of flow entries is decreased at the TCAM.

3. Problem Description

SDN can allow users to operate a network as desired by keeping the network management at the controller to centrally manage various switches and virtual switches. However, if there are large differences in performance between the switches that constitute the network, a problem like the following may occur.

3.1. Table Overflow

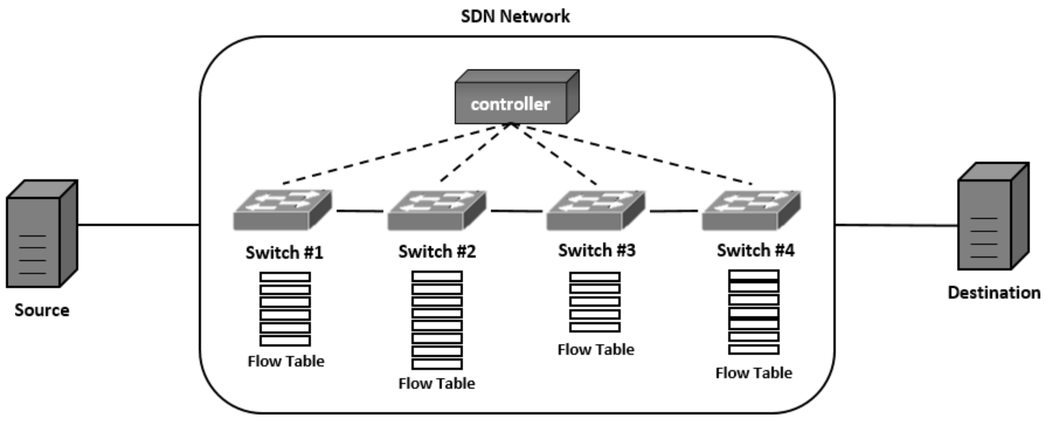

Figure 1. An SDN Network with Heterogeneous Switches

Figure 1 shows a linear topology SDN network consisting of the switches with different flow table sizes. When a packet arrives at a switch and there is no flow matching the arrived packet in the flow table, a communication with the controller is made and a new flow is created and added to the table. When many packets arrive and create flows, the table becomes full. At this stage, any additional flow created by newly arriving packets cause table overflow. When a packet arrives at the switch and there is no matching flow for that packet, the switch and the controller exchange messages and create a new flow. If the table is already full at that moment, it causes table overflow. In order to create space within the table, the switch and the controller exchange messages further to determine which flow to delete, delete that flow from the table and add a new flow. For example, when a network is configured as in Figure 1 and 100 packets matching all different flows are transmitted from the source, 100 flows are created and any switch with table size less than 100 will see an occurrence of table overflow.

3.2. Controller Overhead

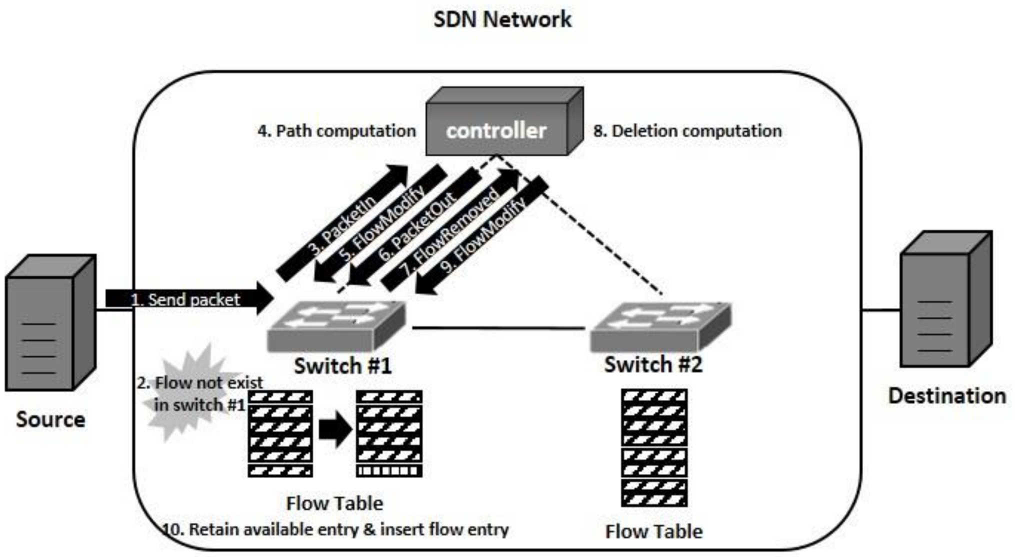

Figure 2. Flow Table Update Process

Figure 2 shows a flow table update process to explain how overhead occurs at the controller.

① When the table is full and a packet arrives, the switch looks for a relevant flow matching that packet in the table. If there is no such flow, it sends a *PacketIn* message to the controller to query flow entry generation.

② The controller then computes the path and sends out a *FlowModify* message to the switch to order a new flow entry to be created.

③ The controller also sets the destination of the arrived packet, adds it to a *PacketOut* message and sends it out to the switch.

④ However, since the table is already full, the switch sends out a *FlowRemoved* message to the controller to determine which flow to delete.

⑤ The controller decides which flow to delete according to the flow table replacement algorithm and sends out a *FlowModify* message to the switch again.

In this process, the controller updates the flow table by exchanging messages 5 times with one switch and runs the flow table replacement algorithm to determine which flow to delete. Thus, five message exchanges occur between a switch and the controller and a process to determine which flow to delete from the flow table is executed in every switch having a smaller flow table than the number of packets transferred. In this process, there is a lot of overhead. In Figure 2, if more packets than the flow table sizes of switches 1 and 2 are transferred, message exchanges and controller computation processes take place at both switches. In a large scale network, such processes will occur more frequently resulting in larger overhead at the controller and the performance of the overall network drops due to the impact on packet transmission speed.

4. Proactive-LRU Scheme

In the Proactive-LRU (P-LRU) flow table management scheme suggested in this paper, when a flow due to overflow at the flow table of a switch is deleted, the same flow as the deleted one is also deleted at any switch with a full flow table. With this scheme, it is possible to secure sufficient flow table space in advance without message exchanges at all switches other than the one with table overflow and this can also reduce the overhead like packet transmission delays.

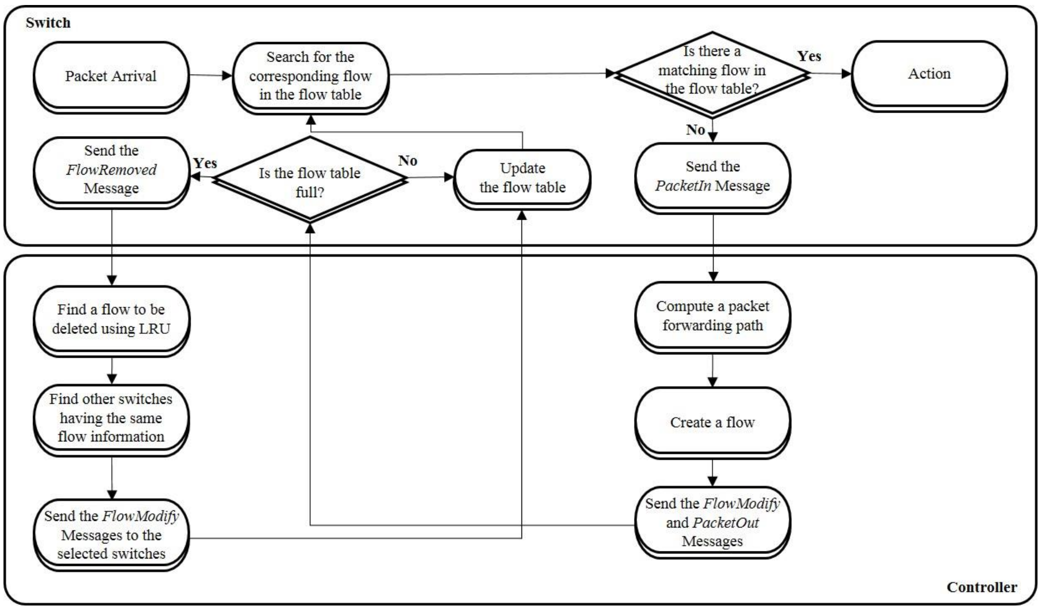
Switch
Packet Arrival
Search for the corresponding flow in the flow table
Is there a matching flow in the flow table?
Yes
Action
No
Send the *FlowRemoved* Message
Yes
Is the flow table full?
No
Update the flow table
Send the *PacketIn* Message
Find a flow to be deleted using LRU
Find other switches having the same flow information
Send the *FlowModify* Messages to the selected switches
Compute a packet forwarding path
Create a flow
Send the *FlowModify* and *PacketOut* Messages
Controller

Figure 3. Flow Chart for P-LRU Scheme

Figure 3 is a flowchart displaying the actions of switches and the P-LRU scheme. The overall flow is as follows: When a packet arrives at a switch, the switch searches for a flow matching the arrived packet. If there is a matching flow, it runs an action relevant to the flow. Otherwise, a *PacketIn* message is sent out to the controller in order to create a flow. The controller creates a flow matching the packet and sends out a *FlowModify* message to the switch and designates the destination of the packet with a *PacketOut* message. Since overflow occurs when the flow table of the switch is full, the switch sends out a *FlowRemoved* message to the controller for flow table replacement. Upon receiving the message, the controller determines which flow to delete by using the LRU algorithm. After that, a *FlowModify* message is sent out to the switch with overflow to delete the existing flow and add a new flow. When the flow tables of other switches having the same flow as the one to be deleted are full, the same *FlowModify* message is also sent out to those switches to delete the flow and secure space. Flow tables are updated during this process.

In order to review the actions of P-LRU in a real network, Figure 4 assumes that there is a network composed of a number of switches with different performances and the moving path of a packet is set as (1). The size of each flow table supported by these switches varies from one table to another.

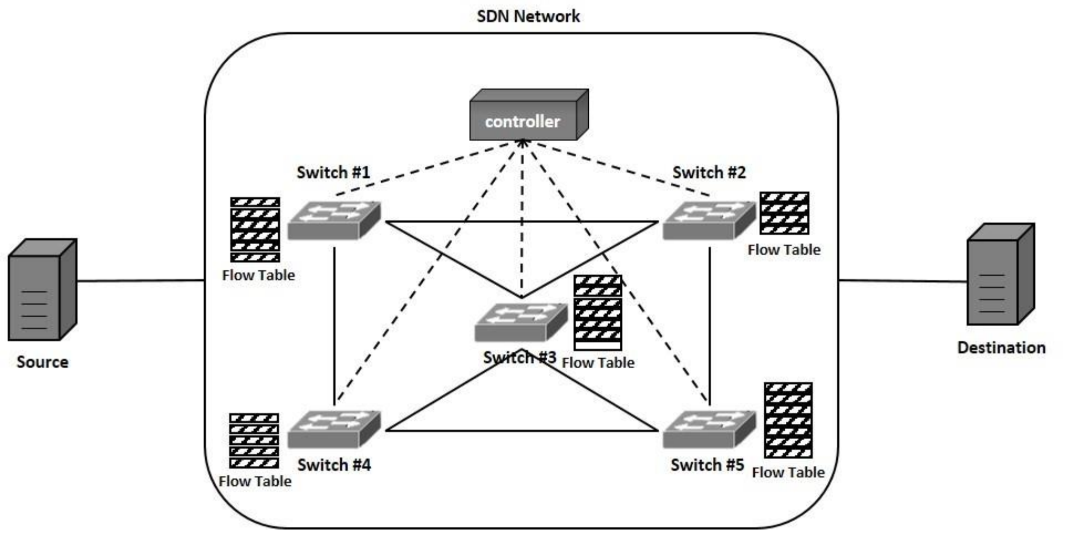

Figure 4. An SDN Network Composed of Switches with Different Performances

Source → Switch #1 → Switch #4 → Switch #3 → Switch #2 → Destination (1)

Among the switches in the path (1), all flow tables except that of switch 3 are full. Since switch 1 is already full, flow generation and flow table modification are needed in order to transmit the packet. Therefore, there are 5 message exchanges between the switch and the controller in order to create and insert a flow as well as to update the flow table, as mentioned in Section 3-2. Such message exchange processes are also run at switch 4 and switch 2 and hence add up to 15 message exchanges in total. There is some spare space in flow table of switch 3 so it will only exchange messages for flow generation and insertion 3 times. Therefore, if a network's configuration is the same as in Figure 4 and the moving path of packets are the same as (1), there will be a total of 18 message exchanges.

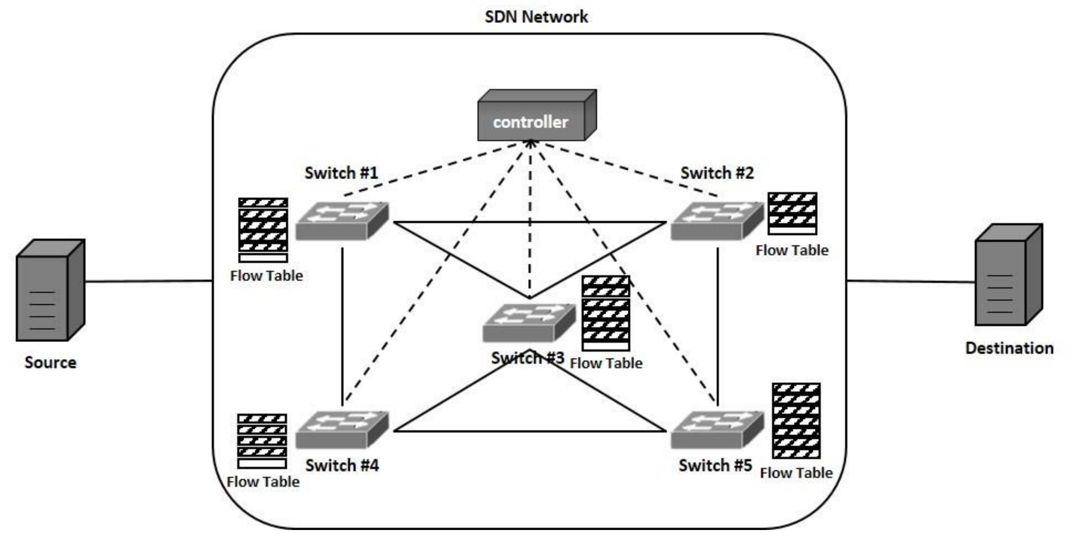

Figure 5. Network Condition after P-LRU Scheme is Applied

Figure 5 shows how spare space is secured at the flow table of switches within path (1) by applying the suggested P-LRU scheme. When a packet arrives at switch 1, it looks for a flow matching the packet. If there is no matching flow, it sends out a *PacketIn* message to the controller to query flow entry generation. The controller computes the path and sends a *FlowModify* message to switch 1 to order the addition of a new flow entry. It also sends a *PacketOut* message to designate the packet's destination. However, since switch 1's table is already full, switch 1 converts the flow deletion request into a *FlowRemoved* message and sends it out to the controller. The controller then determines which flow to delete according to the LRU algorithm and sends out a *FlowModify* message to switch 1 as well as to switches 2 and 4 at the same time. After that, the flows are deleted from switches 1, 2, and 4 resulting in spare space for additional flows in all switches within the path (1). Since switch 3 already has spare space, its flow is maintained.

Table 1. Overhead Comparison with P-LRU

Scheme	Switch 1		Switch 4		Switch 3		Switch 2		Total Overhead
	M.O	C.O	M.O	C.O	M.O	C.O	M.O	C.O	
Current	$5T_X$	$1T_C$	$5T_X$	$1T_C$	$3T_X$	$0T_C$	$5T_X$	$1T_C$	$18T_X + 3T_C$
P-LRU	$5T_X$	$1T_C$	$4T_X$	$0T_C$	$3T_X$	$0T_C$	$4T_X$	$0T_C$	$16T_X + 1T_C$

Table 1 is an overhead comparison table between the existing process and P-LRU. The overhead created when a message is transmitted once between the controller and switch is assumed to be $1T_X$ and the overhead created when the controller calculates the flow to delete is assumed to be $1T_C$. M.O. is the overhead occurring when messages are exchanged and C.O. is the controller overhead occurring when determining the flow to be deleted. In the existing process, there are 5 message exchanges ($5T_X$) for flow generation and table replacement and there is 1 flow computation ($1T_C$) to determine which flow to delete from the table at switches 1, 4, and 2 where the flow table is full. At switch 3, there are only 3 message exchanges needed for flow generation and insertion. Hence the total overhead of $18T_X$+$3T_C$ occurs. On the other hand, with P-LRU, messages for flow generation and table replacement are exchanged 5 times($5T_X$) and the flow computation for flow deletion is done once ($1T_C$) at switch 1. At switches 4 and 2, there is one flow deletion message ($1T_X$) and 3 message exchanges ($3T_X$) needed for flow generation and insertion. In total, 4 message exchanges are made ($4T_X$). Since spare space is already secured by deleting flows, there is no need for computation to be done to determine which flow to delete ($0T_C$). Since switch 3 also has space in the table, it only needs 3 message exchanges ($3T_X$) needed for flow generation and insertion.

When P-LRU is applied as shown in Table 1, the overhead created from message exchange and controller computations decreases and is proven to be more efficient than the existing method where the flow table is updated every time by message exchanges between a switch and the controller.

5. Performance Evaluation

5.1. Experiment Setup

In order to evaluate the performance of P-LRU suggested in this paper, an SDN composed of linear technology having N sources and P destinations, as shown in Figure 6, was constructed by utilizing the Mininet Emulator [10] and a simulation was performed with this architecture. Sources worked as video and web servers while the destinations worked as clients. A Pox Controller [11] was used for the SDN controller.

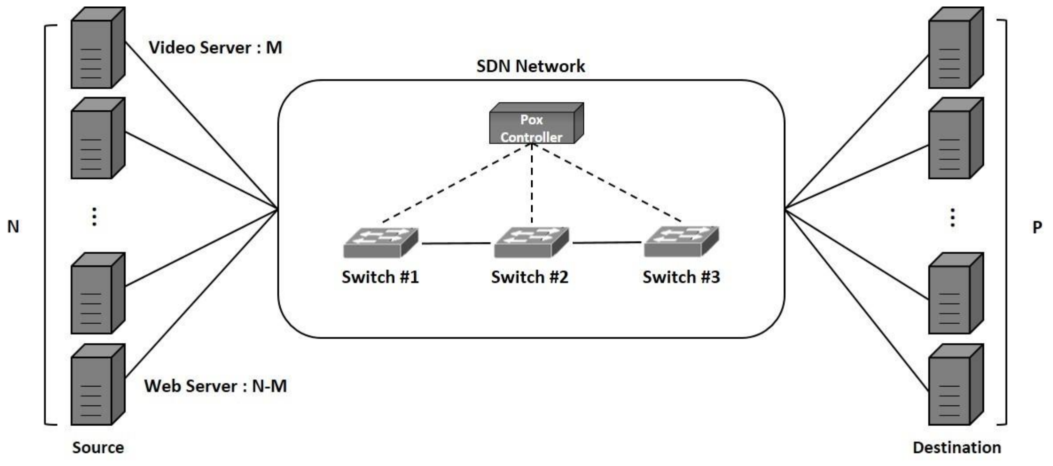

Figure 6. Test Topology Configuration

The packet data set needed for algorithm performance evaluation was generated by the Ostinato Packet Generator Tool [12].

Table 2. Data Set Configuration

Traffic	V1W9	V3W7	V5W5	V7W3	V9W1
Number of packets	3.3K	3.3K	3.3K	3.3K	3.3K
Video Server	10%	30%	50%	70%	90%
Web Server	90%	70%	50%	30%	10%

Table 2 represents the configuration of the packet data set. Each data set is composed of 3300 packets. In each data set, the video and web servers are composed of 10 sources at the ratios given in table 2. For example, the server configuration of the V1W9 data set is one video server and 9 web servers where 1 video server sends packets to 1 client only while 9 web servers send packets to 9 clients randomly.

Table 3 is the table size, the number of video servers, the number of web servers, and the number of clients set up during the test using data sets. The table size represents switches 1, 2, and 3 in chronological order.

Table 3. Configurable Parameters for Data Sets

	V1W9	V3W7	V5W5	V7W3	V9W1
Table Size	10, 13, 16	10, 13, 16	10, 13, 16	10, 12, 14	10, 13, 16
Number of video servers	1	3	5	7	18
Number of web servers	9	7	5	3	2
Number of clients	10	10	10	10	20

For example, the table size of V1W9 is set as 10, 13, and 16, so the table size of switches 1, 2, and 3 are 10, 13, and 16, respectively. The reason why V7W3's table size is set as 10, 12, and 14 is because there will not be any flow table replacement at switch 3 since only 16 flows are created. Tests were done with different flow table sizes of the switches in order to verify the low table replacement value. The number of video servers, web servers, and clients were determined according to the ratio from the Table 2. The reason why the number of video servers and web servers were set as in Table 3 is because it is not possible to check the number of replacements if we set the number of video servers as 9 and web servers as 1, since there would only be 10 flows created and no flow table replacement would take place. The data set used for the experiment was created by referring to Table 3 and Table 4 [13].

Table 4. Some of the Video Traffic Model Parameters

Information Types	Number of packets in a frame	Packet size	Inter-arrival time between packets
Distribution	Deterministic	Truncated Pareto (Min = 50bytes, Max = 125bytes)	Truncated Pareto (Min = 6ms, Max = 12.5ms)

Table 5. Parts of HTTP Traffic Model Parameters

Component	Distribution	Parameters	PDF
Number of embedded objects per page (N_d)	Truncated Pareto	Mean = 5.64 Max = 53	Note: Subtract k from the generated r.v. to get N_d $f_x = \frac{\alpha k^{\alpha}}{x^{\alpha+1}}, k \le x < m$ $f_x = \left(\frac{k}{m}\right)^{\alpha}, x = m$ $\alpha = 1.1, k = 2, m = 55$
Reading time (D_{PC})	Exponential	Mean = 30 sec	$f_x = \lambda e^{-\lambda x}, x \ge 0$ $\lambda = 0.033$
Parsing time (T_P)	Exponential	Mean = 0.13 sec	$f_x = \lambda e^{-\lambda x}, x \ge 0$ $\lambda = 7.69$

Table 4 represents the parameters of the video traffic model. The size and Inter-arrival Time of the packets sent out of the video servers follow a truncated Pareto distribution and have a characteristic of having a constant number of packets.

Table 5 represents the HTTP Traffic Model parameters. Web browsing traffic has its objects per page following a truncated Pareto distribution and has a characteristic of its

Reading Time following the exponential distribution [13] [14]. The video traffic and web browsing traffic data set was created according to such characteristics and the data set created by using the Bit-Twist packet replay engine [15] was made to replay on the Mininet topology. Src IP address, Dst IP address, Src MAC address, and Dst MAC address that are all parts of the flow table were used as the matching fields to compare the packet arriving at the switch with the flow. Table 6 shows some parts of the flow table.

Table 6. Parts of Flow Table Header Information in OpenFlow

Flow Number	Match Fields			
	Src IP	**Dst IP**	**Src MAC**	**Dst MAC**
1	10.0.0.1	10.0.0.2	00:00:00:00:00:01	00:00:00:00:00:02
2	10.0.0.1	10.0.0.3	00:00:00:00:00:01	00:00:00:00:00:03
3	10.0.0.1	10.0.0.4	00:00:00:00:00:01	00:00:00:00:00:04
4	10.0.0.1	10.0.0.5	00:00:00:00:00:01	00:00:00:00:00:05

If a flow table like Table 6 exists and a packet arrives at the switch, Src IP address, Dst IP address, Src MAC address and Dst MAC address of the packet are compared to check whether any flow within the table has the same field values. If they do, they are considered as the same flow and processed accordingly. For example, if a packet has information of Src IP address:10.0.0.1, Dst IP address:10.0.0.4, Src MAC address:00:00:00:00:00:01, Dst MAC address:00:00:00:00:00:04, it is processed as the same as the flow number 3 of the flow table.

5.2. Experiment Results

In order to evaluate the performance of the P-LRU algorithm suggested in this paper, the FIFO, LRU and LFU algorithms, which are the existing flow replacement algorithms, were configured as controller applications and applied to the virtual SDN network for performance comparison.

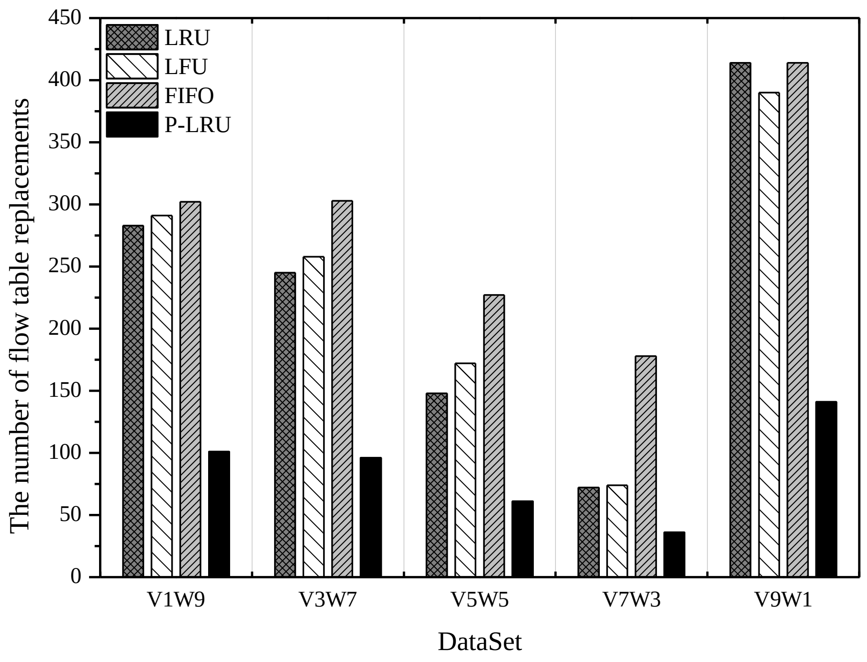

Figure 7. The Number of Flow Table Replacements for each Algorithm

Table 7. The Number of Flow Table Replacements of LRU and P-LRU

	V1W9	V3W7	V5W5	V7W3	V9W1
LRU	283	245	148	72	414
P-LRU	101	96	61	36	141
Reductio n Ratio	64.4%	60.9%	58.8%	50%	66%

Figure 7 is a comparison of the number of table replacements. The number of table replacements refers to the number of occurrences of deleting a flow and inserting a new flow. This is more efficient if the number is smaller. Since the table size of the switch was set as 10, 13 and 16, flow table replacement of switch 1 was more frequent than that of switches 2 or 3. In P-LRU, there was spare space in the tables of switches 2 and 3 even though the flow table of switch 1 was replaced and hence there were no table replacements in switches 2 and 3. However, if the tables of switches 2 and 3 were full, the flows at switches 2 and 3 were also deleted when a table replacement took place at switch 1. Since the flow table replacement only occurred at switch 1, and switches 2 and 3 only had flow deletions, P-LRU's flow table replacement was much smaller than that of the other algorithms.

As mentioned in Section 5-1, if 9 video servers transmit packets 1-on-1 to 9 clients and 1 web server transmits 1-on-1 to one client, there will be 10 flows created. However, the test environment of this paper has set the table size of the switches as 10, 13, and 16 so when there were only 10 flows created, table replacement did not take place. Hence, the topology was changed to allow more flows than the table size.

The table replacement numbers of V9W1 showed an overall increase since the topology was changed to 1 video server, 1 web server and 20 clients where the video servers transmitted packets to 18 clients, the web server transmitted to 2 clients so as to create 20 flows in total.

The reason why the number of flow table replacements increased overall in V9W1 is because more flows were created by adjusting the number of video servers to be 18 and web servers to be 2, as shown in Table 3. This data set repeats transmitting video traffic to 18 clients and transmitting web traffic to 2 clients. Other data sets repeat transmitting to 10 clients with different flows but the V9W1 data set transmits the traffic with 20 different flows. In other words, the V9W1 data set creates more flows than the other data sets when their flow table sizes are the same and hence results in more frequent changes of the flow table and an increased number of flow table replacements. Nonetheless, the reason why the number of P-LRU replacements did not increase greatly is because spare space at flow tables was secured when a flow at one switch was deleted, the same flow was also deleted from all other switches that had full flow tables.

Table 7 shows numeric comparisons between the number of table replacements of the LRU algorithm and the P-LRU scheme. In each data set, P-LRU showed approximately 50-65% reduction in table replacements compared to LRU.

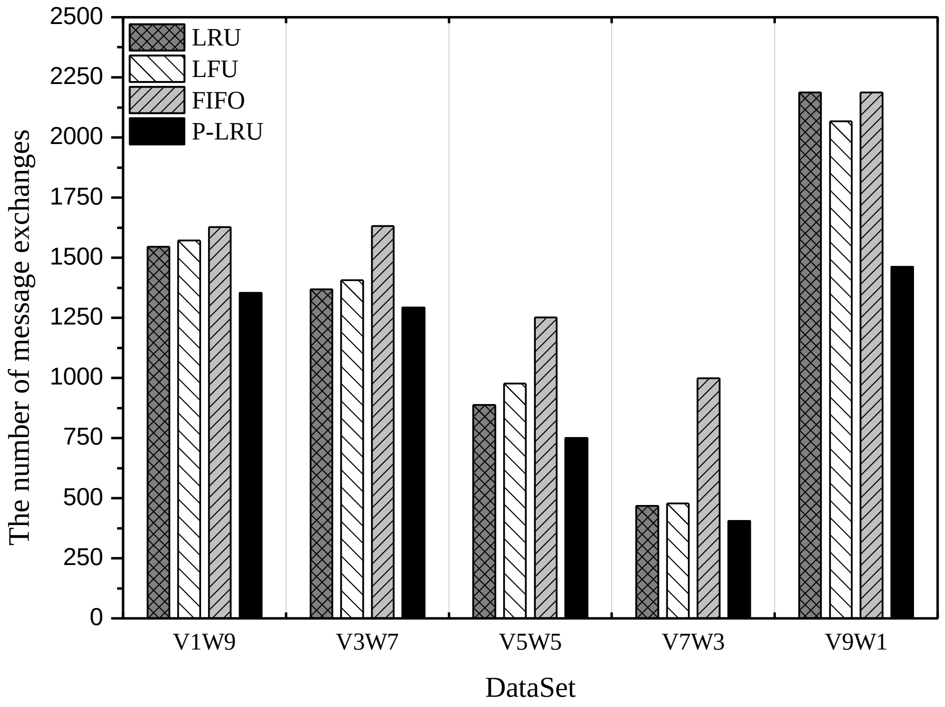

Figure 8. The Number of Message Exchanges between Controller and Switches for Each Algorithm

Table 8. The Number of Message Exchanges between Controller and Switches in LRU and P-LRU

	V1W9	V3W7	V5W5	V7W3	V9W1
LRU	1546	1368	888	468	2187
P-LRU	1354	1293	750	405	1462
Reduction Ratio	12.5%	5.5%	15.6%	13.5%	33.2%

Figure 8 is a comparison of the number of message exchanges between the controller and switches when transmitting each data set. If a flow related to the packet does not exist in the table, the controller and the switch exchange 3 messages: *PacketIn*, *FlowModify*, and *PacketOut*. Two additional messages, namely *FlowRemoved* and *FlowModify*, are further exchanged when the flow is replaced. Hence, 5 messages are exchanged in total. In existing algorithms, each switch manages its own flow table separately but in P-LRU, when a flow is deleted at the switch with the smallest flow table size, the same flow is also deleted from the flow tables of all other switches that have full flow tables. Therefore, table replacement only occurs at the switch with the smallest table size and this results in reduced number of message exchanges in P-LRU.

The reason why there were more message exchanges in the V9W1 data set than other data sets is because the flow table size of the switch was the same but more flows were created and resulted in more flow table replacements.

Table 8 numerically compares the number of message exchanges between controller and switch in LRU algorithm and P-LRU. In each data set, P-LRU displayed 5-33% reduction compared to LRU.

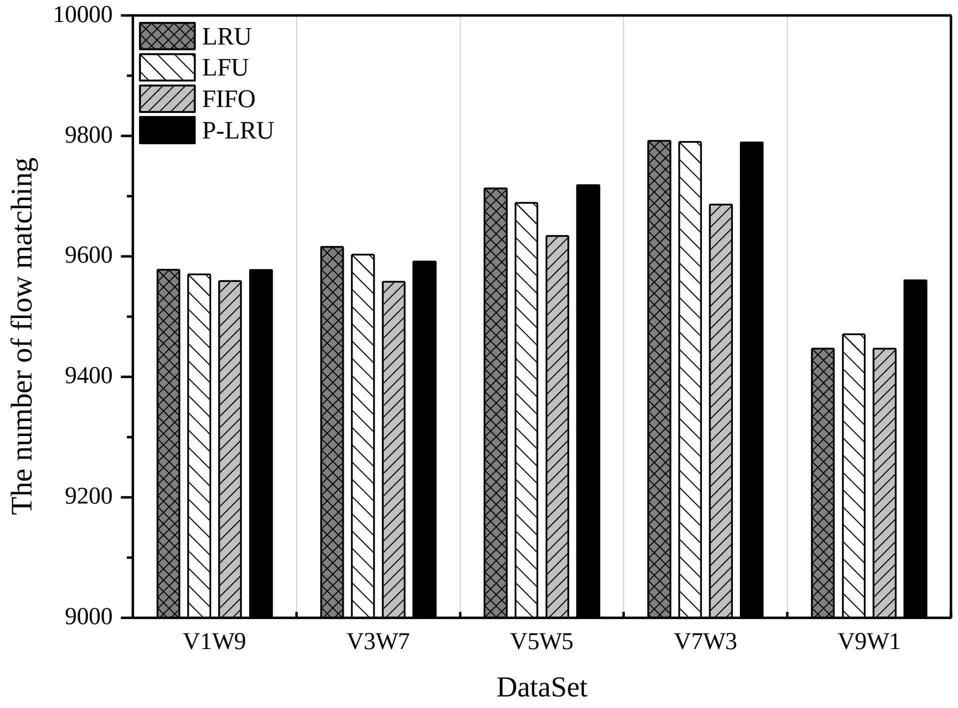

Figure 9. The Number of Flow Matching for each Algorithm

Figure 9 compares the flow matching numbers of existing algorithms against P-LRU. The reason why the number of flow matching increases as the ratio of video traffic increases in the data sets except V9W1 is because the cycle of video traffic becomes smaller and reduces the number of video traffic flows being deleted from the flow table. The reason why the number of each algorithm's flow matching decreased in the V9W1 data set was because the number of video servers increased to 18, as shown in Table 3. The number and period of video traffic flow increased because the number of video servers increased while the flow table sizes of other switches stayed the same as the other data sets. Therefore, the video traffic flow maintained at the flow table was not constant and the overall number of flow matching decreased.

The reason why the flow matching numbers of LRU and P-LRU are similar in other data sets except V9W1 is because the operating method of P-LRU is based on the LRU method. However, the reason why the flow matching numbers of P-LRU is higher than that of LRU with V9W1 is because P-LRU deletes the flow from all switches with full flow table when deleting the flow at one switch. Figure 10 and 11 display the process of LRU and P-LRU when the table size of the switches is 5, 6, and 7 each. S1, S2, and S3 in the figure represent switches 1, 2 and 3, respectively, whereas F1 to F9 represent the flows. LRU deletes the oldest one after matching at each switch every time, but P-LRU deletes the oldest ones after matching at all other switches as well. By deleting the same flow deleted at switch 1 from all other switches, the flow created previously can be maintained in the flow table. For example, V9W1 is a data set that repeatedly sends out 18 video transmissions and 2 web transmissions in such a way that packets belonging to individual transmissions are interleaved sequentially. In other words, all packets belonging to the 18 video transmissions are sent in sequence, followed by the packets pertaining to the two web transmissions in sequence. The sequence is then repeated with 18 video packets sent in sequence, and so on. Furthermore, V9W1 has a long cycle of video traffic transmission. That is, the time interval between consecutive packets belonging to the same video traffic is long, so the LRU algorithm works like FIFO as shown in Figure 10. But for P-LRU, Figure 11 shows that when 6, 2, 3, 4, 5 flows within the flow table of switch 1 are replaced by 6, 7, 8, 4, 5, flow number 3 is deleted at switch 1 as well as at switch 2 and switch 3. At this stage, flow 1 is maintained at switch 2 while flows 1 and 2 are maintained at switch 3. Since the flows maintained without deletion at the flow tables can be re-matched when video packets are once transmitted and begin transmitting again, the number of flows matched by LRU is larger than that by P-LRU when there are large video transmissions.

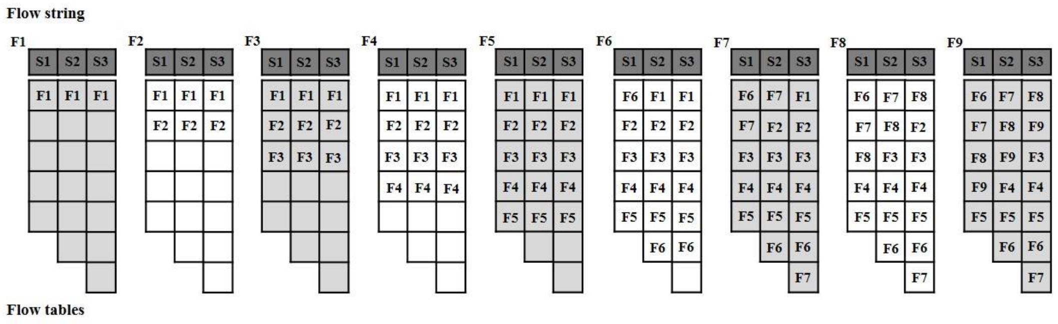

Figure 10. Flow Table Replacement by the LRU Algorithm

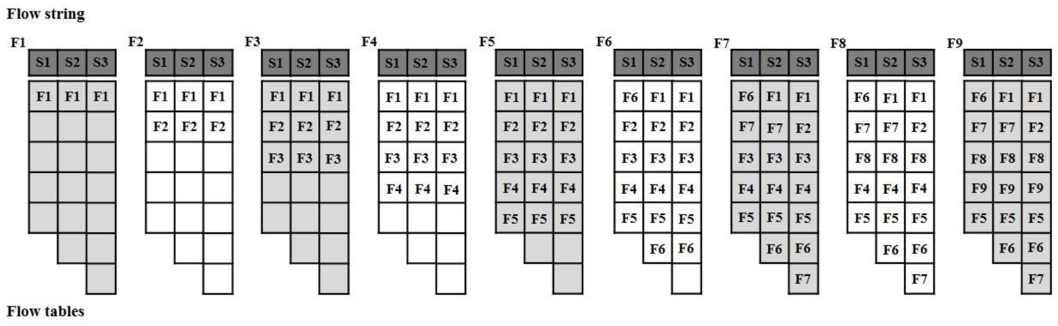

Figure 11. Flow Table Replacement by the P-LRU Algorithm

6. Conclusion and Future Work

This paper has explained the problem of flow table overflow and the controller overhead increase when an SDN network is composed of a number of switches with different performances. The study has also suggested a Proactive-LRU flow table management scheme in order to reduce this problem. Proactive-LRU is a flow table management method wherein when a flow is deleted at a switch and the flow tables of other switches having the same flow are full, that flow is also deleted at all those switches. The existing replacement algorithms, namely LRU, LFU, and FIFO were configured as controller applications and the performance of all algorithms were compared to evaluate the performance of P-LRU. When P-LRU was compared against LRU, which performs the best among the LRU, LFU, and FIFO algorithms, it showed reductions in the number of flow table replacements by 50-65%, in controller computation by 50-65% and in the number of message exchanges for flow creation and flow table replacements by 5-33%.

As future work, a method to increase flow table matching numbers and to decrease the overhead of the controller within the P-LRU algorithm will be investigated. Furthermore, an efficient flow table management scheme according to traffic pattern will be researched and the performance of all these techniques will be evaluated against previous studies.

Acknowledgments

This work is supported by Kyonggi University Research Grant 2015.

References

[1] ONF, "Software-Defined Networking: The New Norm for Networks," ONF White Paper, **(2012)**, pp. 7.

[2] S. Sezer, S. S. Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks", IEEE Communications Magazine, vol. 51, no. 7, **(2013)**, pp. 36-43.

[3] P. Dely, A. Kassler and N. Bayer, "Openflow for wireless mesh networks", Computer Communications and Networks (ICCCN), **(2011)**.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks", ACM SIGCOMM Computer Communication Review, vol. 41, **(2011)**, pp. 254-265.

[5] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman and R. Kompella, "Towards an elastic distributed SDN controller", ACM SIGCOMM Computer Communication Review, **(2013)**.

[6] S. H. Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications", Proceedings of the first workshop on Hot topics in software defined networks, **(2012)**, pp. 19-24.

[7] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow", Proceedings of the 2010 internet network management conference on Research on enterprise networking, **(2010)**, pp. 3-3.

[8] S. Banerjee and K. Kannan, "Tag-in-tag: Efficient flow table management in sdn switches", Network and Service Management (CNSM), **(2014)**, pp. 109-117.

[9] K. Agarwal, "Shadow macs: Scalable label-switching for commodity ethernet", Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, **(2014)**, pp. 157–162.

[10] B. Lantz, B. Heller and N. McKeown, "A network in a Laptop: Rapid Prototyping for Software-Defined Networks", HotNets, **(2010)**.

[11] POX, Python Network Controller. http://www.noxrepo.org/pox/about-pox/.

[12] ostinato.org. Ostinato: Packet/traffic generator and analyzer. Technical report, **(2011)**.

[13] "802.20 Evaluation Methodology Strawman-00", Project of IEEE 802.20 Working Group on Mobile Broadband Wireless Access, C802.20-03/43, 2003.

[14] B. Chandrasekaran, "Survey of network traffic models", Washington University in St. Louis CSE, 567, **(2009)**.

[15] Bit-Twist. http://bittwist.sourceforge.net/.

Authors

Dongryeol Kim, he was born in Ansan, Korea, in 1990. He received the B.S. degree in Computer Science from the Kyonggi University, Suwon, Korea, in 2015, and He is working on a master's degree in computer science. His current research interest includes SDN and OpenFlow protocol.

Byoung-Dai Lee, he is an associate professor at the department of Kyonggi University, Korea. He received his B.S. and M.S. degrees in Computer Science from Yonsei University, Korea in 1996 and 1998 respectively. He received his Ph.D. degree in Computer Science and Engineering from University of Minnesota, Minneapolis, U.S.A. in 2003. Before joining the Kyonggi University, he worked at Samsung Electronics, Co., Ltd as a senior engineer from 2003 to 2010. His research interests include digital broadcast systems, multimedia streaming systems, and networked multimedia communications systems. He is the corresponding author of this paper.