IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 20, 2017, accepted October 27, 2017, date of publication November 13, 2017, date of current version December 5, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2771807

Enhanced Flow Table Management Scheme With an LRU-Based Caching Algorithm for SDN

EUN-DO KIM[1], YUNCHUL CHOI[2], SEUNG-IK LEE[2], AND HYOUNG JUN KIM[2]

[1]Information and Communication Technology, University of Science and Technology, Daejeon 34113, South Korea
[2]Protocol Engineering Center, Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea

Corresponding author: Eun-Do Kim (maniada@etri.re.kr)

This work was supported in part by the Institute for Information and Communications Technology Promotion through the Korea Government (MSIT) under Grant R7116-16-1004 and in part by the Standards Development of IoT Application Services and Interoperability Support.

ABSTRACT We propose a flow table management scheme for OpenFlow switches to minimize table-miss in a flow table. Commercial OpenFlow switches have one or more flow tables that consist of flow entries for processing packets. However, flow entries are managed based only on their timeout parameters, meaning they are expired and deleted by switches regardless of their reusability. The absence of the flow entry then causes table-misses in the future. We propose a solution for the problem of switches disregarding vacancies in the flow table when deleting expired flow entries. When a table-miss occurs, the corresponding switch must perform additional interactions with an OpenFlow controller to insert new flow entries, but this results in additional processing time and communication overhead. Previous studies aimed at solving this limitation have used sophisticated training sets or modified network architectures. In contrast, we propose a simple flow table management scheme with a least recently used-based caching algorithm to keep flow entries in an OpenFlow switch as long as possible. For this purpose, the switch continually adjusts its cache size according to the vacancy of each flow table and the controller determines the packet forwarding path through the switches by referring to the vacancies. We perform an experimental evaluation of the proposed scheme and demonstrate the performance gains in terms of the number of table-missed packets. We also analyze the effectiveness of the flow entry caching scheme using a mathematical model.

INDEX TERMS Flow table, OpenFlow, software-defined networking.

I. INTRODUCTION

In recent years, Software-Defined Networking (SDN) [1] has emerged as an advanced networking technology that decouples the control plane and data plane of each network switch. To control networks in a simple and programmable manner, an SDN architecture is designed such that distributed data planes are managed by a centralized control plane. OpenFlow [2] is one of the communication protocols used for interactions between the control plane and data plane in SDN. By using the OpenFlow protocol, a centralized OpenFlow controller, which plays the role of the control plane, applies packet forwarding policies to distributed OpenFlow switches, which define the data plane. Fig. 1 presents a layered view of an SDN architecture and illustrates the differences between an SDN architecture and traditional network architectures.

In SDN environments, incoming packets are processed according to the rules defined in SDN applications on an OpenFlow controller. Rules can be inserted, modified, or deleted by the controller in the form of flow entries in the flow table of an OpenFlow switch. The main components of a flow entry consists are a subset of the header fields of the packet to be matched (i.e., match fields), the instructions for processing the matched packet (i.e., instructions), and idle time before the flow entry is expired by the switch (i.e., timeouts). By leveraging the flow entries in the flow table, the switch has two methods for processing incoming packets. In the first method, the incoming packet whose header fields match the match fields in the flow entry is processed according to the instructions defined in the flow entry. This is called a proactive process and the switch does not need to interact with the controller to process the packet. However, when the header fields of the incoming packet do not match any match fields of the existing flow entries in the flow table, we call it a table-miss. To handle the table-missed packet, the switch must perform a reactive process that interacts with the controller to insert new flow entries

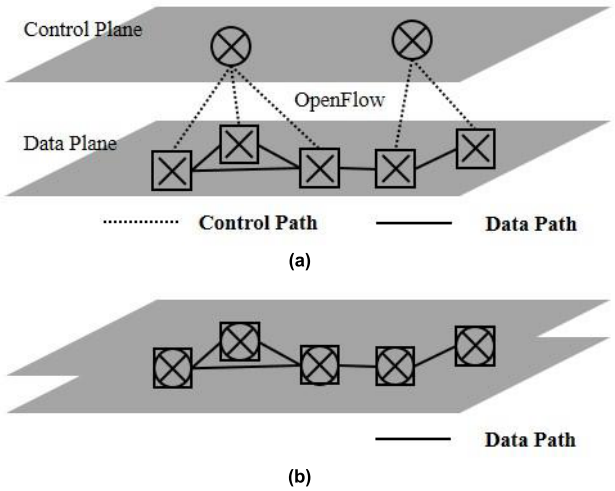

FIGURE 1. Layered view of (a) an SDN architecture with an OpenFlow communication protocol and (b) a traditional network architecture. In the SDN architecture, the control plane and data plane of each network switch are decoupled. They interact by using the OpenFlow protocol.

to process the packet. Fig. 2 illustrates the different steps between the proactive and reactive processes of an OpenFlow switch. One can verify that the reactive process performs additional interactions with an OpenFlow controller based on the table-miss flow entry in each flow table.

If the number of table-missed packets increases, the switch must interact with the controller more often, which may result in additional processing time and communication overhead. In a study by Fernandez *et al.* [3], the experimental results of a performance comparison in terms of throughput (i.e., flows per second) indicated that the proactive process yields higher performance than the reactive process in various scenarios. Because of the frequent interactions with the controller, the throughput difference also becomes larger as the number of switches increases.

We have previously addressed the above limitations caused by frequent table-misses and analyzed the flow table management scheme for existing OpenFlow switches [4]. A flow entry expires if its match fields do not match with the header fields of incoming packets within an idle timeout period. The switch then deletes the expired flow entry without considering vacancy in the flow table. To keep the flow entries in a flow table as long as possible, we propose a modified flow table management scheme that caches expired flow entries as inactive entries rather than deleting them. In our scheme, inactive flow entries are managed based on a Least-Recently-Used (LRU) caching algorithm for their future replacement. An OpenFlow controller determines packet forwarding paths by referring to vacancies in the corresponding flow tables to maximize usage of the cached flow entries. As a result, the switch can handle more incoming packets using the proactive process instead of the reactive process, which results in performance gains in terms of the number of table-missed packets. To analyze the effectiveness of the flow entry caching model, we define a mathematical model and compare its results with the experimental results for verification.

The remainder of this paper is organized as follows. In Section II, we address the limitations of the flow table management scheme for existing OpenFlow switches, which requires additional processing time and communication overhead because of frequent table-misses. Several previous studies aimed at solving this problem are introduced in Section III. We then propose an enhanced flow table management scheme with an LRU-based caching algorithm for SDN in Section IV. For evaluation of the proposed scheme, we define a mathematical model that predicts the table-miss rate of an OpenFlow switch in Section V and present the experimental results in terms of the number of table-missed packets in Section VI. Finally, our conclusions are summarized in Section VII.

II. PROBLEM DEFINITION

When a packet arrives at an OpenFlow switch, the packet is handled using either the proactive or reactive process depending on the presence of the corresponding flow entries in a flow table. Although the proactive process handles the incoming packet much faster than the reactive process, the switch immediately deletes the expired flow entries even if the vacancy in the flow table is sufficient to keep them for future reuse. In this case, if a packet whose header fields match the match fields of the recently deleted flow entry arrives at the switch, the packet will lose the opportunity to be handled by the proactive process because of a table-miss. For example, all non-periodic packets, such as HTTP or instant messaging traffic, are reactively processed if their packet intervals are longer than the idle timeouts of the corresponding flow entries. As a result, a fixed timeout parameter for deleting expired flow entries is not appropriate in some cases. It also increases the table-miss rate in the flow table of a switch.

According to Nguyen *et al.* [5], up to 8 million rules are required to support all essential policies in typical enterprise networks, but commercial OpenFlow switches can only store up to 16 thousand flow entries in a flow table [6]. Kobayashi *et al.* [7] and Kreutz *et al.* [8] also mentioned that a limitation of existing OpenFlow switches is that flow table capacity is not sufficient to cover all incoming packets in emerging networks. Although this is the reason why switches immediately delete expired flow entries to create vacancies for new flow entries, the switch can only store a limited number of flow entries, regardless of its current vacancy. As a simple alternative, an OpenFlow controller can set the idle timeout of each flow entry long enough that the switch keeps more flow entries in the flow table, but this may cause flow table overflow when the number of incoming packets suddenly increases.

Since an initial OpenFlow switch specification [9] was released, the flow table overflow that results from a flow table exceeding its finite capacity has been a recurring problem. In earlier versions of the specification, new flow entries were no longer inserted when the flow table of a switch was full, which caused as disruption of service. To allow an OpenFlow controller to avoid filling the flow table, OpenFlow switch specification 1.4.0 [10] introduced the concept of vacancy

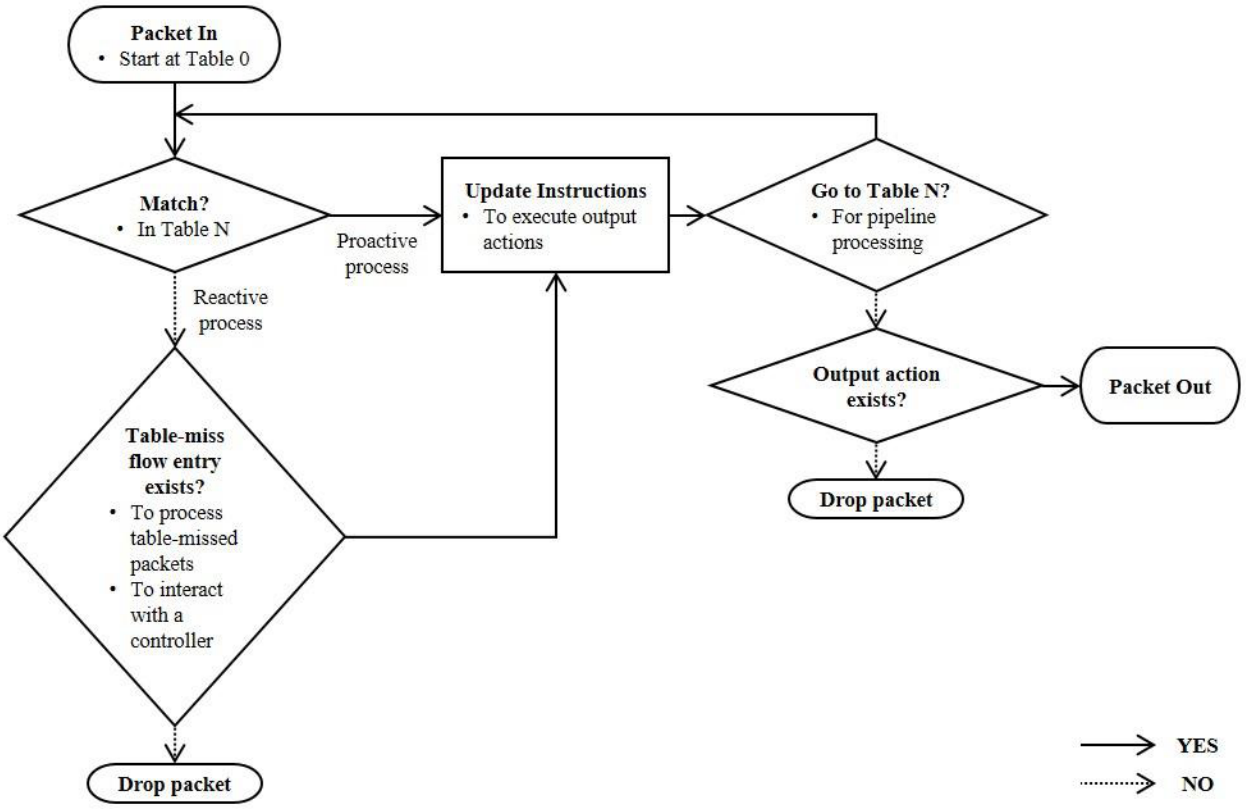

FIGURE 2. Simplified flow chart of packet processing in an OpenFlow switch. Compared to a proactive process, a reactive process passes through one additional step before updating instructions to process incoming packets.

events, which enable the controller to receive early error messages based on the capacity threshold of the switch. In our flow table management scheme, an OpenFlow switch exploits the vacancy parameter of vacancy events to apply the LRU-based caching algorithm for SDN.

III. RELATED WORKS

We have focused on the fundamental limitation of existing OpenFlow switches having insufficient flow table capacity to cover all policies in typical network environments. To solve the problem of limited capacity, the following works have proposed various solutions.

Curtis *et al.* [11] and Lee *et al.* [12] classified incoming packets into elephant flows and mice flows according to their flow patterns. An OpenFlow switch selectively interacts with an OpenFlow controller in their models. Although both models cache flow entries to potentially reduce interactions with the controller, their caching policies are slightly different. In [11], the rules for the mice flows are aggregated according to a set of least-energy paths before being inserted as flow entries and the rules for the elephant flows are individually managed as flow entries. In [12], the cached flow entries for elephant and mice flows are managed separately and the switch does not share the cache memories for those flows. These two approaches provide an alternative method to reduce interactions with the controller by classifying incoming packets according to their flow patterns, but they still require sophisticated training sets for elaborate classification.

In the study by Yu *et al.* [13], authority switches were placed between OpenFlow controllers and OpenFlow switches as intermediate components. They duplicate authority rules from the controller to reduce additional interactions when inserting new flow entries. As an extra OpenFlow switch, the authority switch handles table-missed packets according to predefined flow entries whose rules are aggregated. Although table-missed packets can be processed by predefined flow entries without any interactions with the controller, this model requires large changes to the existing OpenFlow network model for SDN.

Zhu *et al.* [14] and Kim *et al.* [15] proposed an approach in which the idle timeout of each flow entry in a flow table is adjusted dynamically based on corresponding flow patterns. For this purpose, an OpenFlow controller collects various flow patterns from OpenFlow switches and predicts optimized idle timeouts based on packet intervals. To prevent excess communication overhead due to excessive table-missed packets, the controller inserts flow entries with long idle timeouts for periodic flows that have long packet intervals. Their approaches to minimizing the number of flow entry deletions are well suited to overcoming the drawbacks of static idle timeouts, but they come with a tradeoff between

accurate prediction for optimizing idle timeouts and communication overhead for collecting flow patterns from the switch.

To evaluate performance in terms of table-miss rates and idle timeouts, Zarek [16] applied a caching policy to existing OpenFlow switches and compared the performance of LRU, FIFO, and random caching algorithms for replacing the flow entries in a flow table of a switch. According to the experimental results, table-miss rate decreases as flow table capacity increases and the LRU caching algorithm outperforms the other algorithms tested. This is why we chose an LRU-based caching algorithm to manage expired flow entries in a flow table whose capacity is fixed. By exploiting the vacancy in a flow table to temporarily keep expired flow entries for future reuse, we are able to minimize table-missed packets.

IV. PROPOSED SCHEME

Because of their limited flow table capacity, existing OpenFlow switches immediately delete expired flow entries, regardless of the current vacancy in the flow table. As mentioned in Section III, there have been many previous studies aimed at solving this problem, but they have significant side effects, such as requirements for sophisticated training sets or large changes to the OpenFlow network model for SDN. As a much simpler approach, we propose a modified flow table management scheme to keep flow entries in a flow table of a switch as long as possible. The goal of the proposed scheme is to minimize table-misses in flow tables. To exploit the idle space in a switch, we use the vacant room in flow tables to cache expired flow entries instead of deleting them. Additionally, we determine packet forwarding paths by referring to the vacancies in corresponding flow tables such that every flow table maximizes the use of cached flow entries.

A. LRU-BASED CACHING ALGORITHM

Fig. 3 presents the pseudo code for both the LRU-based caching algorithm that we propose and the traditional approach. To apply the LRU policy to inactive flow entries in a simple manner, we use a unique non-zero age parameter for each inactive flow entry in a flow table. The age sequentially increases each time the switch caches an expired flow entry as an inactive flow entry. The inactive flow entry with the oldest age is considered to be the least recently used flow entry. In the proposed scheme, an inactive flow entry is deleted when its age reaches a certain threshold, rather than when its idle timeout expires. To minimize idle space in flow tables, the switch continually adjusts the threshold for each flow table based on its vacancies over the past few seconds. The idle space is used as variable cache memory. Fig. 4 illustrates the differences in flow table usage over time between the proposed scheme and the traditional approach. If the idle space is filled with inactive flow entries in the proposed scheme, the cache size of the flow table can be continuously adjusted to prevent flow table overflow because of its variable size.

```
rule_expire(all flow entries in m-th flow table) {
   ...
   vacancy = update_vacancy(flow_table[m]);
   threshold = vacancy * weight;
   for(all i-th active flow entries)
      if(flow_entry[i].idle_timeout expires)
         set_inactive(flow_entry[i]);
         ++flow_entry[i].age;
         for(all j-th inactive flow entries)
            if(++flow_entry[j].age > threshold)
               delete(flow_entry[j]);
   ...
}

flow_used(n-th flow entry) {
   ...
   set_default(flow_entry[n].idle_timeout);
   update_statistics(flow_entry[n]);
   if(flow_entry[n].age)
      set_active(flow_entry[n]);
      flow_entry[n].age = 0;
   ...
}
```

(a)

```
rule_expire(all flow entries in every flow table) {
   ...
   for(all i-th flow entries)
      if(flow_entry[i].idle_timeout expires)
         delete(flow_entry[i]);
   ...
}

flow_used(n-th flow entry) {
   ...
   set_default(flow_entry[n].idle_timeout);
   update_statistics(flow_entry[n]);
   ...
}
```

(b)

FIGURE 3. Pseudo code for (a) an LRU-based caching algorithm for replacing inactive flow entries and (b) a traditional algorithm for deleting expired flow entries. As a simple modification, every flow entry has its own age and an OpenFlow switch applies the LRU policy with a threshold to inactive flow entries.

Meanwhile, an inactive flow entry whose match fields match with any header fields of incoming packets becomes active again. This process does not require any interactions between an OpenFlow controller and OpenFlow switch. In this case, the age of the inactive flow entry is reset from non-zero to zero and the flow entry is handled as an active flow entry with its initial state again.

B. VACANCY-BASED PACKET FORWARDING POLICY

An OpenFlow controller determines packet forwarding paths according to the rules defined in SDN applications. The shortest path between network nodes is found using Dijkstra's algorithm [17], which calculates the distance from a single source node to a single destination node. In Dijkstra's algorithm, all packet forwarding paths between network nodes are represented by a weighted graph composed of distance vectors, where each vector has a distance and direction. Based

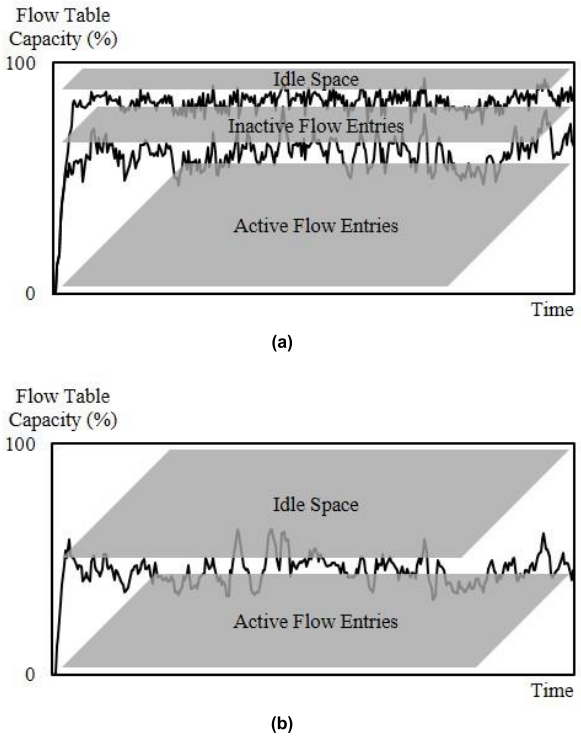

FIGURE 4. Example of flow table usage over time in (a) the proposed scheme to exploit idle space and (b) the traditional approach. To cache inactive flow entries in the variable space without flow table overflow, the switch continually adjusts the thresholds for all flow tables according to vacancies over the past few seconds.

on this weighted graph, SDN applications for determining optimized packet forwarding paths are designed to find the shortest distance and their rules are applied through the controller as a form of flow entry when the corresponding packet arrives at the switch.

To cache more inactive flow entries in a flow table, we determine the optimal packet forwarding paths between network nodes using Dijkstra's algorithm with additional consideration for the vacancies in flow tables. As an additional parameter to calculate distance vectors, higher vacancies are considered as higher weight factors for finding the shortest path. In our scheme, inactive flow entries are evenly stored in adjacent switches as not only alternative paths for forwarding but also as extra spaces for caching.

V. MATHEMATICAL MODEL

To demonstrate the effectiveness of the proposed LRU-based caching algorithm, we introduce a mathematical model that predicts the table-miss rate of OpenFlow switches. Our mathematical model is based on the idle timeout of a flow entry, cache size of a flow table, number of hosts, and pattern of incoming flows.

A. TRAFFIC GENERATION

According to Mori *et al.* [18], realistic traffic patterns based on the characteristics of internet traffic variability have an occurrence rate of 4.65% and occupation rate of 41.30% for elephant flows. To apply these two rates to our probability model, we first divide the flow patterns into elephant flows and mice flows according to the number of incoming packets per second (pps). We then define the rates of occurrence and occupation for the elephant flows as probability variables with values between 0 and 1.

$$OCCURRENCE_{ELEPHANT} = 0.0465 \tag{1}$$

$$OCCUPATION_{ELEPHANT} = 0.4130 \tag{2}$$

$$OCCURRENCE_{MICE} = 1 - OCCURRENCE_{ELEPHANT} \tag{3}$$

$$OCCUPATION_{MICE} = 1 - OCCUPATION_{ELEPHANT} \tag{4}$$

To obtain the ratio of packets per second between elephant and mice flows in a simple manner, we assume that all packets in the flows have the same size. Each flow size can then be defined by dividing the occupation rate by the occurrence rate. We can also consider the ratio of flows per second (fps) to be the same as the ratio of packets per second.

$$FLOWSIZE_{ELEPHANT} = \frac{OCCUPATION_{ELEPHANT}}{OCCURRENCE_{ELEPHANT}} \tag{5}$$

$$FLOWSIZE_{MICE} = \frac{OCCUPATION_{MICE}}{OCCURRENCE_{MICE}} \tag{6}$$

$$FLOWSCALE = \frac{FLOWSIZE_{ELEPHANT}}{FLOWSIZE_{MICE}} = 14.42 \tag{7}$$

Using (1) and (2), we calculate that the number of incoming packets per unit time in elephant flows is 14.42 times more than that in mice flows, which is expressed by (7).

For a simple approach to define the probability of flow occurrence on a host, we assume that every host has the same chance to generate an elephant flow or a mice flow every second, which depends on the mean number of incoming packets per second. Additionally, the total number of packets increases in proportion to the number of hosts and the weighted probability of the flow occurrence is defined by applying the ratio of packets per second between the elephant and mice flows.

$$\begin{aligned} OCCURRENCE_{WEIGHTED} &= OCCURRENCE_{ELEPHANT} * FLOWSCALE \\ &\quad + OCCURRENCE_{MICE} \end{aligned} \tag{8}$$

$$P = \frac{PACKET_{MEAN}}{PACKET_{TOTAL} * OCCURRENCE_{WEIGHTED}} \tag{9}$$

Fig. 5 displays an example of traffic generation based on the equations (1) through (9), where the mean number of incoming packets per second and number of hosts are set to 25 and 20, respectively. In our model, every host has the same probability (9) to generate a flow every second, and we can generate various traffic patterns by changing the parameters

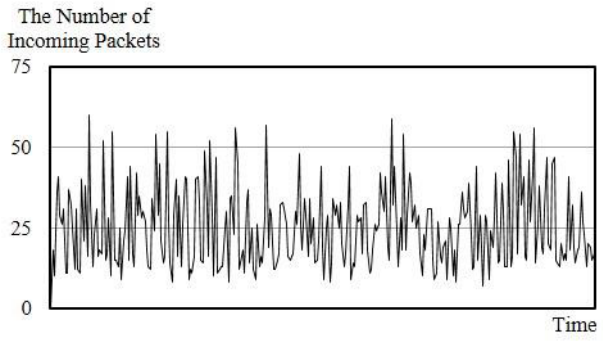

FIGURE 5. Traffic generation example based on the introduced mathematical model. In this model, the mean number of incoming packets per second was set to 25, the number of hosts was set to 20, and the rates of occurrence and occupation for elephant flows were set to 4.65% and 41.30%, respectively.

of (1) and (2), as well as the mean number of incoming packets per second and number of hosts.

B. TABLE-MISS RATE CALCULATION

To predict the table-miss rate of an OpenFlow switch, we first define the probabilities of flow entry existence in the existing and proposed schemes.

$$EXISTENCE_{EXISTING} = \sum_{i=0}^{TIMEOUT-1} P(1-P)^i \quad (10)$$

$$EXPIRATION_j = \sum_{l=TIMEOUT}^{j} P(1-P)^l \quad (11)$$

In the existing scheme, a flow entry exists only before its idle timeout expires and the probability of flow entry existence can be defined as (10). However, the probability of flow entry expiration at time j can be defined as (11), which is used for the calculation of the probability of inactive flow entry existence in the proposed scheme.

We consider the cache size of a flow table because expired flow entries can only be cached as inactive flow entries within the cache size.

$$EXISTENCE_{INACTIVE_j} = \sum_{k=0}^{CACHESIZE-1} \left[\binom{PACKET_{TOTAL}-1}{k} (EXPIRATION_j)^k \left(1-EXPIRATION_j\right)^{(PACKET_{TOTAL}-1)-k} \right] \quad (12)$$

Based on the probability mass function of a binomial distribution, we define the probability of inactive flow entry existence at time j, where the number of inactive flow entries is less than the cache size. This can be expressed as (12) with the assumption that the flow table has a static cache size. To apply the LRU policy to the probability model, we stipulate that only inactive flow entries whose ages are less than the cache size are considered for calculation. Additionally, an expired flow entry should be cached as an inactive flow entry only if the total number of cached flow entries is less than the cache size.

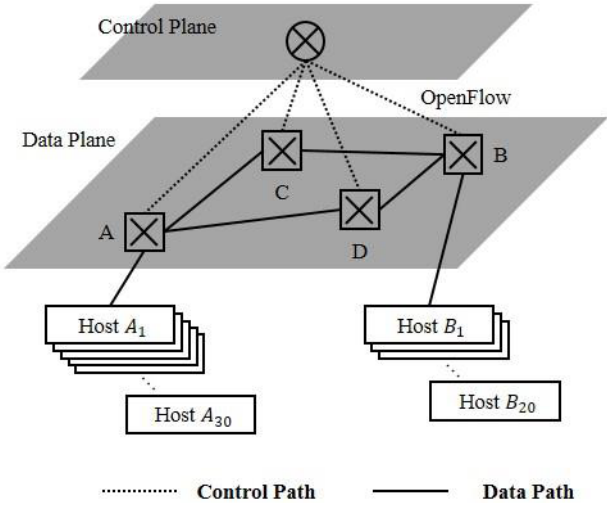

FIGURE 6. Network topology for experiments on the proposed flow table management scheme with the LRU-based caching algorithm for SDN. By using Floodlight, Open vSwitch, and Mininet, we constructed a simple topology consisting of 1 OpenFlow controller, 4 OpenFlow switches, 30 hosts connected to switch A, and 20 hosts connected to switch B.

$$EXISTENCE_{CACHED} = \sum_{j=TIMEOUT}^{\infty} \left[P(1-P)^j \left(EXISTENCE_{INACTIVE_j} \right) \right] \quad (13)$$

Using (12) with the above two considerations, we can obtain the probability of inactive flow entry existence for applying the proposed LRU-based caching algorithm, which is expressed as (13).

$$EXISTENCE_{PROPOSED} = EXISTENCE_{EXISTING} + EXISTENCE_{CACHED} \quad (14)$$

The final probability of flow entry existence in the proposed scheme can be defined as (14) and we can verify that the proposed scheme has a higher probability of flow entry existence than the existing scheme. Their performance difference is expressed by (13).

We now predict the table-miss rate of an OpenFlow switch based on (10) and (14).

$$TABLEMISS_{EXISTING} = P\left(1 - EXISTENCE_{EXISTING}\right) \quad (15)$$

$$TABLEMISS_{PROPOSED} = P\left(1 - EXISTENCE_{PROPOSED}\right) \quad (16)$$

When a packet arrives at the switch, a table-miss occurs if there is no matched flow entry in the flow table of the switch. Based on this definition, the table-miss rates of the existing and proposed schemes can be defined as (15) and (16), respectively.

$$E\left(TABLEMISS_{EXISTING}\right) = TABLEMISS_{EXISTING} * PACKET_{TOTAL} \quad (17)$$

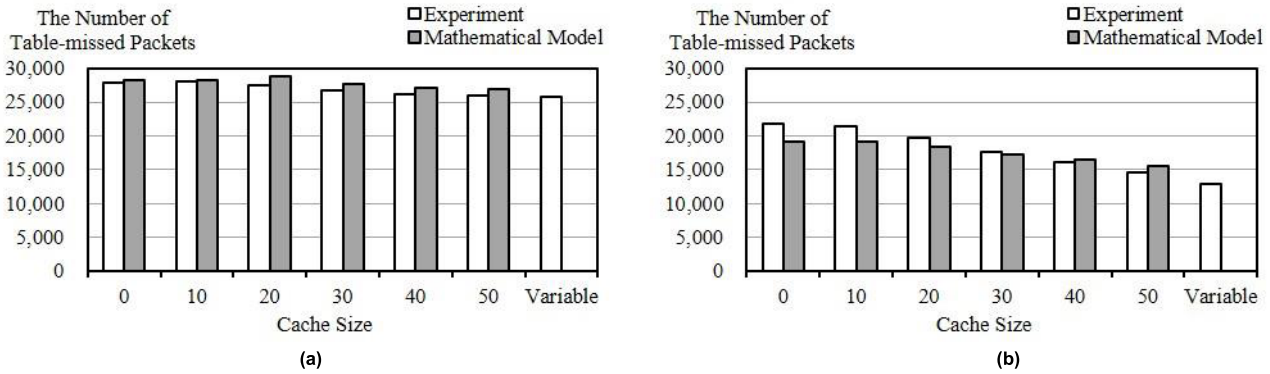

FIGURE 7. Experimental results for changes in the cache sizes of (a) switch A and (b) switch B. OpenFlow switches A and B both achieved higher performance as the cache size increased and achieved the best performance with variable cache size. The mean error rate of our mathematical model is 4.74 %.

$$E\left(TABLEMISS_{PROPOSED}\right) = TABLEMISS_{PROPOSED} * PACKET_{TOTAL} \quad (18)$$

The expectation values of table-miss occurrences per second in the existing and proposed schemes can be defined as (17) and (18), respectively. According to equations (15) to (18), the table-miss occurrence rate in the proposed scheme is lower than that in the existing scheme. We further verify our mathematical model through experiments in Section VI.

VI. EXPERIMENTS

We generated realistic traffic patterns based on the mathematical model described in Section V-A and constructed the simple OpenFlow network topology shown in Fig. 6. In our experiments, when a host connected to switch A communicated with another host connected to switch B, their packets passed through switch C as a default path and switch D was considered as an alternative. Additionally, each experiment ran for 15 minutes with a cold start where the initial state of the flow tables was empty and the flow table capacity to store flow entries was set to 300.

We conducted experiments on the proposed flow table management scheme by applying the LRU-based caching algorithm with a simple modification of the OpenFlow-enabled virtual switch Open vSwitch version 2.7.0 [19]. The OpenFlow controller Floodlight version 1.2 [20] and instant virtual network emulator Mininet version 2.3.0 [21] were also used for our experiments. They ran on the operating system Ubuntu version 16.04 [22].

The experimental results are presented in terms of the number of table-missed packets with changing cache size, number of hosts, and number of incoming packets. All experiments were repeated 10 times and the results of each experiment are expressed as a mean value. To verify the mathematical model from Section V-B, we also compared the experimental results with the expectation values of table-miss occurrence from the probability model.

A. RESULTS FOR CHANGES IN CACHE SIZE

To identify the effectiveness of LRU-based caching in a flow table, we compared the numbers of table-missed packets with various cache sizes. For this purpose, two experiments were conducted: one using only switch A with its 30 hosts and the other using only switch B with its 20 hosts. In both cases, the mean number of incoming packets per second was set to 30 and cache sizes ranged from 0 to 50. According to the vacancy in the flow table, variable cache size was also considered. Fig. 7 indicates that the number of table-missed packets decreases as the cache size increases and that the OpenFlow switch achieves the highest performance with a variable cache size. For LRU-based caching, switches A and B showed 7.48% and 40.73% higher performance than the existing scheme, respectively. We also identified that the results of the experiment and the mathematical model are very similar with an acceptable error rate.

B. RESULTS FOR CHANGES IN THE NUMBER OF HOSTS

According to the results presented in Section VI-A, switch B achieved a higher performance improvement rate than switch A. To identify the impact of the number of hosts on performance improvement, we compared the numbers of table-missed packets with various numbers of hosts. For this purpose, we set the number of hosts independently in each experiment by disabling the constant number of hosts for switches A and B. The cache size in the proposed scheme was set to be variable and the mean number of incoming packets per second was set to 30. Fig. 8 indicates that the performance improvement rate exponentially increases as the number of hosts decreases and that the OpenFlow switch achieves the highest performance when considering both the LRU-based caching algorithm and vacancy in all cases. Additionally, switch B achieved a higher performance improvement rate than switch A. The reason for this is that the total required flow entry for the switch exponentially decreases as the number of hosts decreases, which increases the probability that incoming packets will match the inactive flow entries in a flow table.

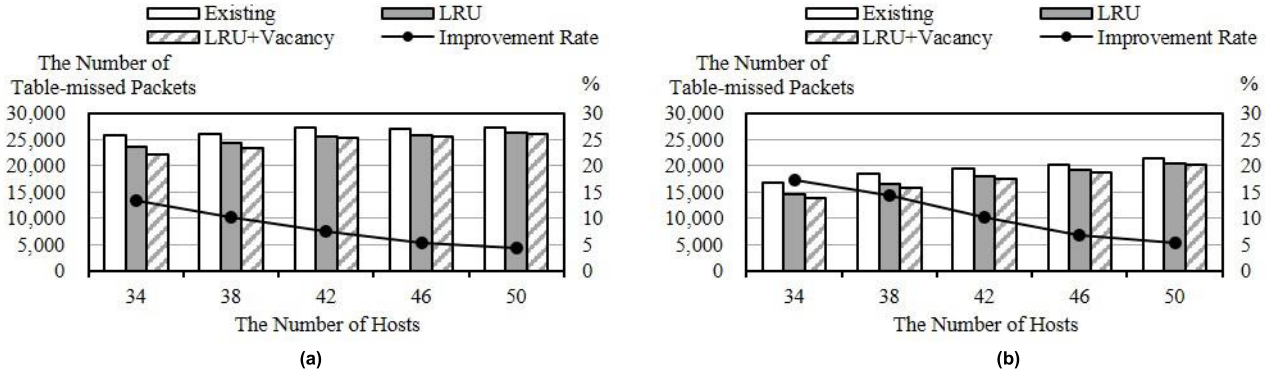

FIGURE 8. Experimental results for changes in the number of hosts on (a) switch A and (b) switch B. To independently set the number of hosts for each experiment, we disabled the constant number of hosts on switches A and B in each experiment. OpenFlow switches A and B both achieved higher performance improvement rates as the number of hosts decreased and achieved the best performance when considering both the LRU-based caching algorithm and vacancy.

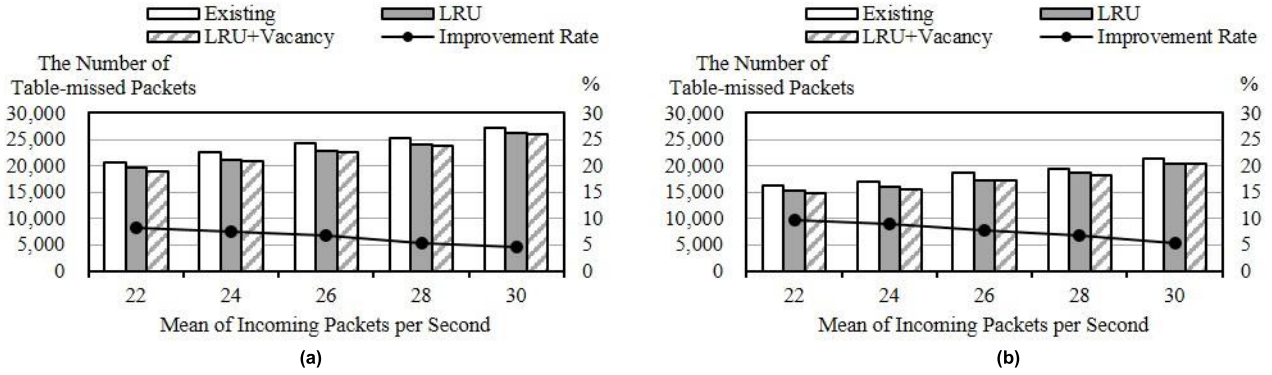

FIGURE 9. Experimental results for changes in the number of incoming packets for (a) switch A and (b) switch B. OpenFlow switches A and B both achieved higher performance improvement rates as the number of incoming packets per second decreased.

C. RESULTS FOR CHANGES IN THE NUMBER OF INCOMING PACKETS

As another factor affecting performance improvement, we compared the number of table-missed packets with various numbers of incoming packets. For this purpose, we changed the mean number of incoming packets per second in each experiment by adjusting the probability of generating packets for all hosts. The cache size in the proposed scheme was set to be variable. Fig. 9 indicates that the performance improvement rate linearly increases as the mean number of incoming packets per second decreases and that the OpenFlow switch achieves the highest performance when considering both the LRU-based caching algorithm and vacancy in all cases. The reason for this is that the period for flow entry replacements becomes linearly longer as the mean number of incoming packets per second decreases, which allows the switch to keep more inactive flow entries in the flow table.

VII. CONCLUSION

Commercial OpenFlow switches have one or more flow tables with multiple flow entries that are managed based only on their timeout parameters. The flow entries are expired and deleted by the switch regardless of their reusability and the absence of flow entries can cause frequent table-misses in the future. To handle a table-missed packet, an OpenFlow switch must perform additional interactions with an OpenFlow controller to insert new flow entries, which results in additional processing time and communication overhead. Previous studies aimed at overcoming this limitation required sophisticated training sets or large changes to SDN architectures. We proposed a simple flow table management scheme with an LRU-based caching algorithm to keep flow entries in a flow table as long as possible. For this purpose, we considered a variable cache size that is continually adjusted by the switch and an alternative packet forwarding scheme based on the vacancies in each flow table. Through an experimental evaluation of the proposed scheme, we demonstrated significant performance enhancement in terms of the number of table-missed packets with changes in the cache size, number of hosts, and number of incoming packets. According to the results of our experiments, the performance improvement rate increases exponentially with a decreasing number of hosts and increases linearly with a decreasing number of incoming packets. Using a mathematical model that we defined to predict the table-miss rate of an OpenFlow switch, we also verified the effectiveness of

the LRU-based caching algorithm for managing expired flow entries in the flow tables of switches.

REFERENCES

[1] N. McKeown *et al.*, "Software-defined networking: The new norm for networks," Open Netw. Found., Palo Alto, CA, USA, White Paper, 2012.

[2] N. McKeown *et al.*, "OpenFlow switch specification version 1.5.1," Open Netw. Found., Palo Alto, CA, USA, Tech. Rep. TS-025, Mar. 2015.

[3] M. P. Fernandez, "Comparing OpenFlow controller paradigms scalability: Reactive and proactive," in *Proc. IEEE 27th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Barcelona, Spain, Mar. 2013, pp. 1009–1016, doi: 10.1109/AINA.2013.113.

[4] E.-D. Kim, Y. Choi, S.-I. Lee, M.-K. Shin, and H.-J. Kim, "Flow table management scheme applying an LRU caching algorithm," in *Proc. IEEE 5th Int. Conf. Inf. Commun. Technol. Convergence (ICTC)*, Busan, Korea, Oct. 2014, pp. 335–340, doi: 10.1109/ICTC.2014.6983149.

[5] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules placement problem in OpenFlow networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1273–1286, 2nd Quart., 2015, doi: 10.1109/COMST.2015.2506984.

[6] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Oct. 2013, doi: 10.1145/2534169.2486012.

[7] M. Kobayashi *et al.*, "Maturing of OpenFlow and software-defined networking through deployments," *Int. J. Comput. Telecommun. Netw.*, vol. 61, no. 11, pp. 151–175, Mar. 2014, doi: 10.1016/j.bjf.2013.10.011.

[8] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," in *Proc. IEEE*, New York, NY, USA, Mar. 2014, pp. 14–76.

[9] N. McKeown *et al.*, "OpenFlow switch specification version 1.0.0," Open Netw. Found., Palo Alto, CA, USA, Tech. Rep. TS-001, Dec. 2009.

[10] N. McKeown *et al.*, "OpenFlow switch specification version 1.4.0," Open Netw. Found., Palo Alto, CA, USA, Tech. Rep. TS-012, Oct. 2013.

[11] A. R. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, Toronto, ON, Canada, 2011, pp. 254–265, doi: 10.1145/2018436.2018466.

[12] B.-S. Lee, R. Kanagavelu, and K. M. M. Aung, "An efficient flow cache algorithm with improved fairness in software-defined data center networks," in *Proc. IEEE 2nd Int. Conf. Cloud Netw. (CloudNet)*, San Francisco, CA, USA, Nov. 2013, pp. 18–24, doi: 10.1109/CloudNet.2013.6710553.

[13] M. Yu *et al.*, "Scalable flow-based networking with DIFANE," in *Proc. ACM SIGCOMM Conf.*, New Delhi, India, 2010, pp. 351–362, doi: 10.1145/1851182.1851224.

[14] H. Zhu, H. Fan, X. Luo, and Y. Jin, "Intelligent timeout master: Dynamic timeout for SDN-based data centers," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, Ottawa, ON, Canada, May 2015, pp. 734–737, doi: 10.1109/INM.2015.7140363.

[15] T. Kim *et al.*, "A dynamic timeout control algorithm in software defined networks," *Int. J. Future Comput. Commun.*, vol. 3, no. 5, pp. 331–336, Oct. 2014, doi: 10.7763/IJFCC.2014.V3.321.

[16] A. Zarek, "OpenFlow timeouts demystified," Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2012.

[17] T. H. Cormen *et al.*, "Introduction to algorithms," in *Computer Science*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001, pp. 595–601.

[18] T. Mori, R. Kawahara, S. Naito, and S. Goto, "On the characteristics of Internet traffic variability: spikes and elephants," in *Proc. Int. Symp. Appl. Internet (SAINT)*, Tokyo, Japan, 2004, pp. 99–106, doi: 10.1109/SAINT.2004.1266104.

[19] *Open vSwitch: A Linux Foundation Collaborative Projects*. Accessed: Mar. 2017. [Online]. Available: http://openvswitch.org/

[20] *Project Floodlight: Open Source Software for Building Software-Defined Networks*. Accessed: Mar. 2017. [Online]. Available: http://www.projectfloodlight.org/

[21] *MiniNet: An instant Virtual Network on Your Laptop (or Other PC)*. Accessed: Mar. 2017. [Online]. Available: http://mininet.org/

[22] *Ubuntu: An Open Source Software Operating System That Runs From the Desktop, to the Cloud, to all Your Internet Connected Things*. Accessed: Mar. 2017. [Online]. Available: https://www.ubuntu.com/

EUN-DO KIM was born in Seoul, South Korea, in 1987. He received the B.S. degree in applied physics from Hanyang University, Ansan, South Korea, in 2012. He is currently pursuing the integrated M.S. and Ph.D. degrees in information and communication technology with the University of Science and Technology (UST), Daejeon, South Korea.

Since 2012, he has been an UST Graduate Student with the Protocol Engineering Center, Electronics and Telecommunications Research Institute, Daejeon. His research interest includes the performance enhancement of OpenFlow network protocol used for SDN.

YUNCHUL CHOI received the B.S. degree in electrical and computer engineering and the M.S. degree in computer network from Chungnam National University, Daejeon, South Korea, in 2007 and 2010, respectively, where he is currently pursuing the Ph.D. degree in computer communication.

From 2010 to 2012, he was an Assistant Researcher with the Next Communication Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Daejeon. In 2012, he was a Visiting Researcher with the Ilmenau University of Technology, Ilmenau, Germany, and an Internship with Deutsche Telekom, Berlin, Germany. Since 2012, he has been a Researcher with the Protocol Engineering Center, ETRI. His current research interest includes the testbed construction for future Internet.

SEUNG-IK LEE received the B.S. degree in computer science and engineering from Handong University, Pohang, South Korea, in 2000, and the M.S. and Ph.D. degrees in computer science from KAIST, Daejeon, South Korea, in 2002 and 2009, respectively.

He joined the Protocol Engineering Center, Electronics and Telecommunications Research Institute, Daejeon, in 2009, and has participated in the standardizations for IPTV, multicast, NGSON, SDN, NFV, SFC, and 5G in ITU-T, JTC1/SC6, IEEE, ETSI, IETF, and 3GPP. His current research interests include 5G core network technologies.

HYOUNG JUN KIM received the B.S. and M.S. degrees in computer science from Kwangwoon University, Seoul, South Korea, in 1986 and 1988, respectively, and the Ph.D. degree in computer science from Chungnam National University, Daejeon, South Korea, in 2007.

He joined the Protocol Engineering Center, Electronics and Telecommunications Research Institute, Daejeon, in 1988, where he has been a Vice President since 2013. He also has been a Professor with the Information and Communication Technology (ICT), University of Science and Technology, Daejeon, since 2008. He was the Visiting Scholars with the University of Brussels from 1994 to 1997 and with the University of Virginia from 2007 to 2008. In addition, he has been currently serving as a Vice Chair of ITU-T SG20, a Chair of WP1/20, and a Co-Convener of JCA-IoT/SC&C (Joint Coordination Activity of Internet of Things and Smart Cities & Communities). He also has experiences served as a Vice Chair of ITU-T SG13, a Chair of WP3/13, a Rapporteur of Q25/16, a Vice Chair of FG on M2M Service Layers, a Chair of WP2 of the FG in ITU-T SG11, a Vice Chair of FG on Future Networks in ITU-T SG13, and so on. Regarding his research and academic achievements, he has contributed over 450 proposals for standards and authored or co-authored over 150 papers in academic journals and conferences as well as more than 100 patents and 20 official technologies transfers to domestic companies. His research interests include ICT standardization, Internet of Things (IoT), and future networks.

Dr. Kim's awards and honors include the National President's Citation in 2003 and 2009, respectively, as well as many Certificates of Appreciation from international standard-related organizations, including ITU-T. In particular, he was a recipient of the National President's Award at the World Standards Day 2009 in Korea for honoring his efforts on international standardization.