J Inf Process Syst, Vol.14, No.1, pp.228~238, February 2018
https://doi.org/10.3745/JIPS.01.0023

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Efficient Flow Table Management Scheme in SDN-Based Cloud Computing Networks

Nambong Ha* and Namgi Kim*

Abstract

With the rapid advancement of Internet services, there has been a dramatic increase in services that dynamically provide Internet resources on demand, such as cloud computing. In a cloud computing service, because the number of users in the cloud is changing dynamically, it is more efficient to utilize a flexible network technology such as software-defined networking (SDN). However, to efficiently support the SDN-based cloud computing service with limited resources, it is important to effectively manage the flow table at the SDN switch. Therefore, in this paper, a new flow management scheme is proposed that is able to, through efficient management, speed up the flow-entry search speed and simultaneously maximize the number of flow entries. The proposed scheme maximizes the capacity of the flow table by efficiently storing flow entry information while quickly executing the operation of flow-entry search by employing a hash index. In this paper, the proposed scheme is implemented by modifying the actual software SDN switch and then, its performance is analyzed. The results of the analysis show that the proposed scheme, by managing the flow tables efficiently, can support more flow entries.

Keywords

Cloud Computing Network, Cloud Service, Flow Table, SDN

1. Introduction

Conventional computer networks aim at forwarding a packet from the source host to the destination host as quickly as possible based on the destination IP address. Consequently, there is an advantage in that this process is simple and easy to implement. However, it is complicated and inconvenient to change the functions of a conventional network architecture once it has been deployed [1]. Thus, when some problems or changes occur in the network, it is difficult to cope with them. Additionally, because the companies that provide switches and routers have their own solutions to such problems along with different managing software, compatibility problems exist between various companies. Moreover, with the birth of very flexible service models where the customers pay only for their usage, such as in cloud computing services, a means of operating networks flexibly is required. Among the proposed technologies to address this issue, software-defined networking (SDN) is one that has been practically implemented [2-4]. SDN is a technology that dynamically enables a more flexible network configuration in the conventional network by allowing centralized processing through the decoupling

Manuscript received March 14, 2017; first revision April 19, 2017; accepted May 6, 2017.
Corresponding Author: Namgi Kim (ngkim@kyonggi.ac.kr)
* Dept. of Computer Science, Kyonggi University, Suwon, Korea ({nbha, ngkim}@kyonggi.ac.kr)

of packet-forwarding and function-control. In the SDN, the control plane fulfills the role of managing information for packet-forwarding, and the data plane is responsible for forwarding the packets according to the information determined by the control plane. Thus, by dividing the control plane and data plane, and by allowing the SDN controller to carry out the centralized processing of packet control, the SDN can easily support a multi-tenant network in a complicated and dynamically changing service, such as a cloud-computing server. Therefore, the application of SDN network technology in cloud computing services is gradually expanding.

To maximize the performance of SDN-based cloud computing networks, effective research is needed in many aspects such as the SDN controller platform, application designs, switching speed, and switch capacity. Research in the field of switching technology that can support more users expeditiously by efficiently utilizing limited resources plays a critical role in the scalability of SDN-based cloud computing services [2,5]. An SDN switch stores packet-forwarding information in the flow table when it obtains the information from a controller. However, owing to the limited capacity of the flow table, the maximum number of users that can be supported by a switch is bounded by the number of flow table entries and the flow processing speed. Therefore, if faster support of a greater number of flow entries is possible by the efficient utilization of the switch's flow table resources, the switching capacity will increase, which, in turn, will result in more efficient cloud computing services. Accordingly, this paper proposes a scheme that increases the number of flow entries while maintaining the speed of flow entry searching via the efficient configuration and management of the flow table.

This paper is organized as follows. Section 2 gives a basic background of the present research and examines the flow table problems that could occur along with the search speed. It also introduces related research and analyzes problems that may arise therein. Section 3 introduces and explains the scheme that is proposed in this paper. Section 4 explains the method to modify the existing OpenFlow switch software and to implement the proposed technique. An experimental environment is established by using the implemented software, and the results of the experiment are analyzed. Finally, Section 5 summarizes the research with conclusions and future research.

2. Background Knowledge and Related Works

There were many suggestions for the introduction of a programmable network to solve the problems of conventional networks that simply forward packets based on the destination IP address. Through various studies, there were several suggestions on new network paradigms [2]. Among the diverse paradigms proposed, SDN is the most widely studied and the practically used technology today.

Since the role of the control plane and data plane in the existing network is not divided, each switch and each router is responsible for all the roles. Thus, when any network problem arises, it is difficult to respond properly. To solve this problem, SDN assigns a unique role for each control plane and data plane. Specifically, the control plane controls the network and the data plane forwards the packet according to the rules determined by the control plane [6]. According to the application that exists in the control plane, the function of the switches in the SDN can be changed, and this enables the provision of various services. Load balancing and service chaining are representational SDN-based services, and the power saving from SDN-based load balancing at Google's data center is an example of this application in the real world.

The SDN operates by exchanging messages between the control plane and data plane. The standard protocol for these messages is called the OpenFlow protocol [7,8]. This protocol is open to the public via open source, and a majority of SDNs use this protocol to exchange messages. Packet-in, Flow-mod, and Packet-out are representational OpenFlow protocol messages. Packet-in is the message that the switch delivers to the controller with packet information when the packet arrives first at the switch. When receiving a Packet-in message, the controller creates the rule for the corresponding packet and sends its information with a Flow-mod message to the switch. Also, it forwards the Packet-out message to the switch. When the switch receives the Flow-mod message, it analyzes the contents of the message and then stores the flow information in the flow table, and proceeds with packet processing. The OpenFlow protocol versions starting with version 1.0 and the current version 1.5 are available now.

2.1 Flow Table Architecture

SDN divides the network organization into the control plane and data plane. The control plane has a set of rules for processing the packet, and when those packets enter the data plane, they are processed according to the pertinent rule. The rules are stored and managed in the flow table inside the switch. However, as the network performance depends on how well the flow information is stored in the flow table, many techniques for defining and managing the flow information in the flow table are being developed. Flow information is stored and managed by the flow entry unit in the flow table. Fig. 1 illustrates the flow table and the flow entry fields stored inside.

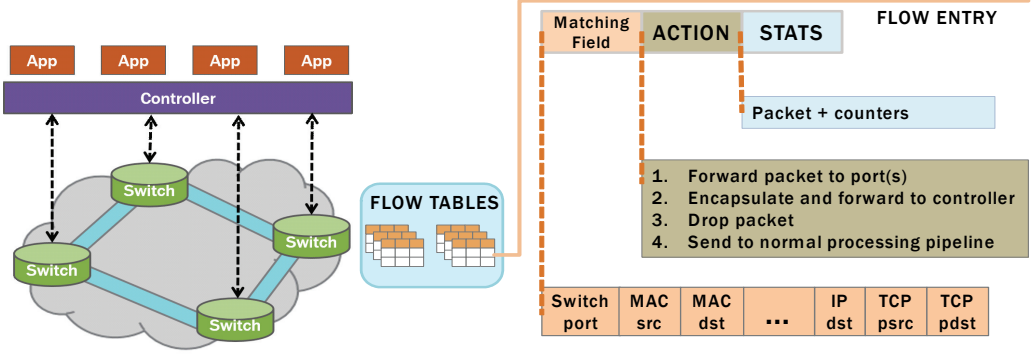

Fig. 1. Flow table and flow entry fields.

Flow entry constitutes Matching field, Action field, and Stats field. The Matching field stores packet information, while the Action field contains the information on what actions to take when the flow entry matches. The Stats field stores information about the number of times the packet is matched.

Memory architecture, which implements a flow table, affects the performance of SDN-based cloud computing networks. The SDN switch implements a flow table using various memory architectures, such as SRAM, CAM, and TCAM [9-11].

The methods and speed of searching for the flow entry in a flow table change according to the characteristics of each memory architecture. The fastest memory architecture is TCAM, which searches the entire flow table in one cycle. However, TCAM has the shortcoming of being too costly for its capacity. Thus, as the demand for its application has increased in many areas, the utilization of a cost-

effective SRAM to implement the flow table can be found more often.

When using the SRAM to implement a flow table, the implementation can be generally categorized into the hash-based flow table and the wildcard-based flow table. The hash-based flow table and wildcard-based flow table, which are also provided by the OpenFlow switch, have their advantages and disadvantages. The hash-based flow table enters flow information as the input of a hash function, and then uses the hash value to store the packet information in a flow table. Since, the hash-based flow table matches the packet entry (which is registered in the flow table) by using the hash function; it has an advantage of a faster search speed. However, even if a single bit from the packet information differs, it has to be stored in a separate flow entry. Thus, the hash-based flow table requires greater memory capacity. For example, consider the case of a packet having W as a destination host and 15 as a destination port, and a packet with W as a destination host and 16 as a destination port. In this case, even if they are outputted to the same x interface, the flow information of these two packets needs to be stored as two different flow entries in the hash-based flow table. Fig. 2 illustrates an example of the composition of a hash-based flow table.

Hash value	Flow identifying fields				Destination
	VLAN	…	SRC port	DST port	
0xAAA0	A	…	a	i	W
0xAABB	B	…	b	j	X
0xCC77	C	…	c	k	Y
0xCCAB	D	…	d	l	Z
…	…	…		…	…

Fig. 2. Example of a hash-based flow table.

The wildcard-based flow table stores flow entries in a flow table, using wildcards. Since it uses wildcards to store flow entries, it is not necessary for the wildcard-based flow table to create a separate flow entry for the packet that contains different packet information. For example, when packets have W as their destination host, and all of them are output to interface x, the corresponding information can be stored as one flow entry in a wildcard-based flow table. Consequently, the wildcard-based flow table can save on memory costs. However, because it has to search the flow entry for the corresponding packet in the flow table, in the worst case, it needs to search the entire flow entry. This is its shortcoming. Thus, as the amount of flow entries increases, the searching speed of a wildcard-based flow table has a tendency to increase linearly. Fig. 3 illustrates an example of a wildcard-based flow table.

Flow identifying fields				Destination
VLAN	…	SRC port	DST port	
*	…	a	h	W
B	…	*	i	X
C	…	*	*	Y
*	…	d	j	Z
…	…	…	…	…

Fig. 3. Example of a wildcard-based flow table.

2.2 Hybrid Flow Table Architecture

In SDN, the flow table architecture has different characteristics according to its type. The hash-based flow table requires large memory capacity, but otherwise, its flow wildcard scheme, after implementing entry searching speed is fast. The wildcard-based flow table utilizes wildcards to store large amounts of flow information in a small memory, but flow-entry search speed is slow. To overcome these differences and to utilize the advantages from both architectures, a hybrid flow-table architecture was proposed [12].

A hybrid flow-table architecture adopts both the hash-based flow table and wildcard-based flow table, and represents a hash-assisted wildcard scheme. The hash-assisted wildcard scheme first stores the information in a wildcard-based flow table when the flow information enters. When the packet that needs to be forwarded is in the switch, the corresponding flow information is first searched in a hash-based flow table, and only when it is absent, the corresponding flow information is searched in the wildcard-based flow table. At this time, if the flow information of the packet matches, then this information is also registered in the hash-based flow table. Thus, when packets with the same flow information enter afterward, the matches occur due to the flow information registered in the hash-based flow table, making the flow-entry search of the wildcard-based flow table unnecessary. Thus, it facilitates the fast forwarding of a packet.

The hash-assisted wildcard scheme that has been explained above uses both hash-based flow table and wildcard-based flow table. Thus, this enables better performance, when compared to the flow table architecture that uses only one of them. However, the hash-based flow table has the following shortcomings: first, a decreased performance as the amount of flow information increases; second, it has to copy the flow information from the wildcard-based flow table into it, when the packets match at the wildcard-based flow table of the hash-assisted wildcard scheme. The task of copying the flow information from the wildcard-based flow table to the hash-based flow table, in case of frequent occurrence, can create too much overhead. In addition, the hash-assisted wildcard scheme still must implement the hash-based flow table, which occupies large memory; there is a restriction on the amount of flow entries. When the utilization rate of the hash-based flow table increases, it can raise the collision rate of hash function values, and eventually, this decreases the search speed of a hash-based flow table [13]. To resolve such issues, in this paper, a new flow-table management scheme, called hash-indexed wildcard scheme, has been proposed. This proposed scheme enables a faster search of the flow information in a wildcard-based flow table using hash function. Simultaneously, it reduces the memory usage of the hash-based flow table by storing only the flow entry index of the wildcard-flow table that corresponds to the matched flow.

3. Proposed Scheme

Since the degradation of performance always occurs in a hash-based flow table, as the hash table's usage rate increases, it is necessary to secure the maximum number of hash entries. However, when the flow matching occurs in a wildcard-based flow table, in the case of the hash-assisted wildcard scheme, the flow information that pertains to the matching flow is stored in the hash-based flow table. Thus, the amount of memory usage of the hash-based flow table is very large. To overcome this issue, and to decrease the memory requirement of the hash-based flow table, the proposed scheme does not store the entire flow matching information in a hash-based flow table. Instead, it stores only the location of the

flow entry that pertains to the flow in the wildcard-based table. Hence, the proposed scheme deletes the matching field from the existing hash-based flow table and keeps only the index value of the matched flow of the wildcard-based flow table. Therefore, it is able to secure more hash entries. Fig. 4 illustrates the proposed scheme.

The proposed hash-indexed wildcard scheme calculates a hash value that corresponds to the packet via a hash function, when the packet enters the switch. By using the computed hash value, the proposed scheme searches the hash-based flow table. If the index information in the entry that matches the hash value in the hash-based flow table exists, it extracts flow information from the flow entry of the wildcard-based flow table that pertains to the index, and matches the information with the inputted packet. If the pertaining flow information is the right one for the inputted packet, then it uses this information to process the packet. If there is a mismatch between the inputted packet and the extracted flow information from the wildcard-based flow table, then it searches for the flow information that corresponds to the inputted packet via the linear-searching of the wildcard-based flow table. Even when the flow information that pertains to the inputted packet is absent from the hash-based flow table, it carries out the linear search operation on the wildcard-based flow table. During the linear searching of the wildcard-based flow table, if the flow information that corresponds to the inputted packet is found, then it inserts the index information about the flow entry (which contains this flow information) into the hash-based flow table. Through this process, afterward, the inputted packets of the same flow will be matched in the hash-based flow table, thus allowing the pertinent flow information to be matched quickly in a wildcard-based flow table.

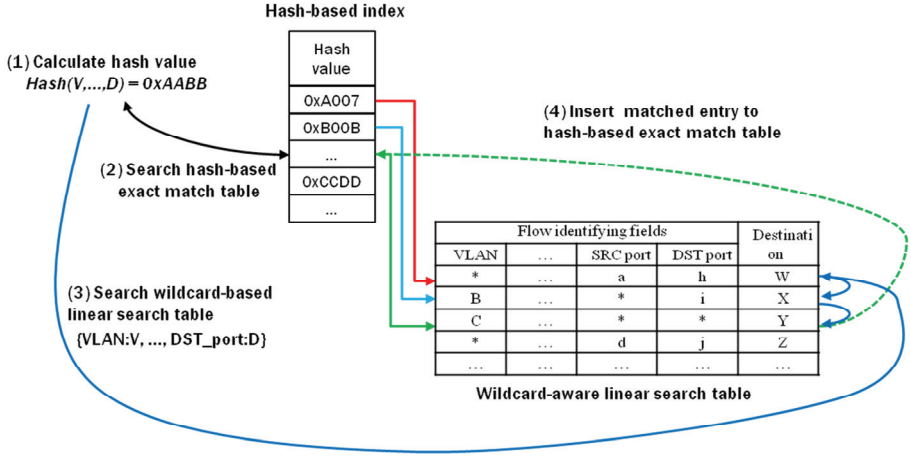

Fig. 4. The hash-indexed wildcard scheme.

4. Performance Evaluation

4.1 Implementation

The switches that support SDN can be largely grouped into hardware SDN switches and software SDN switches. Hardware SDN switches are being produced by many corporations such as HP, Dell, and Piolink. For the software SDN switch, there are OpenVSwitch, OpenFlow switch, Indigo, and others.

Unlike the hardware-based SDN switch, the software-based SDN switch does not require the direct purchase of its tools for installation on the server. This is its advantage. In addition, it is possible to modify the internal source of the switch to add additional functions. Anyone can manipulate the software-based SDN switch because it is an open-source software.

The OpenFlow switch, developed at Stanford University, is a software-based SDN switch that is easy to modify, and has been widely used in many studies. OpenFlow switches implement both hash-based flow tables and wildcard-based flow tables. The hash-based flow table of the OpenFlow switch uses CRC32 as a hash function. The wildcard-based flow table is composed of flow entries with wildcards and search its entries in a linear way. In this paper, we use OpenFlow switch version 1.0 and modifies it to implement and compare the hash-assisted wildcard scheme and the hash-indexed wildcard scheme.

In the OpenFlow switch, the size of the hash-based flow table is fixed as a unit of 2^n. The index value is the lower n bits of the value obtained by computing the packet information by hash function. For example, if n is 10, and the value of the hash is 1105482, then the index value will be the lower 10 bits of 1105482, which is 586, and the corresponding flow entry will be stored in index 586. The flow entry in the hash-based flow table is composed of the flow structure defined in lib/flow.h, the sw_flow_key structure defined in udatapath/switch-flow.h, and the sw_flow structure defined in udatapath/switch-flow.h. It uses 160 bytes of memory per flow entry. The wildcard-based flow table in an OpenFlow switch is composed of a matching field that stores priority and flow information, and as in the hash-based flow table, uses 160 bytes of memory per flow entry.

Fig. 5 shows the architecture of a hash-assisted wildcard scheme implemented by using the OpenFlow switch. In the hash-assisted wildcard scheme, the flow entry of the hash-based flow table is copied and inserted from the wildcard-based flow entry. The index size of the hash-based flow table is 8 Bytes. If the number of entries of a flow table is 2^n, then the size of the hash-based flow table is 168 bytes × 2^n.

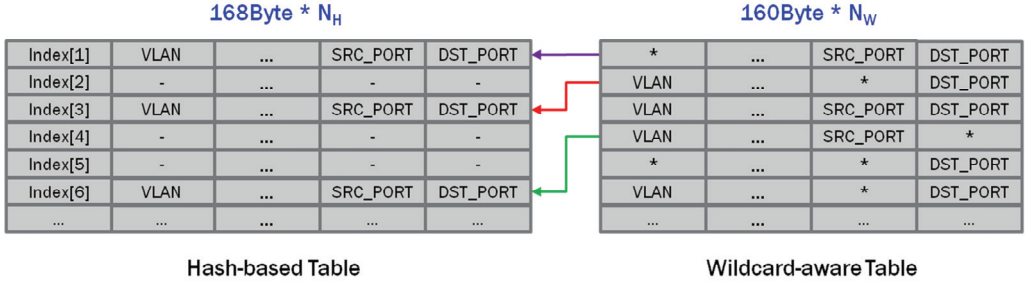

Fig. 5. Implemented architecture of the hash-assisted wildcard scheme.

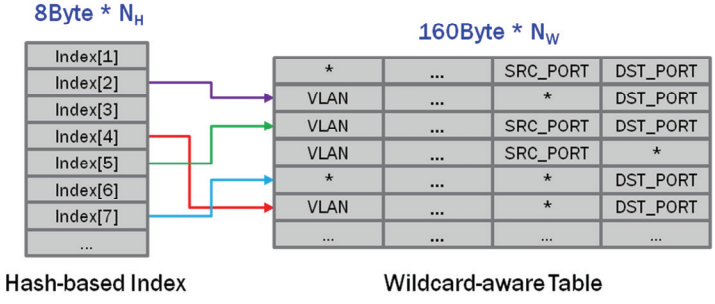

Fig. 6. Implemented architecture of the hash-indexed wildcard scheme.

Fig. 6 illustrates the architecture of the hash-indexed wildcard scheme implemented in the OpenFlow switch. In a Hash-indexed wildcard scheme, only the index information that refers to the flow entry inside a wildcard-based flow table is stored. Thus, in comparison to a hash-assisted wildcard scheme, the proposed scheme can store a greater number of flow entries with the same memory capacity.

4.2 Evaluation

In this paper, an experiment compared the wildcard-only scheme, hash-assisted wildcard scheme, and hash-indexed wildcard scheme to evaluate the performance. For the experiment, a virtual network was generated using Mininet in the Ubuntu 14.04 64-bit, and the source code of OpenFlow switch, version 1.0 was modified to conduct the experiment. Table 1 summarizes the system environment used for the experiment.

Table 1. System environment for the experiments

Parameter	Value
CPU	I5-4460
RAM	4 GB
OS	Ubuntu 14.04 (64 bit)
Virtual network	Mininet
Software SDN switch	OpenFlow switch 1.0

Table 2. Experimental setup

Scheme	Small system	Large system
Wildcard only	1,000 wildcard entries	10,000 wildcard entries
Hash-assisted wildcard	1,024 hash entries + 1,000 wildcard entries	8,192 hash entries + 10,000 wildcard entries
Hash-indexed wildcard	16,384 hash entries + 1,000 wildcard entries	131,072 hash entries + 10,000 wildcard entries

Table 2 summarizes the system environment for the experiment. The experiment was carried out separately in a small system environment and in a large system environment. The small system environment means a system environment with a small system memory capacity, where the number of flow entries in flow tables is set to be small. The large system environment means a system environment with a greater memory capacity, so the number of flow entries is set to the greater amount. In the small system environment, the maximum number of flow entries of the hash-based flow table in the hash-assisted wildcard scheme is set to 1,024, while the maximum number of flow entries of the hash-based flow table in the hash-indexed wildcard scheme is set to 16,834. This is because, when storing the same number of flow entries in a hash-based flow table, the hash-assisted wildcard scheme requires a memory size that is more than 16 times larger than the hash-indexed wildcard scheme. In the large system environment, the maximum number of flow entries in the hash-based flow table is set to 8,192 in the hash-assisted wildcard scheme, and the maximum number of flow entries in the hash-indexed wildcard scheme is set to 131,072 for reasons mentioned above.

Figs. 7 and 8 show the packet-forwarding time according to the number of flows. As one can see from

these figures, when only the wildcard-based flow table is used in the wildcard-only scheme, regardless of the number of flows, it always requires considerably more time. In the case of a hash-assisted wildcard scheme, if the number of flows greater than the number of flow entries that can be stored in the hash-based flow table enters as the switch inputs, whenever the wildcard-based flow table is updated, the flow entry of the hash-based flow table is replaced. Therefore, it shows results similar to those of the wildcard-only scheme. For the proposed scheme in this paper, since the number of flow entries that can be stored in the hash-based flow table is much greater than that in the hash-assisted flow scheme, even if the number of flows increases, its packet forwarding time is much shorter than that of the hash-assisted wildcard scheme.

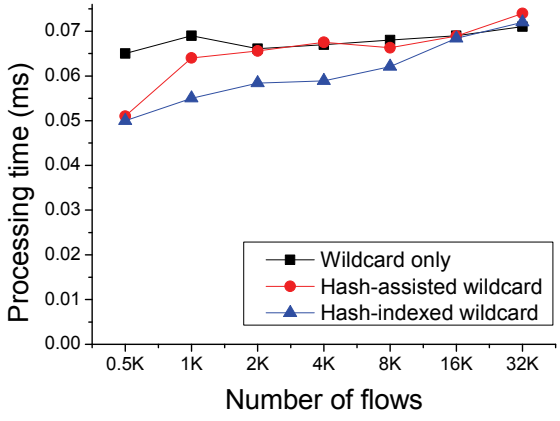

Fig. 7. Packet forwarding time for small system environment.

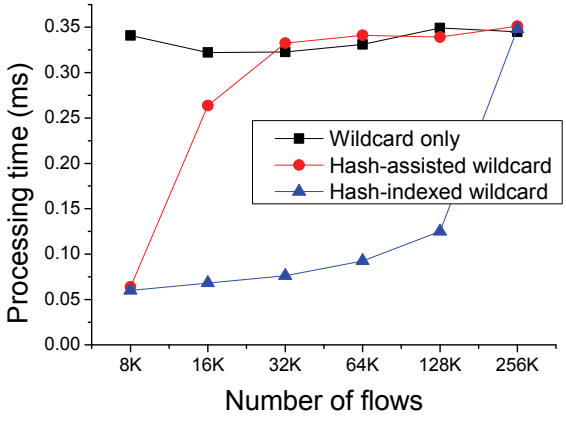

Fig. 8. Packet forwarding time for large system environment.

5. Conclusion

In this paper, a novel flow-table management scheme that can be used in SDN-based network systems, which can efficiently support cloud services, is proposed. In an SDN-based network system, a

flow table can be configured as a hash-based flow table and a wildcard-based flow table. The hash-based flow table of the previously proposed hash-assisted wildcard scheme has the advantage of a faster flow-information search that can shorten the packet forwarding time. But it requires a greater memory capacity because it stores the flow information for every flow entry. Especially with respect to the hash table, when the entry occupation rate is saturated, there is a rapid decline in its search speed. This can reduce its performance. Therefore, it is necessary to secure the maximum number of flow entries in hash-based flow tables. Accordingly, in this paper, a hash-indexed wildcard scheme is proposed. The proposed scheme deletes the flow-matching field stored per flow entry in the hash-based flow tables. It stores only the index information for the matching flow in the wildcard flow table. Consequently, the scheme reduces memory usage so that the number of flow entries of the hash-based flow table can be greater than that with the hash-assisted wildcard scheme.

The results of performance evaluation show that the proposed scheme exhibited a maximum performance increase of 70%, in comparison to the hash-assisted wildcard scheme. Future work will involve transferring the experimental environment implemented in the software-based SDN switch to a hardware-based SDN switch for implementation. Further, the performance of the proposed scheme in a real-time environment will be analyzed.

Acknowledgement

This work was supported by Kyonggi University Research Grant 2016.

References

[1] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, Berkeley, CA, 2009, pp. 335-348.

[2] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," in *Proceedings of the IEEE*, 2015, vol. 103, no. 1, pp. 14-76.

[3] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: decoupling architecture from infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks,* Redmond, WA, 2012, pp. 43-48.

[4] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114-119, 2013.

[5] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136-141, 2013.

[6] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: data plane performance," in *Proceedings of IEEE International Conference on Communications*, Cape Town, South Africa, 2010, pp. 1-5.

[7] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *Proceedings of the 23rd International Teletraffic Congress*, San Francisco, CA, 2011, pp. 1-7.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.

[9] M. Appelman, M. D. Boer, and R. V. D. Pol, "Performance analysis of OpenFlow hardware," University of Amsterdam, *Technical Report*, 2012.

[10] C. H. Hung, C. W. Huang, L. C. Wang, and C. Chen, "Scalable topology-based flow entry management in data center," in *Proceedings of 13th IEEE Annual Consumer Communications & Networking Conference*, Las Vegas, NV, 2016, pp. 85-90.

[11] 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 *Proceedings of IEEE 2nd International Conference on Cloud Networking*, San Francisco, CA, 2013, pp. 18-24.

[12] N. Matsumoto and M. Hayashi, "Performance improvement of flow switching with automatic maintenance of hash table assisted by wildcard flow entries," in *Proceedings of the 10th International Conference on Optical Internet*, Yokohama, Japan, 2012, pp. 12-13.

[13] 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, no. 4, pp. 254-265, 2011.

Nambong Ha https://orcid.org/0000-0001-9028-6775

He received the B.S. degree in Computer Science from the Kyonggi University, Korea, in 2015. He is currently M.S. candidate in Computer Science from Kyonggi University. His research interests include computer vision, software defined network.

Namgi Kim https://orcid.org/0000-0002-0077-6576

He received the B.S. degree in Computer Science from Sogang University, Korea, in 1997, and the M.S. and Ph.D. degrees in Computer Science from KAIST in 2000 and 2005, respectively. From 2005 to 2007, he was a research member of the Samsung Electronics. Since 2007, he has been a faculty of the Kyonggi University. His research interests include sensor system, wireless system, and mobile communication.