A Dynamic Timeout Control Algorithm in Software Defined Networks

Taekhee Kim, Kwonyong Lee, Junhee Lee, Sungyong Park, Young-Hwa Kim, and Byungjoon Lee

***Abstract*—While Software Defined Networking (SDN) has received a considerable amount of attention, improving the scalability of an SDN controller has always been a major concern. One of the main reasons why the controller suffers from this scalability problem is that it is fairly often overwhelmed by a large number of flow setup requests from the SDN switches to the controller. These requests cause decrease in the number of switches that the controller can deal with. Since these requests are usually generated when flows arrive at the switch and their corresponding entries do not exist in the flow table due to eviction, minimizing the number of evictions also reduces the number of requests to the controller. This paper addresses the scalability problem and proposes the algorithm improving the scalability of the SDN controller by dynamically controlling the timeout value of each flow without modifying the switches. In the proposed approach, the controller collects various traffic parameters from the switches and predicts the inter-arrival times of packets in a flow. Based on the information, it dynamically adjusts the timeout value of each flow to reserve spaces in the flow table for newly arrived flows in advance. As a result, this avoids the evictions and reduces the number of flow setup requests to the controller. The benchmarking results show that the proposed algorithm reduces the number of packets to the controller by 9.9 %.**

***Index Terms*—Flow table, open flow, SDN, SDN controller, timeout control.**

I. Introduction

Software Defined Networking (SDN) has emerged as a new computing paradigm to alleviate the complexity of current network protocols and remove the vender-specific problems of network devices. In SDN, control functions are decoupled from data plane and OpenFlow [1] is used as a standard protocol between data plane and control plane. The SDN controller in control plane mainly decides how to deal with the network flows by setting up the flow table, deciding the actions for the flows, or collecting data from the SDN switches. The data plane, on the other hand, just forwards the packets to the SDN controller or to the destination.

With the advances of SDN technology, there have been many studies regarding the scalability, resiliency, consistency, and durability. Among them, improving the scalability of SDN controller has currently been attracting a lot of attentions. The scalability of SDN controller can be improved by two approaches. The first approach uses multiple controllers with shared network view and distributes loads to them such as Onix [2] and ElastiCon [3]. The second approach improves the scalability by decreasing loads on a single controller or by devolving control to the SDN switches such as DevoFlow [4] or DIFANE [5].

This paper focuses on the scalability of a single SDN controller for enterprise networks and proposes a scheme that improves the scalability by reducing the communications between the switches and the controller. While there are several reasons for the switches to send packets to the controller, a large number of packets are generated due to timeouts or premature evictions of unfinished flows [6]. Therefore, controlling the timeout values can help reducing the traffic between the switches and the controller. Zarek [6] explored this issue and proposed a flow table management scheme that combines timeouts with explicit controller eviction messages. However, it used a static timeout value and left dynamic control of timeout values as future studies.

This paper proposes a dynamic timeout control algorithm in the SDN controller. In the proposed approach, the controller collects various traffic parameters from the switches and predicts the inter-arrival times of packets in a flow. Based on this information, our algorithm determines the number of flow entries that are likely to be remained in the table at the next sampling period. It then dynamically adjusts the timeout value of each flow to reserve spaces in the flow table for newly arrived flows in advance. As a result, this avoids the evictions without modifying the switches and reduces the number of flow setup requests to the controller. The experimental results show that our algorithm outperforms the approach with static timeout by about 9.9%.

The rest of our paper is organized as follows. Section II describes the motivation of our algorithm by addressing the challenge of the scalability of a single SDN controller. Section III introduces our algorithm. Section V compares the performance of our algorithm with that of the SDN controller with static timeout. Section V concludes this paper.

II. Motivation

When setting up the flows, all SDN switches send *packet_in* messages to the controller. They usually consume a large amount of processing power within the controller and generate many control packets over the switch-controller connection. This makes the controller or the network

Manuscript received February 1, 2014; revised May 16, 2014. This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013.

T. Kim, K. Lee, J. Lee, S. Park are with the Department of Computer Science and Engineering, Sogang University, Seoul, Korea (e-mail: maxine.kth@gmail.com, kwonyong82@gmail.com, leejunhee23@gmail.com, parksy@sogang.ac.kr).

Y.-H. Kim is with Next Communication Research Lab., Electronics and Telecommunications Research Institute, Daejeon, Korea (e-mail: yhwkim@etri.re.kr).

B. Lee is with SDN Research Section, Electronics and Telecommunications Research Institute, Daejeon, Korea (e-mail: bjlee@etri.re.kr)

DOI: 10.7763/IJFCC.2014.V3.321

overloaded and causes the whole network un-scalable. It has been reported in previous research efforts that the controller can be easily overloaded due to the large number of flow setup requests from the SDN switches. For example, HP ProCurve 5406zl switch has a flow setup rate of 275 flows per second [4]. Meanwhile, the median flow arrival rate in a datacenter with 1500 servers is around 10^5 flows per second [7] and the worst flow arrival rate in a data center with 100 edge switches is around 10^7 flows per second [8].

There are several cases where *packet_in* messages are sent to the controller. When a packet of a new flow enters the switch and corresponding entry does not exist in the flow table, the *packet_in* message is sent to the controller. If the packet enters the switch for the first time, the generation of *packet_in* message is unavoidable. However, if the flow entered the switch previously and removed due to timeout or eviction, there is a possibility that we can reduce the number of *packet_in* messages by controlling the timeout value of each flow.

If we assume that the size of a flow table is infinite, the long timeout values may decrease the flow table miss rate and the number of *packet_in* messages is minimized. Therefore, we do not need to concern about the timeout length. However, as the size of a flow table is limited, we need to find an optimal timeout value that lowers the flow table miss rate and thereby reduces the corresponding flow setup requests to the SDN controller.

The eviction occurs according to the eviction rule of the switch. The current eviction rules of the switch such as First-In First-Out (FIFO), random, and Least Recently Used (LRU), may cause more evictions since they do not consider the characteristics of each flow. For example, in a situation where the flow table is fully occupied, a flow in which packet inter- arrival time is relatively short compared to other flows can be selected as a victim and therefore is evicted from the flow table. When the evicted flow with short packet inter-arrival time enters the switch again after a short period of time, it causes a *packet_in* message to be generated since the entry does not exist in the table. If we allow other flows with longer packet inter-arrival times to be selected as a victim from the flow table, the number of unnecessary *packet_in* messages may be decreased.

III. Dynamic Timeout Control Algorithm

In order to minimize the number of evictions by dynamically adjusting timeout value of each flow, our proposed algorithm needs to answer three questions: 1) how many new flows are entering the switch, 2) how many flow entries with current timeout values are likely to remain at the next sampling period and how many empty slots are needed for the new flows, and 3) how to adjust the timeout value of each flow to reserve space for newly arrived flows. In this section, we briefly overview our algorithm and describe our approach in detail.

Fig. 1 shows an overview of the proposed algorithm. Assume that N_{flow} is the number of entries already in the table and N_{empty} is the number of empty entries at the sampling period $T_{sampling}$. Also assume that N_{flow}' is the number of entries that are likely to remain at the next sampling period $(T+1)_{sampling}$ and $N_{newflow}$ is the number of entries for newly arrived flows. First of all, in order to reserve enough spaces for newly arrived flows at the sampling period $(T+1)_{sampling}$, the number of new flows after the sampling period $T_{sampling}$ should be determined. For this, we used AutoRegression (AR) [9] to estimate the number of new flows, which determines the number of entries $N_{newflow}$ at the sampling period $(T+1)_{sampling}$. Next, we need to decide how many flow entries currently in the table at the sampling period $T_{sampling}$ are likely to remain in the table at the sampling period $(T+1)_{sampling}$, which is represented by N_{flow}'. Since $N_{newflow}$ + N_{flow}' is same as the size of the flow table, if $N_{newflow} > N_{empty}$, some of the entries at the sampling period $T_{sampling}$ need to be evicted to accommodate the newly arrived flows. On the other hand, if $N_{newflow} < N_{empty}$, there are enough spaces for the new flows, which means that the flows at the sampling period $T_{sampling}$ can stay in the flow table as long as possible to prevent unfinished evictions. After deciding $N_{newflow}$ and N_{flow}', the timeout value assigned to each flow entry at the sampling period $T_{sampling}$ is adjusted dynamically to reflect this change. In what follows, we describe the details of the proposed algorithm.

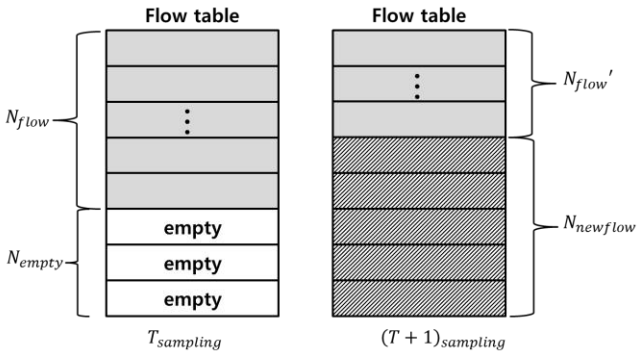

Fig. 1. Overview of our algorithm.

A. Predicting the Number of New Flows

AR is one of the most famous models for analysis and prediction of the time-series data due to its simplicity and accuracy. Since a time series is a collection of observations collected from repeated measurements and we can collect actual number of new flows from the switch, we decide to use AR to predict the number of new flows at the next sampling period. The AR has a trade-off relationship between the accuracy and computational complexity. The higher order of AR results in more accurate results but increases the computational complexity and the time for prediction.

The AR with order ρ is represented by *AR(ρ)* and is defined by (1).

$$X_t = c + \sum_{i=1}^{\rho} \varphi_i X_{t-i} + \varepsilon_t . \quad (1)$$

where φ_1, ..., φ_p are coefficients of the AR, X_t is a time-series data at time *t*, *c* is a constant value, and ε_t is a white noise value at time *t*. To obtain the coefficients of AR, we need to transform the (1) to a set of linear equations called

the Yule-Walker equations [10]. When $N_{newflow}^{t-1}$ is the number of new flows at the sampling period *t - 1* on a switch, the number of new flows at the next sampling period *t* is calculated by (2).

$$N_{newflow}^{t} = \varphi_1 N_{newflow}^{t-p} + \varphi_2 N_{newflow}^{t-p+1} + \ldots + \varphi_p N_{newflow}^{t-1} \quad (2)$$

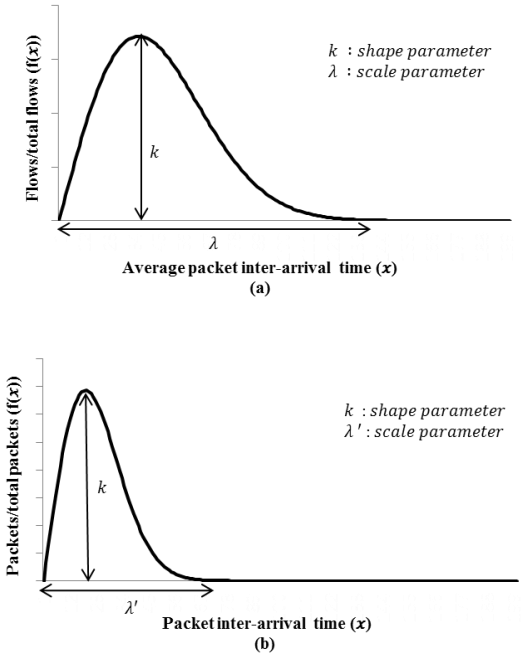

Fig. 2. Probability density function of the Weibull distribution (a) Frequency distribution of flows (b) Frequency distribution of packets of a flow.

C. Estimating the Number of Remaining Flows

The estimation of the number of flows that still remain in the flow table at the next sampling period is done by analyzing the inter-arrival times of packets in a flow.

It is generally known that the average inter-arrival times of flows entering to a switch follow a Weibull distribution [11] with a shape parameter *k* and a scale parameter λ, as shown in Fig. 2 (a). Using the self-similarity properties [8], [12], [13], [14], we can further notice that the inter-arrival times of packets that belong to the same flow also follow the same distribution as shown in Fig. 2 (b). Therefore, the Probability Density Function (PDF) of the inter-arrival times of packets that belong to the same flow is also given by (3).

$$f(x) = \frac{k}{\lambda'} \cdot \left(\frac{x}{\lambda'}\right)^{k-1} \cdot e^{-\left(\frac{x}{\lambda'}\right)^{k}}, x \geq 0 \quad (3)$$

Since the values for shape parameter *k* in both cases in Fig. 2 are identical and the average packet inter-arrival time of each flow can be obtained from Fig. 2 (a), the scale parameter λ' can be calculated by (4), where is *X* a random variable for packet inter-arrival time, *E(X)* is the mean of the Weibull distribution, and $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$.

$$E(X) = \lambda' \cdot \Gamma\left(1 + \frac{1}{k}\right) \quad (4)$$

The algorithm then determines whether the flow entry will remain in the flow table or not by estimating the remaining probability of the flow from the distribution. To estimate the remaining probability of the flow, the algorithm calculates the shaded area of the distribution in Fig. 3, which is calculated by the Cumulative Distribution Function (CDF) of the distribution with the obtained parameters above. The CDF of the distribution is given by (5). For example, if all inter-arrival times of packets belonging to the same flow are shorter than the timeout value assigned to the flow, the flow will likely to remain with a probability of 1 (i.e., *F(x)=1*), which means that the flow will always remain in the flow table according to the distribution.

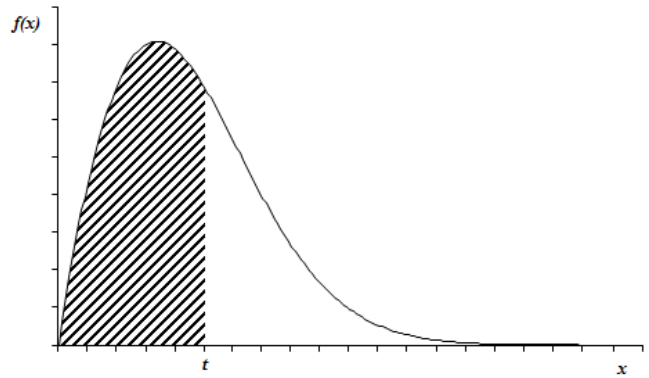

Fig. 3. Remaining probability of a flow with timeout value *t*.

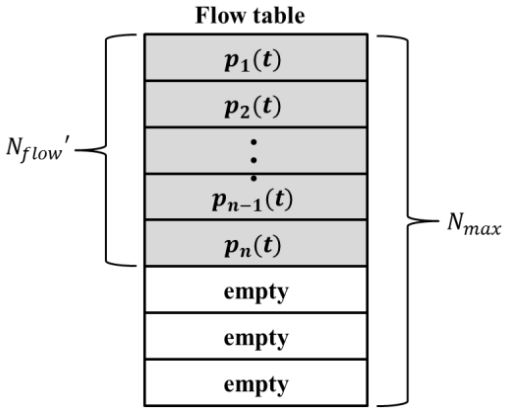

Fig. 4. Getting the number of remaining flows.

$$F(x) = 1 - e^{-\left(\frac{x}{\lambda'}\right)^{k}}, x \geq 0 \quad (5)$$

Finally, the number of remaining flows at the next sampling period can be estimated by summing up the remaining probabilities of all flows, since the remaining probability of each flow is independent one another. Let $p_f(t)$ be the remaining probability of a flow *f* with a timeout value *t*, where $f \in F$ and *F* is a set of flows in the flow table. Then, the number of remaining flows N_{flow}' can be estimated by (6) as shown in Fig. 4.

$$N_{flow}' = \left\lceil \sum_{f \in F} p_f(t) \right\rceil \quad (6)$$

For example, if there are 5 flows on the flow table and the probability of each flow is $p_1(t) = 0.3$, $p_2(t) = 0.88$, $p_3(t) = 0.5$, $p_4(t) = 0.7$, and $p_5(t) = 0.4$ then the summation of the

probabilities is 2.78, which means the number of remaining flows is 3.

C. Adjusting the Timeout Values

The main purpose of the proposed algorithm is to reserve enough spaces for the newly arrived flows. In order to achieve this goal, we dynamically increase or decrease the timeout values assigned to the flows by the amount of Δt so that flows with relatively longer inter-arrival times of packets yield spaces for the new flows. Current version of the algorithm constantly adjust the same value of Δt for all the remaining flows and we are planning to explore the mechanism of assigning different Δt values to different flows based on various traffic parameters. Therefore, after we estimate the values for $N_{newflow}$ and N_{flow}' using the mechanisms explained in sub-sections A and B, we determine Δt not to overflow the flow table in the switch by (7), where N_{max} is the maximum number of flow entries (i.e. flow table size).

$$\frac{N_{flow} + N_{newflow}}{N_{max}} \leq 1 \quad (7)$$

IV. Evaluation

In this section, we evaluate the dynamic timeout control algorithm proposed in this paper by simulation and compare its performance with that of the case without dynamic timeout control. We first present the experimental environment for the simulation and show the performance results.

A. Experimental Setup

For the simulation, we have implemented the SDN controller module, SDN switch module, and host module over OMNeT++ 4.4.1 simulator [15]. The simulation was conducted over a canonical 3-tier network topology as shown in Fig. 5, which is generally known as a conventional architecture for data centers. The topology consists of *two* layer-2 Aggregation Switches (AS), *four* layer-2 Switches (S), and *six* Top of Rack (ToR) switches connected to each layer-2 Switches (24 ToR switches). We assume that there are *five* to *twenty* hosts connected to each ToR switch (120 to 480 hosts), where each host generates packets based on the Weibull distribution (i.e., using the *weibull()* function provided by the OMNeT++ simulator).

The ratio of intra rack traffic and extra rack traffic in the experiment is 2 to 3 [8]. Intra rack traffic is the communication among severs in the same rack using the pass-through ToR switch. On the other hand, extra rack traffic means the communication between the servers which are not in one rack. For the variety of data, we used *three* kinds of protocol with three-different port numbers.

Two important parameters for the simulation are the eviction policy used in the switch and the flow table size. In order to simulate the real environment as much as possible, we simulate three most widely used eviction policies such as First-In-First-Out (FIFO), Least Recently Used (LRU), and random. Since the number of hosts varies from 120 to 480, we need to determine appropriate size for the flow table in order to simulate general eviction workloads. From the workloads reported in [4], [7], [8], we decided to use 40 to 50 entries for the flow table size. It is worthy to note that the size of a flow table in the switch is limited and the evictions from the table occur frequently in a real environment.

B. Experimental Results

In order to compare the performance of proposed algorithm with that of the case without dynamic timeout control, we have conducted three experiments: 1) comparison of the number of *packet_in* messages to the controller to show the scalability improvement, 2) comparison of the number of *packet_in* messages by varying the size of flow table, and 3) comparison of the number of *packet_in* messages by varying the number of hosts. For the experiments above, we compare the performance of our algorithm with those of other cases with static timeouts, where each case adopts three different eviction policies: FIFO, LRU, and Random.

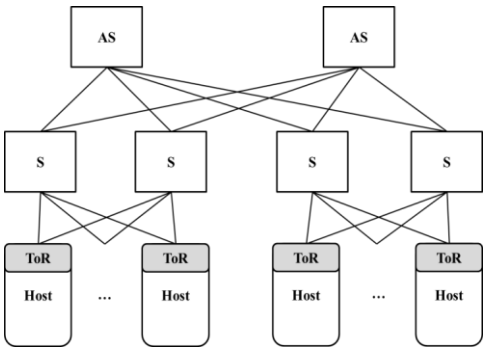

Fig. 5. Topology used in the experiment.

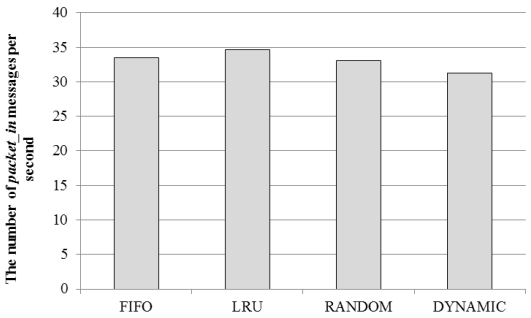

Fig. 6. Comparison of the number of messages to the controller.

Fig. 6 depicts the number of *packet_in* messages and compares the performance of our algorithm with those of other cases with different eviction policies. As shown in Fig. 6, the proposed algorithm outperforms other cases by about 5.6% to 9.9%. For example, the number of *packet_in* messages per second in the proposed algorithm is about 31.2, while the numbers in FIFO/LRU/Random cases are about 33.4, 34.7, and 33.1, respectively. The reason for this improvement is that the proposed algorithm reserves enough spaces for the new flows by adjusting the timeout value of each flow. This adjustment allows other flows with longer packet inter-arrival times to leave from the table voluntarily. This reduces the possibility of evicting wrong flow entries (e.g., evicting entries with short inter-arrival times), which may decrease the number of unnecessary *packet_in* messages.

Fig. 7 compares the number of evictions occurred in all

switches. As expected, the number of evictions decreases by 63.8% to 70.8% in the dynamic algorithm. Since we dynamically adjust the timeout, we could reduce the amount of communications between the controller and switches, and finally avoid the unnecessary evictions. As shown in Fig. 7, the evictions in static configuration are usually occurred in tier-2 switches. Therefore, it is important to reduce the load on tier-2 switches. It is worthy to note that our algorithm reduces the loads on the tier-2 remarkably.

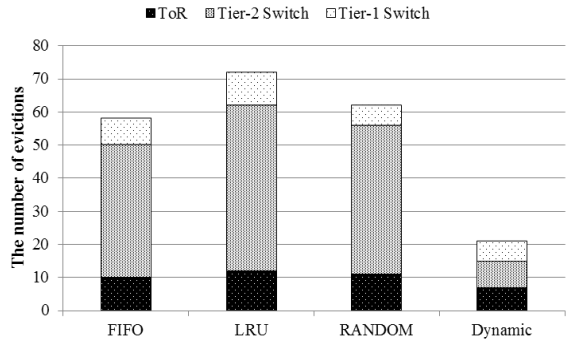

Fig. 7. Comparison of the number of evictions on the switches.

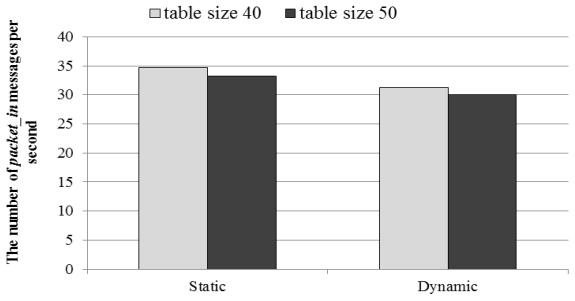

Fig. 8. Effects of varying the size of flow table.

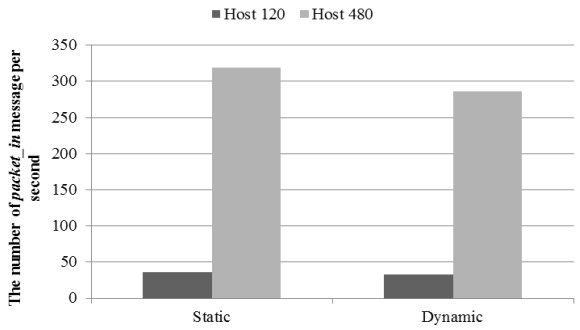

Fig. 9. Effects of varying the number of hosts.

Fig. 8 and Fig. 9 also compare the number of *packet_in* messages by varying the size of flow table and the number of hosts where both Static and Dynamic adopted LRU eviction policy. As we increase the size of flow table from 40 to 50 in Fig. 8, the performance gap between our approach and other cases is getting narrow; from 11% to 8.7%. It should be noted that if we increase the size of flow table, the probability of evictions gets lower, which also reduces the number of *packet_in* messages. However, as we increase the number of hosts in Fig. 9, the performance gap is widening; from 7.6% to 10%. Considering the packets generated from the hosts are getting larger as we increase the number of hosts, it is clear that the chance of getting evicted from the table is also getting larger. In this case, the proposed algorithm lowers the chance of evictions by dynamically changing the timeout values.

V. Conclusion

We have witnessed a large number of research efforts to improve the scalability of SDN controllers. Among them, this paper has focused on improving the scalability of a single SDN controller by reducing the number of flow setup requests to the controller. Although there are still several studies to improve the scalability of a single SDN controller, our approach is novel in that we dynamically adjust the timeout value of each flow based on current traffic conditions in order to minimize the number of evictions. As a result, the proposed scheme allows the switch to reserve enough spaces for newly arrived flows, while making other flows with relatively longer packet inter-arrival time yield their table entries automatically via timeout.

Current version of the proposed algorithm uses AR to predict the number of new flows and utilize the Weibull distribution to estimate the number of remaining flows. Since those estimation schemes are unlikely to produce accurate results all the time, extra spaces in the switch can be reserved to compensate these inaccuracies. However, there is certainly a tradeoff between reserving extra spaces for inaccurate predictions to avoid unnecessary flow setup requests and utilizing the whole table spaces. It is also a research question that how many spaces we need to reserve. We are planning to investigate those issues in the future. We are also currently working on assigning different Δt values to different flows based on various traffic parameters.

Acknowledgment

This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013.

References

[1] 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, April 2008.

[2] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *Proc. Usenix Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 10, pp. 1-6, October 2010.

[3] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *Proc. 2nd ACM SIGCOMM workshop on Hot topics in Software Defined Networking (HotSDN)*, pp. 7-12, August 2013.

[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, no. 4, pp. 254-265, August 2011.

[5] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 351-362, October 2010.

[6] A. Zarek, "Open flow timeouts demystified," Computer Engineering Research Group: University of Toronto, 2012.

[7] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. 9th ACM SIGCOMM conference on Internet Measurement Conference (IMC)*, pp. 202-208, November 2009.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM conference on Internet Measurement Conference (IMC)*, pp. 267-280, November 2010.

[9] H. Akaike, "Fitting autoregressive models for prediction," *Annals of the Institute of Statistical Mathematics*, vol. 21, no. 1, pp. 243-247, 1969.

[10] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, John Wiley & Sons, 4th ed, 2013.
[11] W. Weibull, "A statistical distribution function of wide applicability," *Journal of Applied Mechanics*, vol. 18, pp. 293-297, 1951.
[12] P. Olivier and N. Benameur, "Flow level IP traffic characterization," *Teletraffic Science and Engineering*, vol. 4, pp. 25-36, 2001.
[13] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes," *IEEE/ACM Transactions on Networking (TON)*, vol. 5, no. 6, pp. 835-846, December 1997.
[14] D. A. Martinez, "A verification of selected properties of Telecommunication traffic generated by OPNET simulator," Erasmus exchange project work: University of Ljubljana, 2011.
[15] OMNeT++. [Online]. Available: http://www. omnetpp.org.

Taek Hee Kim has received B.S. degrees from the Computer Science and Engineering Department at Sogang University, Korea in 2013. She is currently studying for the M.S. degree of the Computer Science and Engineering Department at Sogang University, Korea. She has been a researcher at Distributed and Cloud Computing Laboratory, Sogang University. Her research interests include cloud computing and systems, virtualization technologies.

Kwonyong Lee is a doctoral candidate of Computer Science and Engineering Department at Sogang University, Korea. He has received B.S. and M.S. degrees from the Computer Science and Engineering, Sogang University, Korea, in 2007 and 2009, respectively. His research interests have lied in cloud computing, virtualization technologies and distributed computing for big data. His current research is mainly focused on improving the performance of virtual cluster, where he is developing systems and related algorithms to improve the performance of virtual cluster running on cloud and to utilize cloud resources more efficiently.

Junhee Lee has received B.S. degrees from the Computer Science and Engineering Department at Sogang University, Korea in 2014. He is currently studying for the M.S. degree of the Computer Science and Engineering Department at Sogang University, Korea. He has been a researcher at distributed and cloud computing laboratory, Sogang University. His research interests include cloud computing and systems, virtualization technologies.

Sungyong Park is a professor in the Department of Computer Science and Engineering at Sogang University, Seoul, Korea. He received his B.S. degree in computer science from Sogang University, and both the M.S. and Ph.D. degrees in computer science from Syracuse University. From 1987 to 1992, he worked for LG Electronics, Korea, as a research engineer. From 1998 to 1999, he was a research scientist at Telcordia Technologies (formerly Bellcore), where he developed network management software for optical switches. His research interests include cloud computing and systems, virtualization technologies, autonomic computing, and embedded system software.

Young-Hwa Kim is a principal member at ETRI, Korea. He received a B.S. degree at Chunnam National University in 1987, M.S. and Ph.D. degrees at Chungnam National University in 1997 and 2005. Joining in ETRI from 1988, he had worked for signaling system No.7, intelligent network, access network, flow-based router, automatically switched optical network projects, and future Internet. Currently, he participates in the SDN project. His interest areas include communication networks and systems, future internet, software defined networking, and so on.

Byungjoon Lee is a senior researcher of SDN Research Section, ETRI, and Republic of Korea. He received his master degree at Seoul National University in 1998, and received Ph.D. at Chungnam National University in 2011. His key research interests include future internet, software defined networking, and information-centric networking.