2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR) | 978-1-6654-0607-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPSR54439.2022.9831366

A Dynamic Flow Table Management Method Based on Real-time Traffic Monitoring

Jinyuan Zhang*, Xuanbo Huang*, Jian Li*, Kaiping Xue*†‡, Qibin Sun*, Jun Lu†
* School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China
† Department of EEIS, University of Science and Technology of China, Hefei, Anhui 230027, China
‡Corresponding author,lijian9@ustc.edu.cn (J. Li), kpxue@ustc.edu.cn (K. Xue)

***Abstract*—In Software-Defined Networking (SDN), the controllers implement flexible and scalability networking policies by installing different flow rules. Each rule matches a specific class of flows, instructs the switches to execute actions, and then expires when they finish their tasks. OpenFlow introduces the *timeout* mechanism to manage these flow rules. However, finding a reasonable *timeout* value becomes a difficult problem for the network managers. When a relatively small *timeout* value is given to an elephant flow, the rule expires early, introducing extra cost for the controller and long latency for the matching flow, respectively. On the contrary, a large *timeout* value for a mice flow makes a rule occupy the switch memory too long, wasting the caching memory and causing the flow table prone to overflow. Therefore, it is necessary to allocate appropriate timeouts for different flows dynamically. In this paper, we achieve this goal with real-time traffic monitoring and heuristic algorithms. By considering different network loads and designing corresponding dynamic *timeout* algorithms for different scenarios, we make full use of the advantages of SDN to improve the utilization rate of the switch memory and save the controller resources. Further, we implement our scheme in a simulation SDN platform and evaluate the algorithms with the public datasets. Experiments show that our scheme has low control overhead and is memory efficient compared with current mechanisms.**

***Index Terms*—Software-Defined Networking (SDN), flow table management, timeout mechanism, traffic monitoring**

I. Introduction

Software-Defined Networking (SDN) is a new networking framework that overcomes the defects of traditional networks by separating the control plane and the data plane. Benefiting from the programmability of SDN, network operators manage their network resources flexibly through the southbound interface between the control and the data planes. Among all the southbound communication protocols, OpenFlow [1] is the de facto standard in the SDN implementations. OpenFlow allows controllers to install flow rules on the switches to implement agile networking functions. Currently, the commercial off-the-shelf switches store these flow rules in the so-called ternary content addressable memory (TCAM) to quickly find the matches for a specific flow from thousands of rules [2]. Storing flow rules in TCAM makes it costs O(1) time complexity to look up a specific match in flow tables. However, TCAM is expensive and power starving and therefore size-limited. Besides, flows may need to be processed by multiple network functions, *e.g.,* resource allocation, anomaly detection and traffic engineering, which need lots of flow entries for implementation. Although the network operators can reduce the flow entries used by implementing wildcard rules, fine-grained flow management still needs exact-match rules and has irreplaceable advantages for some important applications in the network [3]. Thus, it is significant to use the flow entries efficiently. However, the dynamic network functions and the large amount of flows bring critical challenges for efficient flow table management in SDN networks.

Using a small *timeout* value for all flow rules might save the flow entries but brings other drawbacks. For example, some flows cannot find their matching rules during the processing. When such a no-match happens, switches encapsulate that flow with a `Packet_in` message and send it to the controller, according to OpenFlow1.5 [4]. However, this introduces high latency for the flow [5] and also increases the overhead of the controller [6], which is prone to be the bottleneck in SDN networks [7].

Therefore, choosing a suitable *timeout* value for each flow is a challenging problem. From the discussion above, there is a trade-off between the controller computation resources and the switches' memory. As shown in Fig.1, we use the number of `Packet_in` messages to show the controller's overhead, while we calculate the occupancy of the switches' flow table. With the increase of *timeout*, the controller's overhead monotonically decreases while the occupancy increases fast.

So, can we find a method that is both controller resources efficient and switch memory efficient? In this paper, we propose a dynamic timeout management mechanism for the SDN flow tables to achieve the goal mentioned above. Based on the real-time traffic monitoring, our heuristic algorithm divides the network state into three different statuses, then calculates the *timeout* for each specific flow based on their statistical data. To summarize, this paper makes the following contributions:

1) We propose a dynamic flow table *timeout* management algorithm. Based on the real-time traffic monitoring, we calculate the *timeout* of each flow with their statistical characteristics and dynamically change them with the feedback.
2) By dividing the network loads into different scenarios, we adjust our algorithm with different parameters, which further improves the efficiency of our algorithm.
3) We design and implement our scheme in a prototype

SDN system with the Open vSwitch[1], and evaluate the efficiency with the public flow datasets. Extensive experiments show that our method can efficiently save the number of `Packet_in` messages, which shows the controller resources efficiency of our scheme. Besides, our approach can effectively reduce the number of flow rule evictions, which shows the switch resource efficiency of our strategy.

The rest of this paper is organized as follows. Section II introduces the background and motivation of this paper. Section III discusses some related work. We describe our algorithm and the architecture and the implementation of our scheme in Section IV. And in Section V, we carry out simulation experiments for different network scenarios to verify our method. Finally, we draw the conclusion of this paper in Section VI.

II. Background and Motivation

In this section, we first summarize the current flow table management in Open vSwitch (OVS), and analyze the advantages and disadvantages of these strategies. Then we introduce the related work of this paper. And finally, we explain the problem we are trying to solve and raise the motivation of our scheme.

A. Flow Table Management in OVS

OpenFlow provides three mechanisms for flow table management [4], the flow timeout, the proactive flow entry deletion, and the flow entry eviction.

For the *timeout* mechanism, OVS provides the following two types of timeouts:

- Idle Timeout: If no packet matches the flow entry during the idle *timeout* period, the flow entry will be removed from the flow table.
- Hard Timeout: Any flow entry whose duration reaches its hard *timeout* will be removed.

With these two types of timeouts, the OVS removes the flow entries that are no longer in use as soon as possible. However, the flow table may still overflow when there are lots of new flows. Hence, OVS provides an Eviction Policy for this situation. The eviction process only considers flows with an idle *timeout* or a hard *timeout*, because OVS does not support eviction mechanisms such as LRU and FIFO, but can only be approximated by *timeout* mechanism. When a flow must be evicted due to overflow, OVS will choose the flow entry that expires soonest for eviction. This evict policy still relies heavily on the *timeout* allocation.

What's more, OVS provides a proactive flow entry deletion initiated by the controller. The controller sends a `Flow_mod` message to the switch with the `OFPFC_DELETE` command. Once the switch receives this message, it proactively deletes the flow entry that meets the condition, even if the flow has not reached the timeout. While doing so does save the flow table space by deleting infrequently used flow table entries in advance, managing flow tables through controller delivery control messages has high communication overhead and high latency.

In fact, these three flow table management strategies are independent of each other. In this paper, the flow table *timeout* mechanism is mainly adopted for two reasons: First, the *timeout* mechanism does not introduce additional communication overhead; Second, the *timeout* mechanism does not require the replacement of existing switch hardware devices, which facilitates future deployment.

B. Motivation of Our Scheme

Either idle *timeout* or hard *timeout*, the optimization goal of *timeout* mechanism is to increase the hit rate of all flows within limited storage space. However, *timeout* can greatly improve flow table utilization, only when it is set properly. We make statistics and analysis on the number of `Packet_in` messages and the average occupancy of the flow table under different fixed timeouts.

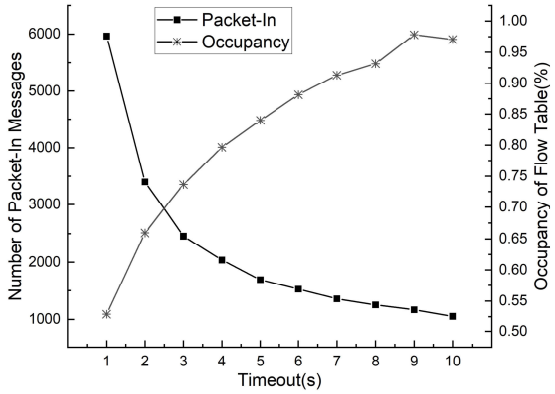

Fig. 1: The trade-off relationship between control overhead and flow table occupancy.

As shown in the Fig.1, small *timeout* will cause that switches to send more superfluous messages to controllers for new flow rules, increasing the overhead of the control channel. And large *timeout* leads to a long rule lifetime, which means that more expired entries will be stored in the flow table, resulting in the overflow of the flow table. However, most flow table management strategies usually use fixed *timeout* obtained from experience, which cannot cope with the complex and changeable network environment well. Fixed *timeout* results in inefficient use of switch flow table storage which means that it may be too long for short-live flows or too short for long-live flows. Therefore, it is necessary to dynamically assign different timeouts to different flows.

Based on the analysis above, our scheme is mainly designed to solve the following two problems: On the one hand, in the face of the complex and changeable network environment, it is very difficult and user-unfriendly to set an appropriate fixed *timeout* without prior knowledge. On the other hand, setting the same *timeout* for streams with different characteristics will result in low utilization of the flow table.

[1]https://www.openvswitch.org/, Feb. 2022

Our scheme can not only improve the scalability of SDN, that is, dynamically adapt to different network environment, but also effectively improve the utilization efficiency of the flow table, especially for dealing with the situation where the flow is particularly heavy and the flow table is insufficient.

III. Related Work

There have been some existing solutions to alleviate the space strain problem of flow table by *timeout* mechanisms. Vishnoi *et al*. [8] proposed an effective switch memory management method for SDN named SmartTime, which combined a dynamic *timeout* heuristic to compute efficient idle timeouts with proactive eviction of flow entries. Zhang *et al.* [9] proposed an adaptive hard *timeout* method (AHTM) to improve the flow table utilization by modeling the *timeout* optimization problem as a queue system and achieving the balance between blocking probability and extra workload to SDN controller. To improve the efficiency of AHTM, Zhang *et al.* [10] proposed TimeoutX, which combined traffic characteristics, flow types and flow table utilization in flow table management. TimeoutX [10] and AFTM [11] do not use the monitor, but only collect information through the messages when the flow entry is installed or removed, which results in the controller not being aware of the occupancy of the flow table in time.

In other work, [12], [13] and [14] use the proactive flow entry deletion in the flow table management policy. Although this scheme is effective, it requires the controller to continuously send deletion requests to the flow table, and the deletion instructions will have a very bad impact on the real-time performance of the switch. In [15], a management mechanism named intelligent *timeout* master was proposed, which for the first time introduced a feedback mechanism in SDN to adjust the maximum *timeout* value accordingly through flow table load. However, this feedback mechanism is too simple, and the feedback to the maximum *timeout* value does not perform well.

IV. Design and Implementation

In this section we will introduce the framework and implementation of our algorithm in detail. The algorithm is mainly divided into four parts: A. Flow monitoring module; B. Timeout calculating module; C. Flow table management module; D. Flow statistic database. The architecture of these three modules is shown in Fig.2, which fully works on the control plane.

The monitoring module wakes up periodically and queries the current load and traffic statistics of the switch. The database stores the traffic statistics from the monitoring module or `Packet_removed` messages. When a flow arrives at the switch and sends `Packet_in` message to the controller, the controller invokes the corresponding *timeout* calculating algorithm according to the current load, and then delivers the flow entry through the flow table management module.

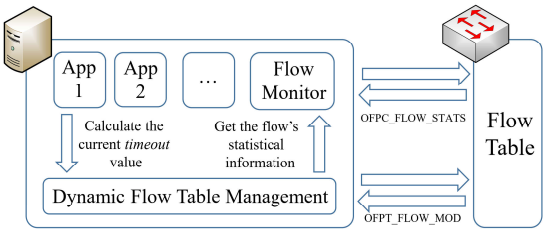

Fig. 2: Architecture of our scheme.

A. Flow Monitoring Module

The flow table monitoring module is designed to collect the statistic of flows and flow tables periodically. The monitor sends status requests to the switch at regular intervals (1 second in this work). This period can be adjusted according to the network environment. The 1 second set in this paper works for most common cases. Through this module, we mainly collect the following two pieces of information about flows and flow tables, which can help the computing module capture the characteristics of different flows:

1) λ: Packet arrival rate of flow in the previous period.

2) μ: Current occupancy of the flow table.

The parameter λ reflects the traffic of different flows, so that we can assign a larger *timeout* to a flow with large traffic and a smaller timeout to a flow with small traffic. While the parameter μ reflects the current flow table usage, so that we can timely control the flow table when the flow table is about to overflow. When the monitor collects the statistics above, the data in the flow statistic database will be updated.

B. Flow Statistic Database

The flow statistic database collects data from two places: the monitor and the switch. Whenever a flow table is removed due to timeout, the switch proactively sends statistics about the flow table to the controller. The controller will update the database with this information. Each flow through the switch is stored as an entry as follows in the flow statistic database.

Hash	***Repeat_Count***	***lastRemoved***	***lastTimeout***	λ

Fig. 3: The Entry in Flow Statistic Database.

Here, we use the hash value of the triplet (srt, dst, in_port) to represent the different streams. This can greatly reduce the storage of the database and also reduce the time it takes to find the database. *Repeat_Count* is used to count the number of stream repeats, incremented by 1 for each repeat. The field of *lastRemoved* stores the last time the flow entry was deleted, while the *lastTimeout* stores the *timeout* value that allocated last time. Finally, the parameter λ represents the packet arrival rate of the flow during the previous period of the flow table, which is collected by the flow monitoring module.

C. Flow Table Management Module

This module is mainly responsible for the delivery of flow entries. Each time a new flow arrives on the switch, the switch generates a `Packet_in` message and sends it to the controller. Then flow table management module needs to call

the *timeout* calculating module and get the appropriate timeout value for the new flow. And finally, it will install the dynamic flow entry on the switch.

D. Timeout Calculating Module

The main function of this module is to calculate the appropriate *timeout* of different flows through the data provided in the database. There are two main phases: the slow start and the overflow avoidance as mentioned above. To achieve the above scheme, a dynamic *timeout* based on flow statistics and flow table utilization is summarized as Alg.1.

Algorithm 1: Dynamic *timeout* Calculating Algorithm in Different Modes

Input: t_{init}, t_{max}, $H = hash(src, dst, in\_port)$;
Output: $Timeout$;
1 ***Require:*** TB_MODE, $Inf_H = \phi(H)$, T_{pktin};
2 **timeout *Calculation:***
3 **if** $Inf_H == NULL$ **then**
4 Log the new flow into database;
5 $T = t_{init}$;
6 **else**
7 $T_{interval} = T_{pktin} - Inf_H.lastRemoved$;
8 **if** $TB\_MODE == OAM$ **then**
9 **if** $Inf_H.\lambda < \lambda_{TH}$ *and* $T_{interval} > INR\_Threshold$ **then**
10 Skip the process of the flow entry install;
11 **else**
12 $T = ceil(Inf_H.lastTimeout/2)$;
13 **end**
14 **else if** $TB\_MODE == HLM$ **then**
15 $T = Inf_H.lasttimeout + 1$;
16 **else**
17 $T = Inf_H.lasttimeout * 2$;
18 **end**
19 $T = min\{T, T_{init}\}$;
20 $T = max\{T, T_{max}\}$;
21 Update the flow information in the database;
22 **end**

It then determines the current load mode in real time based on the information gathered above. We define three different load modes for various scenarios and design dynamic algorithms for each mode.

a) Low Load Mode (LLM): When the monitor detects that $\mu \leq 50\%$, the flow table is currently under low load. In this case, it increases the *timeout* of the flow table exponentially from a small timeout, as shown in line 21 of the code. The small initial *timeout* is set because of the high percentage of short traffic on the Internet, and it needs to take some time for the controller to collect statistics of new flows. If the occupancy of the flow table remains low, then the *timeout* grows up to a preset upper limit t_{max}, which is set to prevent high overflow rates due to excessively long timeout. This allows long flows to be assigned a long *timeout* and short flows to be assigned a short timeout.

b) High Load Mode (HLM): When $50\% < \mu \leq 90\%$, the load of the flow table is already high, and the *timeout* with exponential growth may cause the overflow of the flow table, so we use additive growth to adjust the timeout. Whenever the flow repeatedly sends `Packet_in` messages to the controller, we increase its *timeout* by one second each time which can appropriately reduce the survival time of entries in the flow table.

c) Overflow Avoidance Mode (OAM): In many existing schemes, *timeout* only increases but not decreases, which may reduce the utilization of the flow table. Once the monitor detects flow table occupancy $\mu > 90\%$, the algorithm first calculates the expiration interval of the flow table through the following formula. For packets with large packet arrival rates and small arrival intervals. We refer to the idea of multiplication reduction in AIMD to avoid flow table overflow as much as possible by halving the *timeout* and rounding up.

According to the analysis of data flow [16], many short flows have only 1-2 packets, or the interval between packets is very large. In our experiments, we find that many flow entries are not even called once from installation to deletion. For this type of stream that is not commonly used, either the *timeout* mechanism or the early deletion mechanism still has some overhead. We set an upper limit on the packet arrival rate and a lower limit on the arrival interval, and the flow exceeding the limits is judged to be an infrequently used flow(line9∼10). For this type of flow, we skip the process of installing the flow table on the switch and forward it directly. Compared with *timeout* mechanism and proactive flow entry deletion mechanism, direct forwarding of short-flow packets under high load can omit unnecessary flow entry installation and deletion process, which can not only reduce the control overhead between controller and switch, but also effectively improve the utilization of flow table to avoid overflow.

V. Evaluation

To prove the feasibility of our scheme, we test the performance of our scheme on the overhead between controller and switch and the utilization of flow table, compared with the fixed *timeout* scheme under different flow table sizes. In this section, we will first introduce the simulation environment and experimental parameters, including the simulation platform, datasets, simulation parameters and evaluation metrics. Later, we will present the experimental results and analysis in detail.

A. Simulation Environment

We take Ryu[2] as the controller of SDN and use Mininet[3] to simulate the network topology. Our solution is implemented as an application that can be deployed on the controller without requiring changes to existing OVS hardware devices, which means it has better portability. As for the experimental data, we used two different datasets to reflect the performance of our solution in different network scenarios. Datasets UNIV1 and UNIV2 proposed in [17] have been widely used in previous

[2]https://osrg.github.io/ryu/, Feb. 2022
[3]http://mininet.org/, Feb. 2022

work about flow table management [8] and [14]. The first data set represents a trace in which most of the traffic is TCP flow. And most of the traffic in the second dataset is UDP flow, representing another data center traffic that contains more burst flows. We respectively extract the first 100,000 data packets and 500,000 data packets from the two datasets for our experiment. The specific characteristics are shown in the following table.

TABLE I: The Characteristics of Datasets.

	Duration(s)	*Packet_count*	*Flow_count*
UNIV1	37	100000	390
UNIV2	55	500000	425

In the algorithm proposed in SectionIV, some parameters, such as the upper limit of flow interval and the limit of large and small flows, can be adjusted dynamically. In our implementation, we give a set of most general parameters, $T_{init} = 1s$,$T_{max} = 10s$.

We mainly chose two metrics to evaluate the improvement of system performance by our algorithm:

- `Packet_in` number: It refers to the number of `Packet_in` messages between the switch and the controller. When a flow arrives at the switch, the switch first matches the flow entries that have been installed. If no suitable flow entry is found in the flow table, the switch will send a `Packet_in` message to the controller. Therefore, it reflects the cost of control channel, and also the utilization of the flow table by counting the number of table_misses.
- Eviction number: This metric counts the number of flow table eviction. We use the default eviction policy that provided by OVS, which chooses the flow entry that expires soonest for eviction. It is used to reflect the number of flow table overflow.

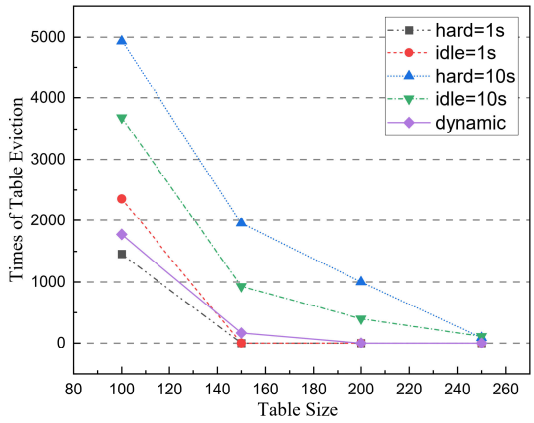

Fig. 4: Eviction Number for UNIV1.

We simulate different network load conditions by setting different flow table sizes. In this simulation, when $table\_size = 100$, the flow table is seriously insufficient and will overflow frequently. When $table\_size = 150$, the flow table occupies a high rate and is in high load mode. While $table\_size > 150$, the flow is in low load mode.

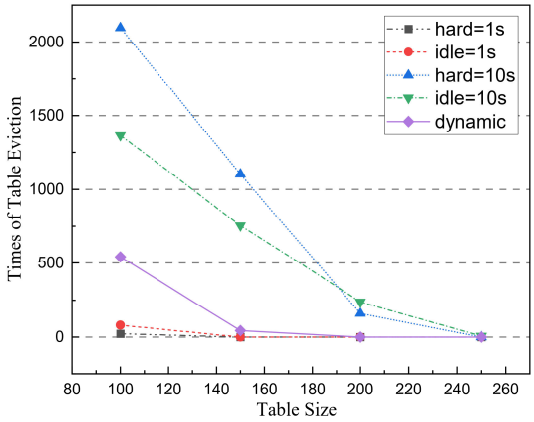

Fig. 5: Eviction Number for UNIV2.

B. Results and Discussion

The results of our experiment are shown from Fig.4 to Fig.7. The first two graphs correspond to the dataset UNIV1, while the last two graphs correspond to the dataset UNIV2.

a) Dynamic Adjustment of Flow Timeout: Firstly, we will analyze the necessity of dynamic adjustment of flow timeout. It can be seen that when the fixed *timeout* is 1s, the times of table eviction is all at a very low level, but the number of `Packet_in` messages is relatively high. In contract, when *timeout* is set to 10s, there are higher eviction times and lower `Packet_in` messages. That is, the flow table occupancy and the control overhead between switches and controllers are two tradeoff factors, which is consistent with our analysis in SectionII. Therefore, choosing an appropriate fixed *timeout* in a complex and changeable network is difficult and user-unfriendly. Compared with the high control overhead of 1s fixed *timeout* and the high overflow times of 10s, our scheme dynamically selects the appropriate *timeout* for each flow, which makes a good balance between these two factors.

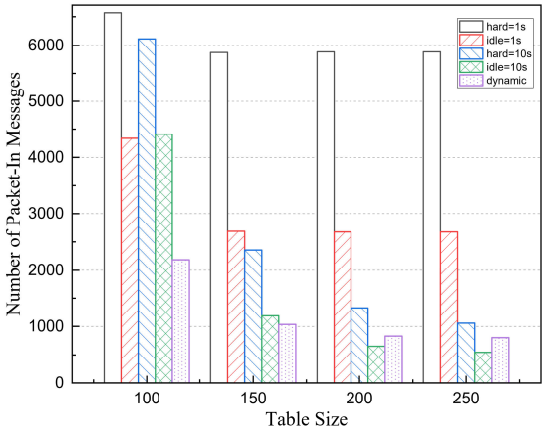

Fig. 6: Packet-In Number for UNIV1.

b) Improvement of Flow Table Utilization: Then, we will analyze the improvement of flow table utilization by our algorithm. It is obvious that if a *timeout* is simply chosen between 1s and 10s, then both the number of `Packet_in` messages and the times of flow table overflow should be

somewhere in between. However, our dynamic algorithm works better at control overhead, especially in the case of high load. Taking $table\_size = 100$ as an example, the number of control messages in our scheme is reduced by $50\% \sim 67\%$ as shown in Fig.6, while it is reduced by $10\% \sim 68\%$ as shown in Fig.7. It is because that we use additional growth under high load conditions and effectively filter some special flows when overflow occurs, which greatly improves the utilization efficiency of the flow table.

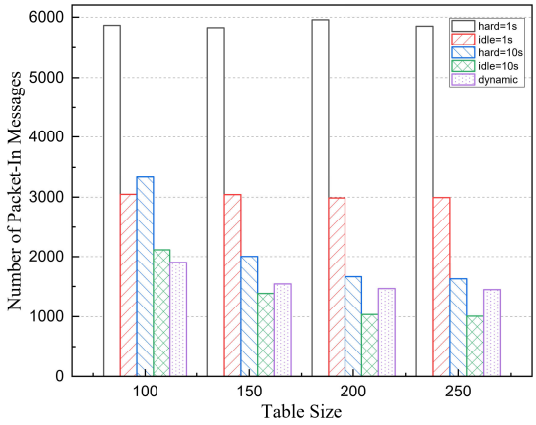

Fig. 7: Packet-In Number for UNIV2.

Compared with the fixed *timeout* of 10s, our scheme may not improve the number of `Packet_in` messages significantly, especially under low load conditions. This is because in our scheme, it needs to take some time to get the statistics for the flows, and the timeouts will increase from a small *timeout* to a large timeout. We have used exponential growth in our algorithm to speed up convergence time as much as possible.

c) Additional Overhead: The monitor we introduced in this work will carry some additional control overhead, that is, the controller needs to send packets to the switch every cycle to request information about the status of the flows and flow table. However, a small amount of overhead doesn't have a bad effect on the system. For example, the monitoring period is 1s in our algorithm. In the case of tight storage space, one query per second is negligible compared to the thousands of `Packet_in` messages reduced by our algorithm. And if storage space is sufficient, the bandwidth of the control channel is also sufficient for the additional overhead.

VI. Conclusion

In this paper, we proposed a real-time monitoring based flow table management strategy. The network monitor observes network load in real-time and collects statistics for each network traffic. Then we proposed a heuristic algorithm to calculate the *timeout* for each flow, considering the network load, flow repetition interval, and the packet arrival rate. Experiments show that our strategy can efficiently save the controller resources and the switch resources.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant No. 61972371 and No. U19B2023, and Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, *et al.*, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, 2008.

[2] D. Kreutz, F. Ramos, P. V., *et al.*, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] G. Zhao, H. Xu, J. Fan, L. Huang, and C. Qiao, "HiFi: Hybrid rule placement for fine-grained flow management in SDNs," in *Proceedings of the IEEE International Conference on Computer Communication (INFOCOM)*, pp. 2341–2350, 2020.

[4] "Openflow v1.5 specification." [Online], 2022. Available:https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[5] M. Zhang, G. Li, L. Xu, J. Bai, M. Xu, G. Gu, and J. Wu, "Control plane reflection attacks and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 623–636, 2021.

[6] X. Huang, K. Xue, Y. Xing, D. Hu, R. Li, and Q. Sun, "Fsdm: Fast recovery saturation attack detection and mitigation framework in sdn," in *Proceedings of the 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 329–337, 2020.

[7] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 165–166, 2013.

[8] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in openflow networks," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pp. 177–188, 2014.

[9] L. Zhang, R. Lin, S. Xu, and S. Wang, "Ahtm: Achieving efficient flow table utilization in software defined networks," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, pp. 1897–1902, IEEE, 2014.

[10] L. Zhang, S. Wang, S. Xu, R. Lin, and H. Yu, "Timeoutx: An adaptive flow table management method in software defined networks," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2015.

[11] Y. Shen, C. Wu, Q. Cheng, and D. Kong, "Aftm: An adaptive flow table management scheme for openflow switches," in *Proceedings of the 22nd International Conference on High Performance Computing and Communications; 18th International Conference on Smart City; 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 917–922, IEEE, 2020.

[12] T. Kim, K. Lee, J. Lee, S. Park, Y.-H. Kim, and B. Lee, "A dynamic timeout control algorithm in software defined networks," *International Journal of Future Computer and Communication*, vol. 3, no. 5, p. 331, 2014.

[13] H. Liang, P. Hong, J. Li, and D. Ni, "Effective idle_timeout value for instant messaging in software defined networks," in *Proceedings of the 2015 IEEE International Conference on Communication Workshop (ICCW)*, pp. 352–356, IEEE, 2015.

[14] L. Wang, C. Song, Z. Xu, *et al.*, "Proactive mitigation to table-overflow in software-defined networking," in *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*, pp. 00719–00725, IEEE, 2018.

[15] H. Zhu, H. Fan, X. Luo, and Y. Jin, "Intelligent timeout master: Dynamic timeout for sdn-based data centers," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 734–737, IEEE, 2015.

[16] A. Zarek, Y. Ganjali, and D. Lie, "Openflow timeouts demystified," *Univ. of Toronto, Toronto, Ontario, Canada*, 2012.

[17] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 267–280, 2010.