Intelligent Timeout Master: Dynamic Timeout for SDN-based Data Centers

Huikang Zhu*, Hongbo Fan†, Xuan Luo†, Yaohui Jin†
*State Key Laboratory of Advanced Optical Communication Systems and Networks, Shanghai Jiao Tong University
†Network & Information Center, Shanghai Jiao Tong University, China
{ 13142e, jackwindows, xluo, jinyh }@sjtu.edu.cn

***Abstract*—Software Defined Networks (SDN) such as OpenFlow provides better network management for data center by decoupling control plane from data plane. Current OpenFlow controllers install flow rules with a fixed timeout after which the switch automatically removes the rules from its flow table. However, this fixed timeout has shown many disadvantages. For flows with short packet interval, the timeout may be too large so that flow rules stay in the flow table for too long time and result in unnecessary occupation of flow table; for flows with long packet interval or periodic flows, the timeout may be too short, hence producing too many packet-in events and causing overload on the controller. In this paper, we propose the Intelligent Timeout Master, which can assign suitable timeout to different flows according to their characteristics, as well as conduct a feedback control to adjust the max timeout value according to the current flow table occupation, in an effort to avoid flow table overflow. In our experiments, we use real traffic trace and the result confirms that our Intelligent Timeout Master performs quite well in reducing the number of packet-in events as well as flow table occupation.**

***Keywords*—Software Defined Networks (SDN), OpenFlow, dynamic timeout, timeout prediction, load aware**

I. INTRODUCTION

The software defined network (SDN) is an emerging network management paradigm, one of its open standards is OpenFlow [1], which has been widely used in industry and academia.

In OpenFlow networks, traffic is forwarded according to flow rules. However, switches cannot store unlimited number of flow rules, hence a timeout mechanism is introduced to flush them. There is a specific timeout for each flow rule which represents the maximum amount of idle time before this flow rule entry is removed by the switch. Different values of timeout have been adopted in recent works, ranging from 5 seconds in the NOX [2] and Floodlight [3] to 20 and 60 seconds in DevoFlow[4]. However, all these timeouts are fixed in traditional OpenFlow networks [5] and don't take the characteristics of different flows into consideration. The absence of an intelligent mechanism to adaptively adjust timeout assigned to flow entries has led to inefficient use of the flow table resources and unnecessary overhead on the controllers. For example, suppose a flow f_1 is assigned with a timeout T_1 smaller than its packet interval I_1. Then this flow will trigger packet-in event for each of its packet, leading to an overload on the controller since ideally it should only trigger packet-in event once. A large number of packet-in events due to the premature removal will consume many resources and add a significant load on the controller. For another example, suppose a flow f_2 whose largest packet interval is I_2. If we assign a significantly larger timeout value T_2 to this flow, the flow rule will stay in the flow table for additional $T_2 - I_2$ time after this flow ends, which leads to unnecessary redundant and cost. Therefore, adaptive timeout is of great importance since fixed timeout cannot work well in both circumstances.

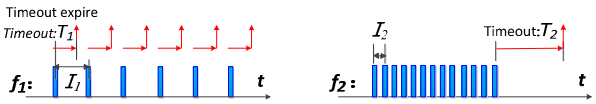


Fig. 1. Example of a too small timeout and a too large timeout

Besides, another important issue about timeout is the load of flow table. Currently, in order to guarantee fast looking up and good forwarding performance, the flow entries are mostly stored in TCAM (Ternary Content Addressable Memory). However, TCAM is quite expensive in terms of the cost and energy. Consequently, the flow table supports a very limited number of entries. For example, the HP5406zl OpenFlow -enabled switch only supports about 1500 entries[4]. Intuitively, heavier traffic load or larger timeout will lead to more TCAM occupation. In some extreme cases, the flow table may overflow, resulting in denial of service and massive packet-in events. Therefore, the load of flow table should be taken into consideration in an effort to prevent flow table overflow.

In this paper, we propose Intelligent Timeout Master to solve the problems mentioned above. The contributions of this work include:

- We propose Intelligent Timeout Master for OpenFlow to make better use of flow table resources. We add a cache module in controller to record the last expire time and use history based algorithm to predict the suitable timeout value. Thus, different timeouts are assigned to flows according to their characteristics.
- Our design exploits the flow table load as feedback control to adjust the max timeout value accordingly, which can help avoid the overflow of the flow table. To the best of our knowledge it's the first work to apply the idea of feedback control to the timeout assignment.

II. DESIGN

Aiming at making efficient use of the flow table resources, we propose the scheme of Intelligent Timeout Master shown in Fig.2, to dynamically manage the timeout. It is composed of two core parts, the cache based timeout prediction module and the load aware feedback control module. The former one adjusts timeout values according to the characteristics of flows, while the latter one serves as a feedback control to adjust the max timeout values according to current load of TCAM, thus to avoid overflow.

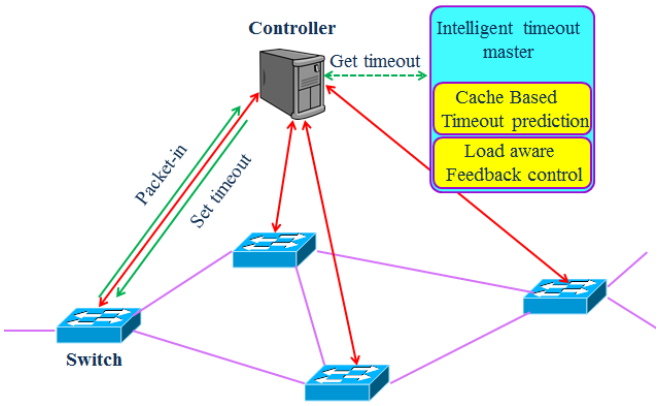


Fig. 2. Intelligent timeout master

A. Cache based timeout prediction

Before proposing the prediction, we first do some analysis about the characteristics of network traffic. Here we analyze real packet traces from university data centers used in [6], whose data set is available to public. In our analysis, packets with the same five-tuple are treated as the same flow.

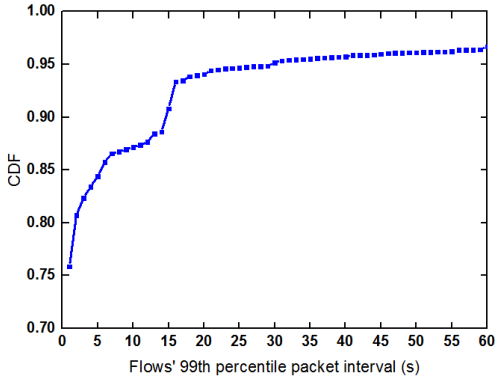
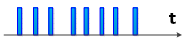


Fig. 3. CDF of different flows' 99th percentile packet interval

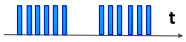
Fig.3 is the CDF (Cumulative Distribution Function) of different flows' 99th percentile packet interval (we sort each flow's packet interval from low to high and take the 99th percentile packet interval value to draw this graph). From the curve we can see that different flows have their own suitable timeouts. Therefore, a fixed timeout for all flows is definitely not a good idea. Without loss of generality, we divide these flows into the following three typical types:

(i) Flow with short packet interval

(ii) Flow with long packet interval



(iii) Flow with periodicity



Furthermore, the CDF curve shows that 76% of the flows are with packet interval less than 1s, which indicates an initial timeout of 1s would be enough for most of flows. For flows with longer packet interval, we use the method in Fig.4 to adjust their timeouts to a suitable value.

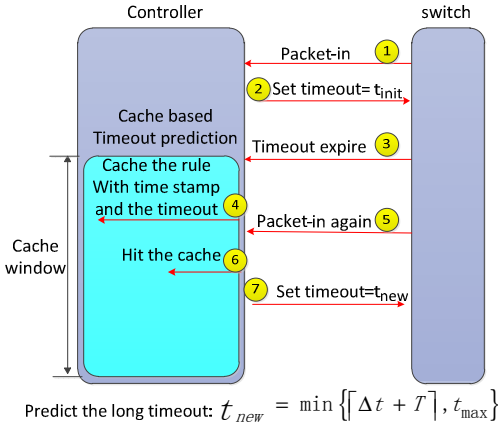


Fig. 4. Cache based timout prediction module

Ideally, we want to assign exactly right timeout to different flows according to their packet intervals. However, this is not an easy task because controllers cannot determine a flow's packet interval on the first sight. To accomplish this task, we come up with a scheme to predict the timeout of a flow entry by adding a cache module on the side of the controller, as shown in Fig.4.

When a new flow arrives, it will generate a packet-in event to the controller. Then, the controller assigns an initial timeout t_{init} to this flow. t_{init} is our initial estimate of timeout value, for trace in Fig.3, it can be set to 1s, since most flows' packet intervals are within 1s. After a flow table entry times out, the switch will remove it from flow table and send the removal message to the controller. Then, the cache module in controller will record this expired rule with a time stamp and its timeout. Once the same flow triggers a packet-in event again, the controller will use the information stored in the cache to determine its new timeout value t_{new} :

$$t_{new} = \min\{\lceil \Delta t + T \rceil, t_{max}\} \qquad (1)$$

Where Δt represents the time interval between the retriggering and the last expire event plus the previous timeout value. T is a constant designed to cover the potential fluctuation of packet interval. t_{max} is the upper bound of t_{new} to prevent it from infinitely increasing. For trace in Fig.3, t_{max} can be set to the value of 11s, since it can cover almost 88% of the flows.

The idea behind this design is quite simple. The retriggering of a flow indicates the initial timeout of t_{init} is not enough, therefore, we set a longer timeout for this flow using the

information in the cache module. Furthermore, the max timeout parameter t_{max} serves as the upper bound of the timeout, which can avoid infinite increasing of timeout as well as limit the amount of information stored in the cache module. Note that the algorithm to determine the new timeout value is heuristic and any other algorithms are acceptable.

B. Load aware feedback control

In order to keep the flow table load at a suitable level, a feedback control is introduced, as outlined in Algorithm1. Here we define two thresholds: FB_{start} and FB_{peak}, as shown in Fig.5. FB_{start} is the threshold beyond which the feedback control starts to take effect. Beneath FB_{start} is the safe area, which means no additional adjustment is needed when TCAM occupation is within this area. FB_{peak} stands for the threshold beyond which the feedback control imposes the most aggressive adjustment to avoid further occupation of TCAM. The area between FB_{start} and FB_{peak} is the warning area. Occupation within this area will trigger feedback control, and the level of feedback control will gradually reach its peak as the occupation approaches FB_{peak}. Beyond FB_{peak} is the reserved area for potential burst situation.

Two main periodic processes are introduced here to accomplish the feedback control. The first one is monitoring process, it will record the TCAM occupation of each switch every t seconds, while the load aware module in controller will only hold the last n observed values of TCAM occupation. The second one is adjustment process, every $n * t$ seconds, it will use a control algorithm to adjust the max timeout value t_{max}, in an effort to keep the TCAM occupation around the FB_{start} threshold. Additionally, to prevent t_{max} from infinitely increasing, we introduce t_{bound} to serve as the upper bound of t_{max}. For efficiency and performance concern, t_{bound} is usually set to the same value as the initial value of t_{max}. Note that the adjustment algorithm is heuristic and any other algorithms can be used.

By adjusting max timeout value according to TCAM occupation, the cache module can react to the load changes in the network and prevent potential flow table overflow.

Algorithm 1: Load Aware Feedback Control Algorithm

1: Data: FB_{start} , FB_{peak}
2: Initially, Old max timeout = t_{max} , $t_{bound} = t_{max}$

3: Monitoring process (every t seconds):
4: **For each** switch:
5: Monitor the TCAM occupation (TCAM OCUP)
6: Record the last n values of TCAM OCUP

7: Adjustment process (every $n * t$ seconds):
8: **For each** switch:
9: calculate the average of the last n TCAM OCUP
10: **if** (TCAM OCUP $\geq FB_{start}$) **then**
11: $k = (\text{TCAM OCUP} - FB_{start})/(FB_{peak} - FB_{start})$
12: $t_{max} = \lfloor \text{Old max timeout} - k * (\text{Old max timeout} - 1) \rfloor$
13: **else**
14: $k = (\text{TCAM OCUP} - FB_{start})/FB_{start}$
15: $t_{max} = \lceil \text{Old max timeout} - k * (\text{Old max timeout}) \rceil$
16: $t_{max} = \max(t_{max}, 1)$
17: $t_{max} = \min(t_{max}, t_{bound})$
18: Old max timeout $= t_{max}$

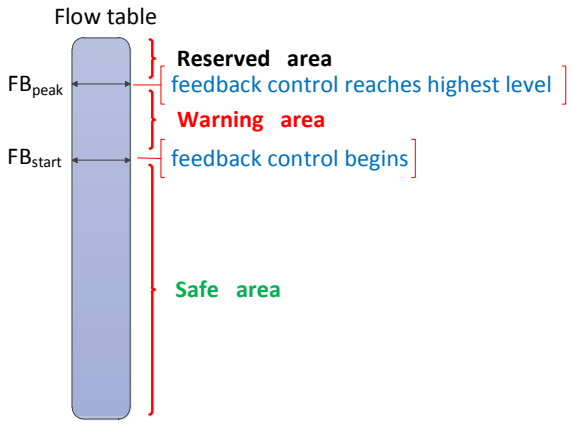


Fig. 5. Load aware module

III. EXPERIMENTAL VALIDATION

We use part of the dataset in [6] to conduct two sets of experiments to evaluate the performance of our Intelligent Timeout Master. In these two experiments, we set the parameter T to 2s.

A. Cache based adaptive experiment

In this experiment, we apply the same trace to both the fixed timeout scheme and adaptive timeout scheme. By comparing the peak TCAM occupation and average packet-in packet per second of each scheme, we find that our adaptive timeout scheme behave much better than the fixed timeout scheme.

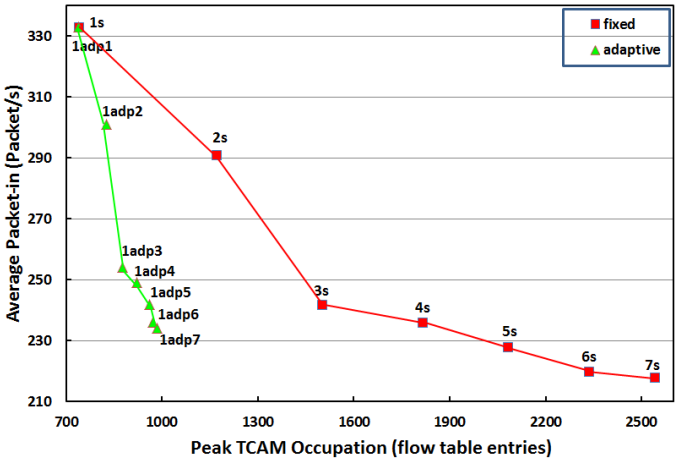


Fig. 6. Comparison of the fixed and adaptive scheme

The result of the experiment is depicted in Fig.6. The green triangle represents our adaptive timeout scheme while the red square stands for the fixed timeout scheme. Each point represents a test case with certain parameter. For fixed timeout scheme, we use different timeout values ranging from 1s to 7s to depict their corresponding peak TCAM occupation and average packet-in packet per second. For adaptive timeout scheme, different max timeout values have been examined (note that 1adp7 in the figure means an initial timeout of 1s with max timeout 7s). From the figure we can discover that the trends of these two curves are similar: the packet-in goes down while the peak TCAM occupation goes up. Furthermore, the

curve of adaptive timeout scheme is always below the curve of fixed timeout scheme, which indicates our adaptive timeout scheme has better performance than the traditional fixed timeout scheme, since it achieves low TCAM occupation as well as less average packet-in simultaneously.

B. Load aware experiment

To verify the performance of our load aware scheme, we compare it with the load ignorance scheme. Here the parameters are set as follows: $t_{init} = 1\text{s}$, $t_{max} = 11\text{s}$, $FB_{start} = 900$, $FB_{peak} = 1300$, table capacity $= 1500$, $t = 1\text{s}$, $n = 10$. We use a trace with changing traffic load to carry out this experiment. The result is depicted in Fig.7. For load ignorance scheme, it works fine at the beginning. However, as time goes by, the traffic becomes heavier, and the load ignorance scheme doesn't have a feedback mechanism to adjust the timeout accordingly, hence leading to flow table overflow. On the contrary, our load aware scheme proactively reduce the value of max timeout when the flow table is about to overflow, which in turn reduces the occupation of flow table and prevent flow table overflow. Furthermore, after the traffic load becomes light again, the recovery mechanism in our algorithm takes effect to increase the max timeout value. Eventually the flow table occupation of load aware scheme comes back to its original level, as shown in the latter part of Fig.7.

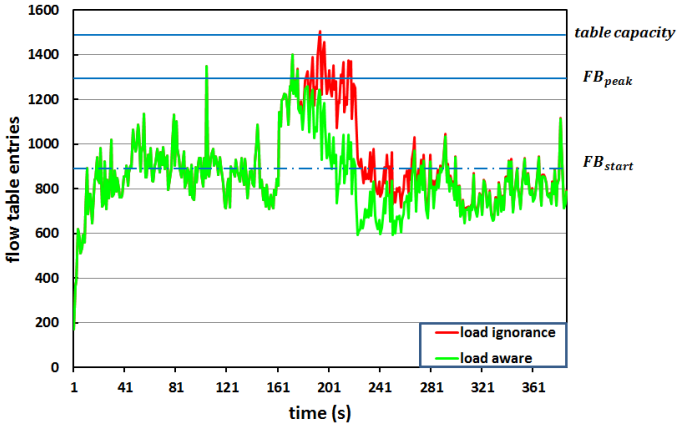


Fig. 7. Comparison of the load ignorance and load aware scheme

IV. RELATED WORK

Nowadays, much attention has been drawn to compress the flow rules in an effort to reduce the TCAM space [7], but the research about assigning dynamic timeout according to flow characteristics and TCAM load is under-explored so far.

Zarek et al. [8] did a set of experiments to explore the effect of idle timeout on both the number of the flow table entries and the miss rate. As a conclusion, he gives some suggested values of timeout for different networks. Another work [9] proposes a way to examine the flow entries already in the TCAM and predict when the flow entries should be evicted. But this method can't be implemented in reality, because the OpenFlow specification doesn't allow altering the timeout of a flow rule already installed in the switch.

Distinguished from the above work, our work designs a more intelligent mechanism. We use a cache to record the expired flow rules to help us make more accurate prediction of real flow timeout. Besides, we introduce a feedback control to automatically adjust the max timeout value according to the current flow table occupation, which is of great importance in preventing flow table overflow.

V. CONCLUSION

In this paper, we propose the Intelligent Timeout Master for OpenFlow to make better use of the limited flow table resources. Firstly, we point out the importance of suitable timeout for each flow as well as the load awareness of flow table. Secondly we develop a mechanism to assign different timeouts to each flow according to its characteristics by adding a cache module in the controller. Thirdly, apart from adaptive timeout according to flow characteristics, we invent a feedback control to dynamically adjust the max timeout value according to the current load of flow table, in an effort to prevent flow table overflow. The results of our experiments demonstrate that the Intelligent Timeout Master performs quite well in both reducing the number of packet-in events as well as reducing the occupation of TCAM.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant No. 61371084, 61433009) as well as the Science and technology project of State Grid Corporation of China on the research of Software defined network system and its key techniques in power applications (2014).

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," SIGCOMM Comput. Commun. Rev.,38(2):69–74, 2008.

[2] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: towards an operating system for networks. ACM SIGCOMM Computer Communication Review 383 (2008).

[3] Floodlight: A Java-based OpenFlow Controller. http://www.projectfloodlight.org/floodlight/.

[4] J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. Curtis, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in SIGCOMM, August 2011.

[5] Open Networking Foundation. OpenFlow switch specication. https://www.opennetworking.org/

[6] T.Benson, A. Akella, and D.Maltz. Network Traffic Characteristics of Data Centers in the Wild. In IMC, 2010.

[7] Xuan-Nam Nguyen, Damien Saucez, Chadi Barakat, Thierry Turletti. Optimizing Rules Placement in OpenFlow Networks: Trading Routing for Better Efficiency. In HotSDN 2014.

[8] A. Zarek, Y. Ganjali, and D. Lie. Openflow timeouts demystified. In MSc Thesis for Department of Computer Science, University of Toronto, 2012

[9] K. Kannan and S. Banerjee. Flowmaster: Early eviction of dead flow on sdn switches. In ICDCN, 2014