SDN Flow Entry Adaptive Timeout Mechanism based on Resource Preference

SDN Flow Entry Adaptive Timeout Mechanism based on Resource Preference

Ziyong Li*, Yuxiang Hu, Xueshuai Zhang

National Digital Switching System Engineering and Technology Research Center, Zhengzhou 450002, China

*Corresponding author's e-mail: 17629352940@163.com

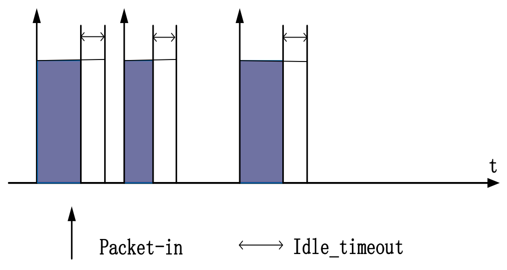
Abstract. Software-defined networking brings rich and more efficient means to network control management. However, the fixed timeout mechanism of flow entry causes the waste of flow table resources and controller computing resources. This paper proposes an SDN flow entry adaptive timeout mechanism. The mechanism allocates a suitable timeout for different flows based on the characteristics of the data flow, and dynamically adjusts idle timeout value to prevent the flow table from overflowing. The experimental results show that the proposed adaptive timeout mechanism can make full use of flow table resources and controller computing resources to achieve higher average network throughput.

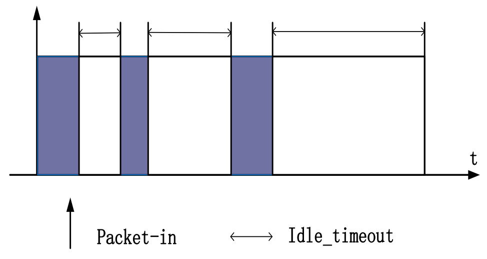
1. Introduction

In recent years, with the continuous enrichment of internet applications and the explosive development of network traffic, SDN has emerged as a new type of network architecture. SDN provides fine-grained data flow control functions through the centralized control plane and open programmable data plane, thus supporting flexible network management and agile service innovation [1].

The control policy of SDN network needs to be installed into the switch flow table. Currently, in order to guarantee good forwarding performance, the flow entries are mostly stored in TCAM (Ternary Content Addressable Memory). However, considering the disadvantages of TCAM technology, such as high cost and high power consumption, the TCAM storage space is limited [2]. The cost per *Mbit* TCAM chip is about 400 times higher than that of RAM chip with the same size. Therefore, the TCAM storage space provided by switches is very limited (supporting *2k~4k* rules [3]), which often cannot meet the needs of large-scale networks. The SDN data plane faces serious shortage of flow table resources.

Limited flow table resources and controller computing resources may cause network scalability problems [4]. Huang *et al.* [5] found that the setting of idle timeout value will affect both types of limited resources. As shown in Fig. 1, when the idle timeout value is set to be too small, the flow table resource can be fully utilized. However, the controller needs to process more *packet-in* messages at this time, and the controller's computing load increases accordingly. When the idle timeout value is set too large, the number of flow requests processed by the controller is reduced, but the flow table resources are not fully utilized at this time, because the redundant flow table rules will occupy the TCAM storage space, causing unnecessary redundancy and overhead. In addition, the mouse flow in the network occupies 80% of the flow, elephant flow only accounts for 20% [6], these large number of mouse flows will inevitably occupy a large number of flow table resources and controller computing resources, so we should dynamically set the idle timeout of the flow entry to make full use of flow table resources and controller computing resources.





(a) Too small timeout (b) Too large timeout

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

Due to the different durations of flows, it is more reasonable to design a dynamically changing timeout threshold. AHTM [7] and TimeoutX [8] proposed that the duration characteristics of flows based on queuing system modeling and actual measurement, and combined the flow table resources and controller computing resource to optimize the hard timeout mechanism. Liang [9] analyzed the idle timeout mechanism of OpenFlow based on the ON/OFF traffic model, and gave the upper and lower bounds of the timeout threshold that meets the resource limit. Huang [10] comprehensively considered the two costs of controller computing resource and flow table resource, and optimized the strategy of whether the new incoming flow is cached and how long the timeout threshold is set.

In this paper, we study the impact of idle timeout on the flow table resource and the controller computing resource qualitatively and how to set the idle timeout value according to the current SDN network resource state. Our goal is to set the reasonable idle timeout value when assigning switch flow table resources to achieve timely deletion of idle flow entries, and to consider controller computing resources and flow table storage resources to increase network average throughput.

2. Models and analysis

2.1 ON/OFF model of data flow

As shown in Fig. 2, a data flow can be modeled as a typical *ON/OFF* model [11]. The *ON* state indicates that the packet length $X_1, X_2, ..., X_n$, and the *OFF* state indicates that the packet interval lengths $Y_1, Y_2, ..., Y_n$. In this model, the *ON* state and the *OFF* state are mutually independent random variables. It can be known from the literature [12] that $X_1, X_2, ..., X_n$ are independent and identically distributed, obeying the Pareto distribution, and the packet intervals $Y_1, Y_2, ..., Y_n$ are equally independent and distributed, obeying the exponential distribution.

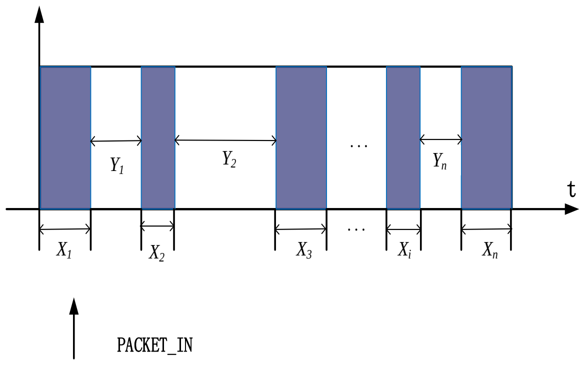


Fig. 2 The ON / OFF model of the data flow

2.2 Quantitative analysis of the impact of idle timeout

2.2.1 Flow Table Resource Overhead

The lifetime of each flow entry can be composed of two parts, including active time and invalid time.

The active lifetime refers to the time period in which the flow entry has a matching data flow. On the contrary, the invalid lifetime indicates the time period in which the flow entry exists but no data flow matches.

The active lifetime is actually the transmission time of the data flow. For the ON state, the active lifetime of the flow entry is $\sum_{i}^{n} X_i$. It can be seen that the active lifetime of the flow table is independent of the setting of idle timeout, and there is no way to change the value by setting idle timeout. When the value of idle timeout is set to *t_{idle}*, the invalid lifetime of the flow entry is derived as Equation (1):

$$\begin{aligned} G(Y,t_{idle}) &= \lim_{n\to\infty}\sum_{i=1}^{n} Y_i\varepsilon(t_{idle}-Y_i)+f(Y,t_{idle})t_{idle} \\ &= \lim_{n\to\infty}\sum_{i=1}^{n} Y_i+(t_{idle}-Y_i)\varepsilon(Y_i-t_{idle}) \end{aligned} \tag{1}$$

The invalid lifetime reflects the invalid usage of the flow table entry for the switch flow table resource, and also reflects the utilization of the flow entry. The smaller the invalid lifetime, the more fully the flow table entry is utilized. According to qualitative analysis, the smaller the idle timeout value, the smaller the invalid lifetime. Therefore, the invalid lifetime of the flow entry is used here to represent the flow table resource overhead. Normalize the flow table resource overhead, the results are as follows:

$$g(Y,t_{idle})=\frac{G(Y,t_{idle})}{\sum_{i=1}^{n} Y_i}=1-e^{-\lambda t_{idle}} \tag{2}$$

2.2.2 Controller computing resource overhead

When the flow entry in the OpenFlow switch is deleted because of $Y_i > t_{idle}$, the subsequent data flow will generate a packet-in message, and the controller needs to process the packet-in message generated and re-add the flow entry to the switch. In fact, the number of packet-in message is a major factor in measuring controller load. Therefore, we use the packet-in message as the controller computing resource overhead. It is assumed that the calculation cost of the controller processing a single packet-in message is a fixed value, so the controller computing resource overhead is described in equation (3). It is normalized to equation (4).

$$Z(Y,t_{idle}) = f(Y,t_{idle}) \cdot \text{cost} \tag{3}$$

$$z(Y,t_{idle})=\lim_{n\to\infty}\frac{Z(Y,t_{idle})}{\cos t\cdot n}=\lim_{n\to\infty}\frac{f(Y,t_{idle})}{n}=e^{-\lambda t_{idle}} \tag{4}$$

2.2.3. Resource preference degree

According to the analysis of the data flow, the elephant flow needs to be divided into multiple data packets to be forwarded, it is more likely that more packet-in messages are generated because the flow entry times out. The mouse flow contains fewer packets, but the number is larger and requires more flow table resources. It can be seen that the elephant flow prefers the controller computing resources, while the mouse flow prefers the flow table resources. Therefore, we define controller computing resource preference and flow table resource preference to reflect the dependence of data flow on controller computing resources and flow table resources.

The average bandwidth of the data flow in the network is represented by *d*, and the number of flow is represented by *r*, and the number of packets a flow contains is *n*. Therefore, the flow table resource preference α and the computing resource preference β can be defined as follows:

$$\alpha = g(\frac{n}{r}),\quad \beta = g(\frac{n}{d\cdot r}) \tag{5}$$

The function *g(.)* represents the resource preference function indicating the relationship between the resource preference degree α , β and the number of data packets *n*, the number of data flow *r*, the average bandwidth of the flow *d*. The default value of the flow table resources preference and the controller computing resource preference is 0.5. It can be seen from the above formula that when the number of flows is large, more flows are requested to obtain flow table resources, and the flow table resource preference of the flows needs to be reduced. When the network traffic is high, the computing resource preference of the flows needs to be reduced.

2.3 flow Entry adaptive timeout problem

In order to get a reasonable idle timeout value of the flow entry, we have the following objective function.

$$c(Y,t_{idle}) = \beta g(Y,t_{idle}) + (1-\beta) z(Y,t_{idle}) \tag{6}$$

It can be seen that our optimization objective is to set reasonable idle timeout t_{idle} to balance the controller computing resource overhead and the flow table resource overhead. α and β are weighted parameters representing the flow table resource preference and computing resource preference calculated according to the current network state.

3. SDN flow Entry adaptive timeout mechanism

3.1. Algorithm Design

The algorithm mainly considers the limitation of controller computing resources and switch flow table resources based on network state and data flow characteristics. Therefore, when the new flow arrives at the switch, the following processing process must be implemented:

(1)It will calculate the flow table resource overhead and the controller computing resource overhead of the flow according to equations (2) and (4).

(2)The controller will update the flow average bandwidth *d* and the number of flow *r*, and obtain the traffic information, and then adjust the flow table resource preference and computing resources preference by the formula (5).

(3)The idle timeout value will be set to balance flow the table resource overhead and the computing resource overhead.

The detailed process of the algorithm is shown in Algorithm 1.

Algorithm 1 flow table adaptive timeout algorithm

```
Input: the average bandwidth of flow: d, the number of flow: r
          the bandwidth requirement: p, the number of packets: n
Output: the idle timeout of flow table entry: $T_{idle}$
1) Initialization parameter: unit resource overhead cost,
     $\alpha = \beta = 0.5$
2) Set the default idle timeout value: $T_{inital}$
3) Repeat
4)       Updating network status information when the data flow arrives.
5)       Calculating the values $x(t,T_{idle})$, $y(t,T_{idle})$ and adjusting the parameters $\alpha, \beta$.
6)     Generating a $T_{idle}$ and updating the idle timeout value of the flow entry.
7) When $c(Y,t_{idle})$ Z reaches a minimum, the algorithm terminates.
```

3.2. Cache based timeout prediction module

It is not an easy task to assign a completely correct timeout for different flows because the controller cannot determine the packet arrival interval. To accomplish this task, we propose a scheme to predict the timeout of a flow entry by adding a cache module in the control plane, as shown in Fig. 3.

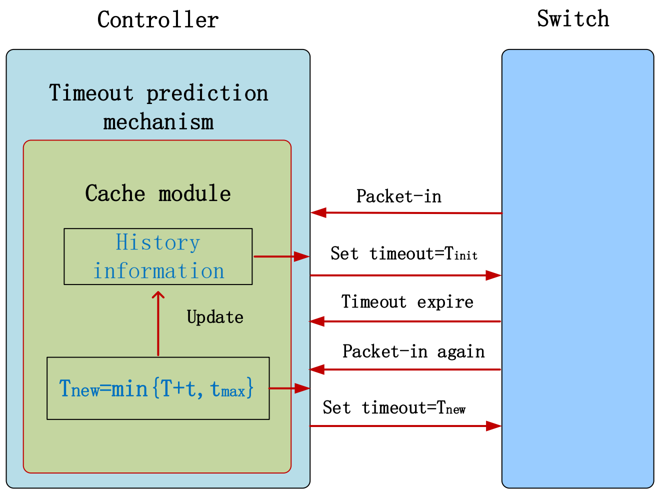


Fig. 3 Cache based timeout prediction module

Its work process is as follows: when the new flow arrives, it will generate a packet-in message to the controller. The controller then assigns an initial idle timeout value to this flow entry. It can be set to 1s because the packet interval for most flows is within 1 s. After the flow table entry beyond idle timeout, the switch removes it from the flow table and sends the delete message to the controller. The cache module in the controller will then record this expiration rule using the timestamp and its idle timeout. Once the same flow triggers the packet-in event again, the controller will use the information stored in the cache to determine its idle timeout value.

$$t_{new} = \min\{T + \Delta t, t_{\max}\} \qquad (7)$$

Δt indicates the time interval at which the flow triggers the Packet-in event again and the last packet ends. T is a constant used to indicate potential fluctuations in the interval between packets. $t_{\max}$ is the maximum value of the idle timeout, preventing it from being set too large and causing the flow table to overflow. The formula (7) indicates that when the flow triggers the Packet-in event again, it indicates that the previously set timeout value is too small, so it is necessary to set a longer timeout value according to the information in the cache module to adapt to the arrival time of the data packet.

4. Simulation results

The simulation selects RYU as the experimental controller and tests on the network simulation platform Mininet. The RYU controller is written in Python and supports multiple versions of the OpenFlow protocol. In order to improve the efficiency of the experiment, the RYU controller and Mininet are deployed as virtual machines on two different physical hosts. The virtual machine is Ubuntu Server 16.04, equipped with Intel Core i7-3770 3.40 GHz processor and 8 GB of memory. In the simulation, the flow entry will only be deleted due to the idle timeout. The number of flow entries in the SDN switch and the number of packet-in messages that the controller needs to process per second can be obtained by sampling.

We compare the two indicators, namely the flow table matching rate, and the number of active flow tables, to the fixed timeout mechanism.

(1) Flow table matching rate

Fig. 4 shows the variation of the flow table matching rate over a period of time. As can be seen from the Fig. 4, the flow matching rate of the adaptive mechanism is always above 80%. Because the adaptive mechanism can dynamically adjust the idle timeout value. It can adjust the resource preference factor through the feedback mechanism to set a reasonable idle timeout value, speeding up the elimination of flow entries with low matching rate, thereby improving the utilization of flow table resources.

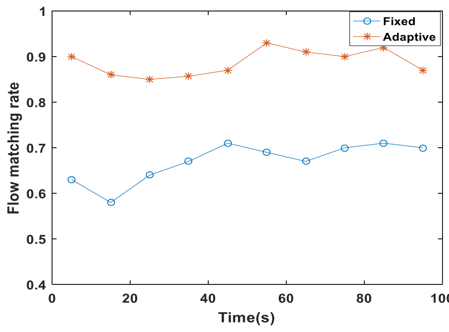


Fig. 4 Flow matching rate

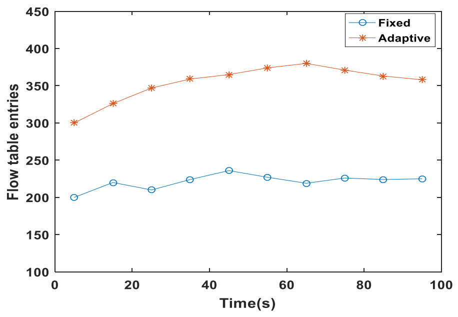


Fig. 5 Number of active flow entries

(2) Number of active flow entries

The Fig. 5 shows the number of active flow entries over a period of time. It can be seen that there are always more than half of the idle flow entries in the fixed mechanism, which seriously wastes the flow table space and affects the performance of the switch. The active flow table entries of the adaptive mechanism is fully utilized, and the performance of the switch is fully utilized.

5. Conclusions

In this paper, an adaptive flow entry timeout mechanism based on resource preference is proposed to balance the limited flow table resource and controller computing resource. To set a reasonable idle timeout value of the flow entry, we introduce the concept of flow table resource overhead, controller computing resource overhead, and resource preference according to the current network state and data flow characteristics. In addition, we design the flow Entry adaptive timeout mechanism. Our mechanism can set a different idle timeout for different flows according to flow characteristics to prevent the flow table from overflowing. The experimental results show that the proposed flow entry adaptive timeout mechanism can make full use of flow table resources and controller computing resources to achieve higher network average throughput.

Acknowledgments

This work is supported by the National Natural Science Foundation of China for Innovative Research Groups (61521003), the National Natural Science Foundation of China (61872382), the National Key Research and Development Program of China (2017YFB0803204), and the Research and Development Program in Key Areas of Guangdong Province (No.2018B010113001).

Reference

[1] Karakus M, Durresi A. A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN)[J]. Computer Networks, 2016, 112:279-293.
[2] Liu A X, Gouda M G. Complete redundancy removal for packet classifiers in TCAMs[J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(4): 424-437.
[3] Stephens B, Cox A, Felter W, et al. PAST:scalable ethernet for data centers[C]. International Conference on Emerging Networking Experiments and Technologies (CoNEXT). ACM, 2012:49-60.
[4] Tootoonchian A, Gorbunov S, Ganjali Y, et al. On Controller Performance in Software-defined Networks[C]// Usenix Conference on Hot Topics in Management of Internet. USENIX Association, 2012.
[5] Huang H , Guo S, Li P , et al. Cost Minimization for Rule Caching in Software Defined Networking[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(4):1007-1016.
[6] Lan K C, Heidemann J. A measurement study of correlations of Internet flow characteristics[J]. Computer Networks, 2006, 50(1):46-62.
[7] Zhang L, Lin R, Xu S, et al. AHTM: Achieving efficient flow table utilization in Software

Defined Networks[C]. IEEE Global Communications Conference. IEEE, 2014:1897-1902.
[8] Zhang L, Wang S, Xu S, et al. TimeoutX: An Adaptive Flow Table Management Method in Software Defined Networks[C]. IEEE Global Communications Conference. IEEE, 2015:1-6.
[9] Liang H, Hong P, Li J, et al. Effective idle_timeout value for instant messaging in Software Defined Networks[C]. IEEE International Conference on Communication Workshop. IEEE, 2015:352-356.
[10] Huang H, Guo S, Li P, Liang W, Albert Y. Zomaya. Cost minimization for rule caching in software defined networking[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(4):1007-1016.
[11] Understanding the Nature of Social Mobile Instant Messaging in Cellular Networks[J]. IEEE Communications Letters, 2014, 18(3):389-392.
[12] Study on GEANT4 code applications to dose calculation using imaging data[J]. Journal of the Korean Physical Society, 2015, 67(1):195-198.