An Efficient Flow Table Management Of Flow Entry In Openflow Switch/Router

S Veeramani, B Indira, M Pandi and T Sivakumar

Abstract— In OpenFlow switch, a limited number of row entries can be stored in the flow table, to the maximum of its capacity. Once its maximum capacity is reached, one has to remove one of the entries to accommodate the new entry. This paper proposes a mechanism to remove the entry from the flow table of OpenFlow switch, which uses the combination of less frequently used and least recently used. Removed entries are stored in a separate cache memory. Since OpenFlow switch has multiple row table, any one of the tables can be utilized for this approach. Instead of approaching the controller for every new packet, the entry can be looked up in the cache table. For inserting any new entry into the flow table, it will be searched in the flow table. If it is not found in the flow table called as table miss. Then it can be searched in the cache table. The experimental results show that the proposed approach gives an average performance improvement (in terms of table miss) of 13.87% and 13.88% against LRU and LFU. This approach reduces the frequent interaction between OpenFlow switch and the controller. The experimental results also show that proposed approach reduces the controller interaction by 17% and 15% more than LRU and LFU approaches.

Index Terms— OpenFlow, Controller, FIFO, LRU, LFU.

1 INTRODUCTION

OPENFlow network [1] evolved in order to reduce the complexity of current networking devices like Switches/ Routers. OpenFlow decouples data and control planes and there will be one centralized controller which monitors all the activities of the network. Since the controller can communicate directly with the flow table of each device as shown in Fig. 1, it is easy to insert or delete an entry from the centralized controller. The Switch/Router can act as a forwarding engine, which will forward the packet to the next hop according to the instruction given by the controller. Since a single controller is monitoring the activities of the entire network, scalability needs to be improved in OpenFlow network. Scalability can be improved by adding more number of controllers or reducing the interaction time of the switch and the controller. In this paper, an efficient row entry management scheme has been proposed to reduce the interaction time of the switch and the controller. Since for each entry in the flow, the table is associated with static time out, an entry can be removed automatically irrespective of its usage pattern. Though the same row can be referenced in the near future, it will not be available in the flow table. When a new row arrives at the switch it will check the flow table if an entry is not available, which is called table miss, and then it will be directed to the controller for the new entry update. OpenFlow switch has limited memory capacity. The switch cannot accommodate a number of entries more than its maximum capacity. To add a new row entry, the switch must remove existing entries to make room for the new row. When there is a number of table misses, the performance of network drastically reduces. The objective of this paper is to reduce the number of table misses in the flow table. This approach uses one of the flow tables from multiple row table [2] for implementation. The rest of the paper is organized as follows. Section 2 list out various notations used, Section 3 describes the background by addressing the challenges. Section 4 brief about related works. Section 5 introduces our proposed algorithm. Section 6 compares the performance of the proposed approach with existing techniques and followed by conclusion in Section 7.

- Dr S Veeramani is currently working as Assistant Professor(SS) in Dept. of CSE at Dr.MCET, Pollachi. E-mail: veeramanicse@gmail.com
- Ms. B Indira is currently working as Assistant Professor in Dept. of CSE at PSR Engineering College Sivakasi. E-mail: aridnib@gmail.com
- Dr M Pandi is currently working as Assistant Professor(SG) in Dept. of CSE at Dr.MCET, Pollachi. E-Mail:mpandi123@gmail.com
- DrT Sivakumar is currently working as Associate Professor in Dept. of CSE at Dr.MCET, Pollachi. E-mail: ts@drmcet.ac.in

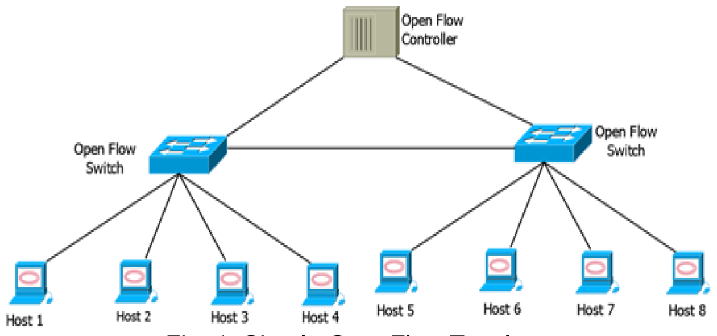

Fig. 1. Simple OpenFlow Topology

2 NOTATIONS AND DEFINITIONS

[1] Table_Miss: A flow entry is not available in the row table of OpenFlow switch.
[2] Table_Hit: A flow entry is available in the flow table of OpenFlow switch.
[3] Interaction: When there is a table_miss, the switch has to interact with the controller for new rule installation.
[4] Table_Miss rate: Number of table_misses when inserting a flow into the flow table.
[5] Table_Hit rate: Number of table_hits when inserting a row into the flow table.

3 BACKGROUND

In order to manage row entries in the flow table, a flow entry can be removed using random or time stamp or frequency based policies. Fig 2 shows that when there is no place to add new row in the flow table, then the controller randomly replaces the entry in the flow table using row mod instruction. Assume the flow entry 168:0:1:4 to be added in the flow table. Since there is no place to insert into the flow table, then the controller randomly selects a third row entry to replace with a new one.

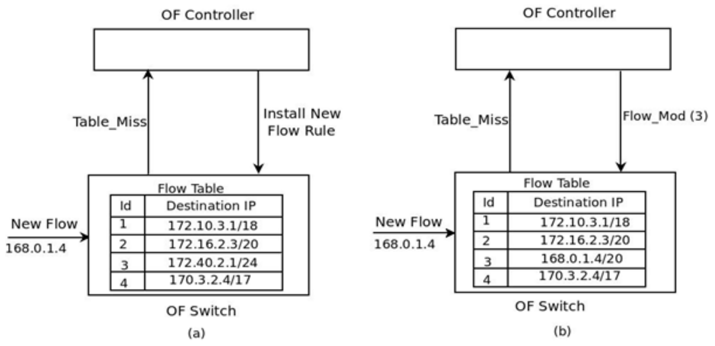

Fig. 2. The controller replaces an entry using a random approach

(1) FIFO (First In First Out): The entry entered first into the flow table can be removed first in order to make room for the new entry. This approach will not be suitable for real-time applications, because when the number of flows increases, table miss also increases. To insert a new row entry 168:0:1:4 in the flow table, the controller selects the first entered flow 170:3:2:4/17 for the replacement as shown in Fig. 3.

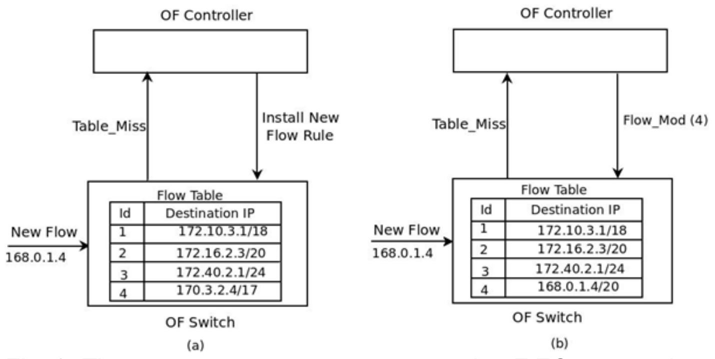

Fig. 3. The controller replaces an entry using FIFO approach

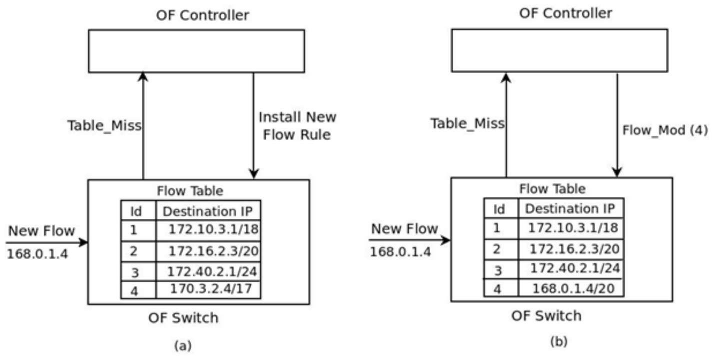

Fig. 4. The controller replaces an entry using the LRU approach

(2) LRU (Least recently Used): In this approach, an entry is removed based on the time stamp. This approach removes an entry, based on recent usage, but it does not consider the frequency of the flow. To insert a new row entry 168:0:1:4 into the flow table, the controller selects the entry which is used least recently (170:3:2:4/17) as shown in Fig. 4.

(3) LFU (Least Frequently Used): In this approach, an entry is removed based on frequency. Though it removes entries based on the frequency of usage, it does not give importance to recency of flow entry. To insert a new row entry 168:0:1:4 into the flow table, the controller selects the entry which is less frequently used (170:10:3:1/18) as shown in Fig. 5.

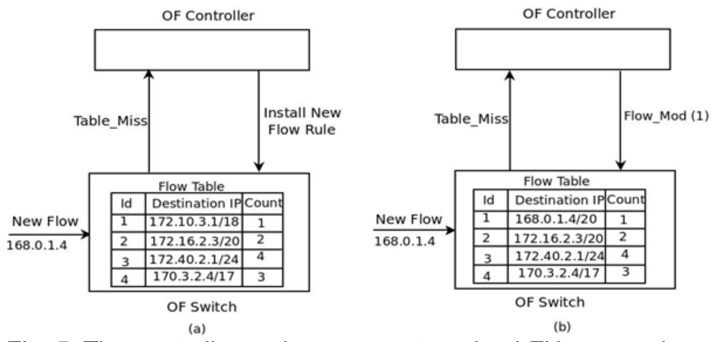

Fig. 5. The controller replaces an entry using LFU approach

4 Related Works

A mechanism using LRU cache, which keeps the flow entry as many as possible, is proposed in [3] [4]. When the flow entry is not available in the switch row table, then the switch sends packet in message to the controller. As many numbers of switches send the packet in messages to the controller, the controller has to process all such requests. Another approach which keeps entries that are matters for the management is proposed in [5]. This approach keeps necessary entries in the flow table based on the previous traffic pattern. The traffic pattern is predicted based on the flow inter-arrival time and adjusting the timeout value of each row in advance is proposed in [6]. This paper also shows that interaction time reduces by 9%. The traffic traces from the different scenario and shows the trade-off between table sizes. Installation of missed row entry is proposed in [7]. A method called Intelligent Timeout Master which assigns time out for each of the flow, according to their flow characteristics is proposed in [8]. This paper also uses feedback control to adjust the timeout for each row, according to its occupation. Only the selective packets are redirected through the intermediate switches is proposed in [9]. The analysis of real data center packet traces and design a heuristic approach based on some key observations is proposed in [10]. Based on the observation, many flows in the network never repeat and assigns a smaller minimum timeout value for row rules. The report [11] describe replacement policies like random, FIFO and LRU. They have compared these approaches on various row table sizes. The management of row entry in the flow table is proposed in [12]. This paper does not discuss the crucial factor of controller interaction time. The author has proposed various lookup operation performed in the Open Flow switch using trie based data structure [16]. In order to reduce the number of entries in the flow table entries virtual compression techniques is proposed in [17].

5 Proposed Flow Table Approach

In an OpenFlow network, the controller can interact directly with the switch flow table. The fields of the flow table are match field, counter, timeout, action. Since these row tables can store a limited set of row entries, the switch will be able to store entries based on its maximum capacity. These row entries can be modified directly through the controller. Moreover, the flow entry from the table can be deleted automatically once the timeout expires. If the same row arrives at the switch (table misses), it has to send to the controller for an update. If multiple switches, row entries approaching the controller at the same time, it leads to the scalability problem. Since the frequent interaction of controller with multiple switches degrades the performance of the entire network. An efficient mechanism is required to keep only the relevant entry and remove the irrelevant entry from the flow table. In this paper, instead of deleting an entry from the flow table, it can be kept active for a certain duration in a separate row table, as shown in Fig. 6.

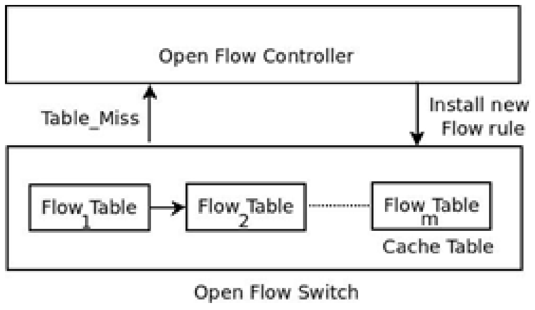

Fig. 6. Overview of Proposed Approach

Assume that the flow table can accommodate 'k' number of entries {e_1, e_2, … e_k}. If a new entry say ek+1 arrives at the OpenFlow switch, then the switch cannot store the entry since it can store only 'k' number of entries in the flow table. In order to accommodate new row entry ek+1 in the flow table, the controller should remove any one of the flow entry randomly from the flow table. Then the controller replaces the entry with Flow_Mod instruction. The drawback of replacing an entry randomly is that the removed entry can be used in the near future or it can be used very frequently. To overcome this problem, the controller either can use least recently used (LRU) or less frequently used (LFU) approach to remove the entries from the row table. In LRU approach, it keeps track of the time when it is accessed. Assume each entry in the row table is associated with a time stamp {t_1, t_2,… t_k}. If the controller wants to remove some entry, then it can remove the entry which has accessed least recently i.e., least ({t_1, t_2,… t_k}). With LFU approach, every entry is associated with a counter so-called frequency which stores how many times those entries are accessed. Assume each entry in the flow table associated with frequency {f1, f2…fk}. If the controller wants to remove some entry, then it can remove the entry which has been accessed least frequently i.e., least ({f_1, f_2,… f_k}). Since OpenFlow switch has multiple row table, any one of the tables can be used to store the deleted entries. The selected row table can act as cache table and it can be partitioned into two halves. The upper portion stores the entry which is least recently used and lower portion stores the entry which is less frequently used. Table 1 consists of row entries which are captured from Wireshark Open packet analyzer [13]. If both the halves are completely filled, certain entries from the bottom of the upper portion and certain entries from the top of the lower portion will be removed, so that the newer entries can be accommodated. When a new row entry arrives at the switch, it will perform a lookup in the first row table. If the entry is not present, it is called table miss. Then it continues to look up in the second row table and so on. Since the last row table acts as a cache table it will have deleted entries from multiple row table. The entry can be found in this table, which will reduce the interaction time of switches and the controller. Algorithm 1 is implemented in the cache table to manage flow entries efficiently. When a new row arrives at the switch, initially it will be added to the upper portion of the flow table. Suppose, if the same row arrives again, then it will be pushed down to the lower half. So, the flow table can be adjusted according to the flow table. Assume that the cache table has K entries in it. Both the upper and lower pointer points to K/2-1 or K/2+1 locations and the top most and bottom most entries from upper and lower portion of the flow table will be removed once the limit exceeds. Hence, the maximum row table size can be utilized according to the capacity. Assume that upper portion of the cache table is filled, then the entry will be moved from the upper portion to the lower portion. So, the new entry can be added from the bottom to the top of the upper portion and the known entry can be inserted from the top to the bottom of the lower portion. Removal of an entry will be at the bottom most of the upper portion or at the top most of the lower portion. Consider that a new entry number 149 from Table 1 is to be added at the very first time at the table. Then it can be added at the top most of the upper portion according to the Algorithm 1. If the entry is known to the flow table, then it will be shifted to the bottom most of the lower portion. If both the upper and lower portion gets filled, then some of the entries from the bottom of the upper portion and the top of the lower portion will be removed automatically. By doing the so, the flow entries are adjusted to the maximum size. Algorithm 2 computes number of misses and hits from the flow table when number of requests are added in the flow table.

Algorithm 1 Proposed Flow Table Management Algorithm

```
Require: Assume row table has K entries and divide the table into two
equal half with pointer upper_ptr, lower_ptr
Ensure: Accommodate all the flows in accordance with row table size
Insert null value at K/2 location of the table
Let n denote available number of flow entries in the flow table
Let ft_table denote maximum row table size
n=0
while n <=ft_table do
if Flow entry is new then
Insert into upper portion of the table at the index (K/2 -1)
Keep the time stamp for each entry
Decrement upper_ptr by 1
n = n + 1
else if Flow entry is known then
Push the flow entry from upper portion to lower portion of the table at the
index (K/2 + 1)
count the frequency of each entry inserted in the lower portion
Increment lower_ptr by 1
end if
end while
if n > ft_table then
Delete an entry from the top of the lower portion
Move all the entries one position up in the lower portion
Delete an entry from the bottom of the upper portion say 'e'
Move all the entries one position down in the upper portion
Insert new row entry at upper_ptr location
Push down the entry 'e' at lower_ptr location
end if
```

Algorithm 2 Calculation of table miss and table hit

```
Require: Assume row table with K number of entries
Require: f denotes row entry to be matched
Ensure: Check the flow entry f in the flow table
table_miss = 0
table_hit = 0
n = 0
while n <=ft_table do
    if f present in the flow table then
        table_hit = table_hit + 1
    else if f is not present then
        table_miss = table_miss + 1
        n = n + 1
    end if
end while
```

Let m=Number of table_miss and h=Number of table_hit.

$$\text{table\_miss\_rate} = \frac{m}{m+h} \tag{1}$$

The table misses rate is calculated using Equation (1).
In a similar manner, the table_hit rate is calculated using Equation (2).
The percentage of reduction in interaction between the controller and the switch is computed from the Equation (3).

$$\text{table\_hit rate} = 100 - \text{table\_miss\_rate} \tag{2}$$

$$\%\ \text{of interaction} = \frac{\text{table\_hit\_rate}}{\text{number of entries to be stored}} \tag{3}$$

Consider the flow table which is filled completely as shown in Fig. 7(a). Assume a new row entry to be inserted in the flow table be 'j'. To insert this entry, a new room should be created at both the upper and lower portion. Because the entry removed from upper portion is pushed into the lower portion. As shown in Fig. 7(b), the top of the lower portion entry (f) is removed and the entry removed from the bottom of the upper portion is shown in Fig. 7(c). The new entry 'j' is inserted at the location upper_ptr and the entry removed (d) from upper portion is pushed into the location lower_ptr as shown in Fig. 7(d).

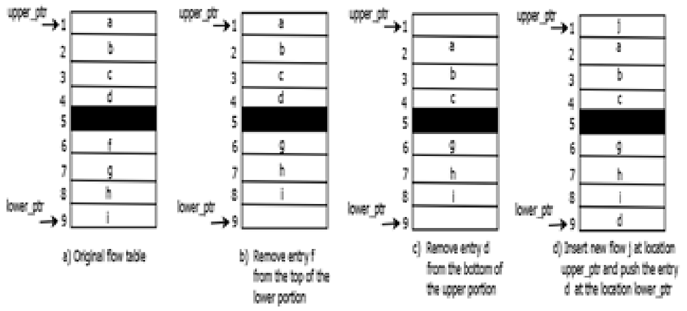
upper_ptr
lower_ptr
a) Original flow table
b) Remove entry f from the top of the lower portion
c) Remove entry d from the bottom of the upper portion
d) Insert new flow j at location upper_ptr and push the entry d at the location lower_ptr

Fig. 7. Illustration of row table when both portion of the table filled

6 Results and Discussion

The proposed mechanism uses Mininet [14] and Wireshark [13] which are open source tools for OpenFlow simulations. The topology is generated using Mininet and it can be configured with the controller called Floodlight [15]. Flows are generated using Wireshark from the different switches to the controllers. The current OpenFlow switch will store only 5K to 6K row entries in the table which are costly. The controllers in the current networks become heavily overloaded due to frequent updating of row entries and this will limit the ability of controller. The flows are generated at various time duration (670.585 sec and 142.058 sec) as shown in Figure 8 and 9. The proposed mechanism is compared with existing LRU and LFU policies with various row tables sizes ranging from 100 to 1000 and a various number of row entries ranging from 1000 to 4000. Fig. 10, 11, 12, 13 shows comparison of table miss with respect to various row table sizes (100 to 1000) and various row entries (1000 to 4000). Fig. 14, 15, 16, 17 shows a comparison of percentage of improvement of the proposed approach with existing approaches with respect to various row table sizes (100 to 1000) and various row entries (1000 to 4000). Fig. 18, 19, 20, 21 shows a comparison of percentage of reduction of controller interaction of the proposed approach against existing approaches with respect to various row table sizes (100 to 1000) and various row entries (1000 to 4000). The upper portion keeps tracking the order of row entered in order to know which entry to be removed when it is full. It uses linked list to store in the order in which the row has entered and to quickly access in constant time O(1). Similarly, the lower portion maintains two linked list; one to keep the frequency and another one is to keep a list of row that has the same frequency. The head node of the linked list has a set of row that has a larger frequency. The tail node will have the list of row that has the least frequency. Hashing is used to access the row based on frequency as key in constant time (O(1)). The proposed approach also uses the time complexity O(1) to access or delete. Assume that the row table can accommodate n number of row entries, then the space complexity for all the approaches will be O(n).

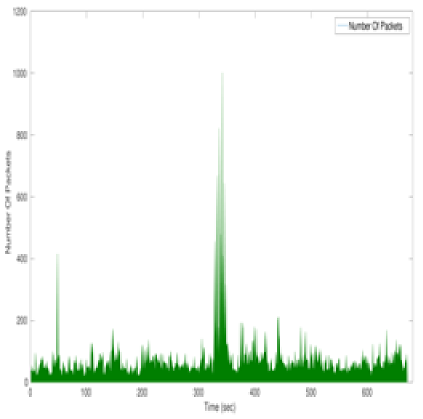

Fig. 8. Figure traffic generated from wireshark (670.585 sec)

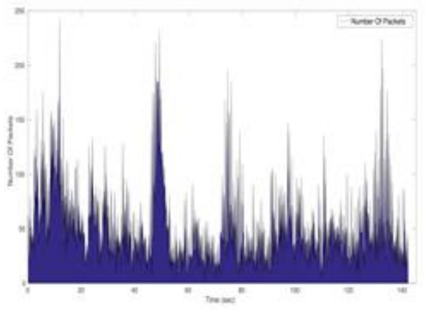

Fig. 9. Figure traffic generated from wireshark (142.058 sec)

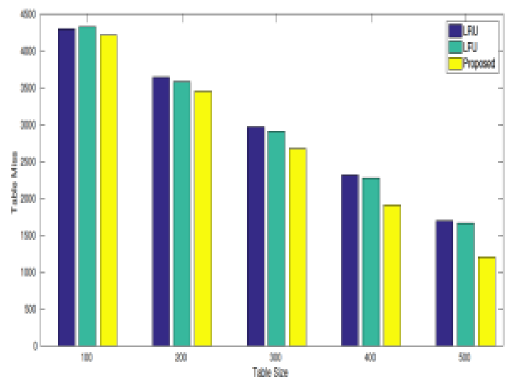

Fig. 10. Figure Number of entries to be accommodated: 1000

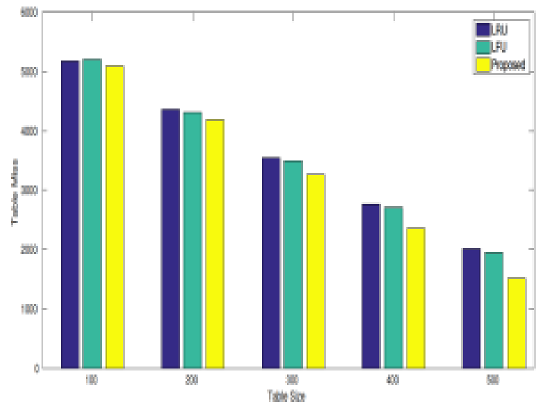

Fig. 11. Figure Number of entries to be accommodated: 2000

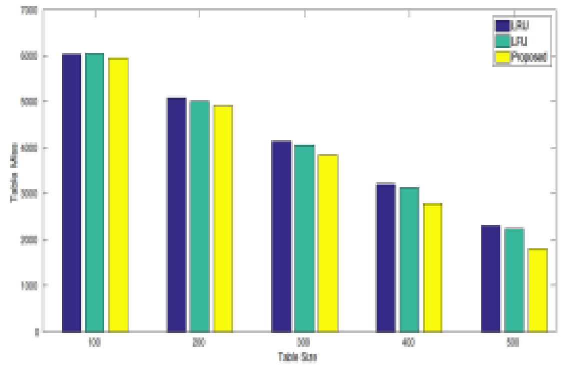

Fig. 12. Figure Number of entries to be accommodated: 3000

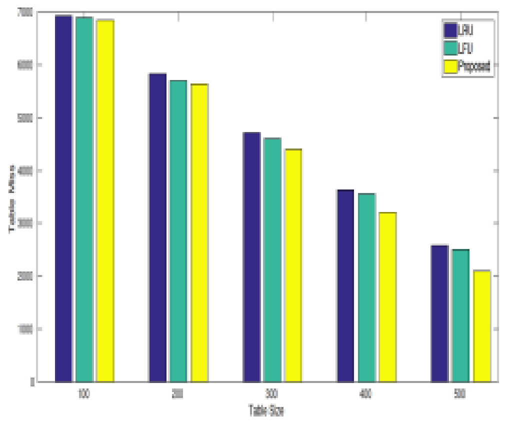

Fig. 13. Figure Number of entries to be accommodated: 4000

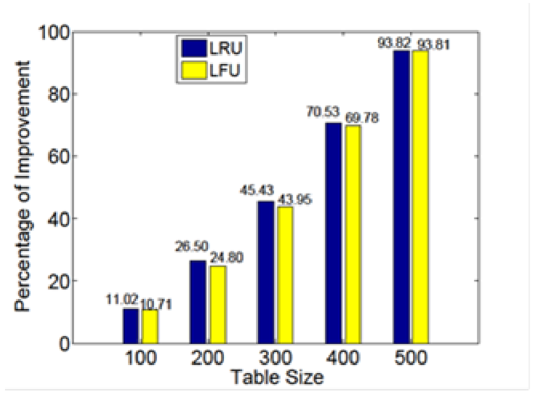

Fig. 14. Figure % of improvement when number of entries to be accommodated: 1000

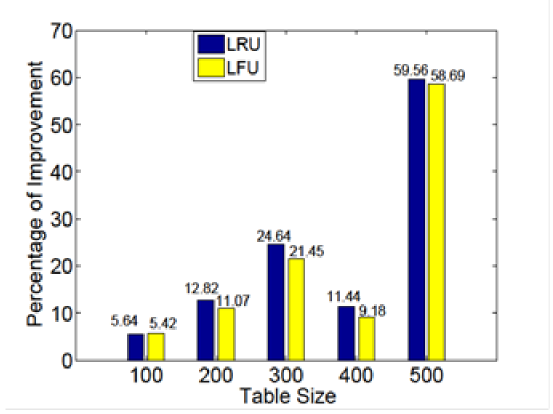

Fig. 15. Figure % of improvement when number of entries to be accommodated: 2000

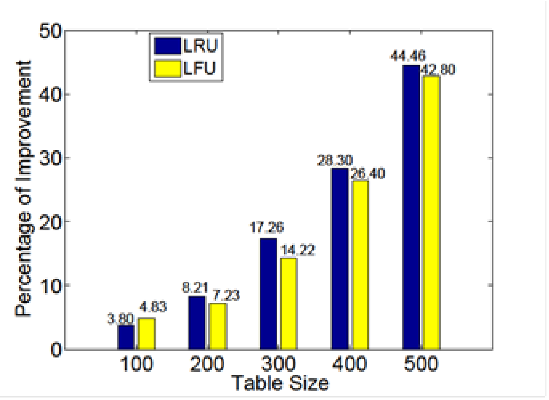

Fig. 16. Figure % of improvement when number of entries to be accommodated: 3000

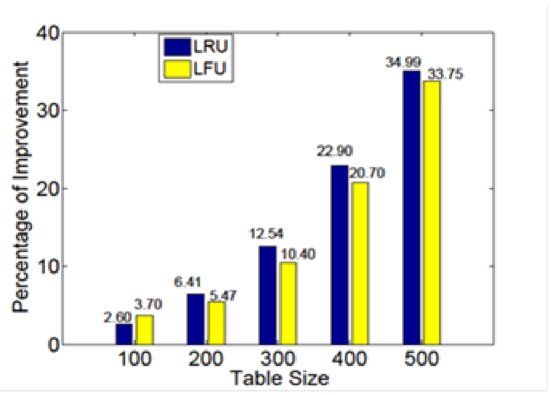

Fig. 17. Figure % of improvement when number of entries to be accommodated: 4000

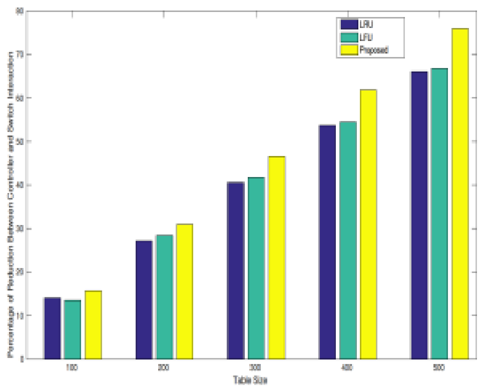

Fig. 18. Figure % of reduction in controller interaction when number of entries to be accommodated:1000

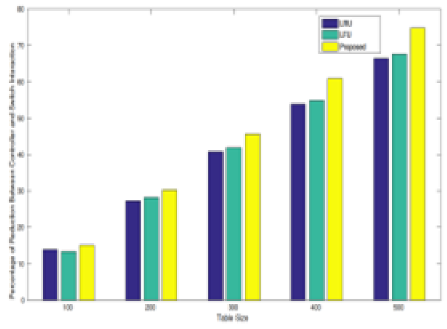

Fig. 19. Figure % of reduction in controller interaction when number of entries to be accommodated: 2000

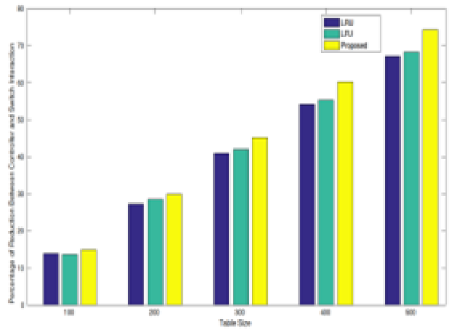

Fig. 20. Figure % of reduction in controller interaction when number of entries to be accommodated: 3000

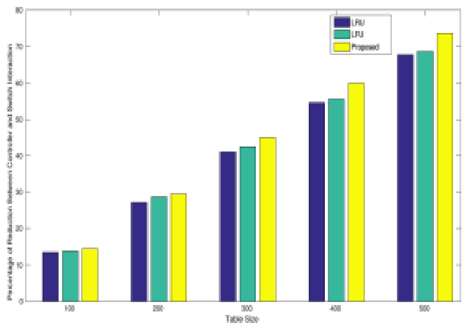

Fig. 21. Figure % of reduction in controller interaction when number of entries to be accommodated: 4000

7 Conclusion

The proposed approach reduces the interaction time of the switch and the controller by efficiently managing the flow entry in the flow table. Since OpenFlow switch suffers from limited storage capacity instead of removing the flow entry randomly from the flow table, the proposed approach keeps those entries which are based on both timestamp and its frequency. So that the entry which can be used in the near future is kept in the cache table to avoid the interaction time of the switch and the controller. By doing so, the maximum size of the flow table utilized properly and table misses can be reduced. On an average, the proposed approach shows 13.87% and 13.88% improvement in (terms of table miss) than existing LRU and LFU approaches. Similarly, the reduction in controller interaction by 17% and 15% more than LRU and LFU approaches.

Acknowledgment

The authors wish to thank anonymous reviewers for spending their valuable time to shape our article.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openow: Enabling Innovation In Campus Networks", ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, 2008.
[2] "Openow Switch Specification version 1.3.4 0 implemented wire protocol 0x04", 2014.
[3] E.-D. Kim, S.-I. Lee, Y. Choi, M.-K. Shin, and H.-J. Kim, "A flow entry management scheme for reducing controller overhead", 16th International Conference on Advanced Communication Technology (ICACT), pp. 754-757, 2014.
[4] E.-D. Kim, Y. Choi, S.-I. Lee, M.-K. Shin, and H.-J. Kim, "Flow table management scheme applying an LRU caching algorithm", International Conference on Information and Communication Technology Convergence (ICTC), pp. 335-340, 2014.
[5] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks", ACM SIGCOMM Computer Communication Review, vol. 41, no. 4, pp. 254-265, 2011.
[6] 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.
[7] M. Dusi, R. Bifulco, F. Gringoli, and F. Schneider, "Reactive logic in Software-Defined Networking: Measuring ow-table requirements", International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 340-345, 2014.
[8] H. Zhu, H. Fan, X. Luo, and Y. Jin, "Intelligent timeout m aster: Dynamic timeout for SDN-based data centers", IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 734-737, 2015.
[9] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE", ACM SIGCOMM Computer Communication Review, vol. 41, no. 4, pp. 351-362, 2011.

[10] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in OpenFlow networks", Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, pp. 177-188, 2014.
[11] A. Zarek, Y. Ganjali, and D. Lie, "OpenFlow timeouts demystified", Univ. of Toronto, Toronto, Ontario, Canada, 2012.
[12] S. Veeramani, S. R. Sharma, and S. N. Mahammad, "Constructing scalable hierarchical switched OpenFlow network using adaptive replacement of flow table management", IEEE International Conference on Advanced Networks and Telecommunication Systems (ANTS), pp. 1-3, 2013.
[13] https://www.wireshark.org/download.html/
[14] http://mininet.org/
[15] http://www.projectoodlight.org/oodlight/
[16] S.Veeramani and Noor Mahammad, "Efficient IP Lookup Using Hybrid Trie-Based Partition-ing of TCAM-Based OpenFlow Switches", Springer Photonic Network Communication, Vol.28, no. 2, PP. 135-145, 2014.
[17] Veeramani and Noor Mahammad, "Minimization of ow table for TCAM based OpenFlow switches by virtual compression approach", In Proceedings of IEEE International Conference on Advanced Networks and Telecommunication Systems (ANTS), pp. 1-4, 2013.