UDP Flow Entry Eviction Strategy Using Q-Learning in Software Defined Networking

Hanhimnara Choi
Department of Artificial Intelligence
Sungkyunkwan University
Suwon, Korea
hanhimy@skku.edu

Syed M. Raza
Department of Electrical and Computer Engineering
Sungkyunkwan University
Suwon, Korea
s.moh.raza@skku.edu

Moonseong Kim
Department of Liberal Arts
Seoul Theological University
Bucheon, Korea
moonseong@stu.ac.kr

Hyunseung Choo
Department of Electrical and Computer Engineering
Sungkyunkwan University
Suwon, Korea
choo@skku.edu

Abstract*— Software-defined networking provides a programmable and flexible way to manage the network by separating and centralizing the control plane. The data plane entities like software-defined switches and routers use flow entries in flow tables for forwarding the packets. However, the limited switch memory restricts the number of flow entries in the flow tables. This leads to flow table overflow and flow entry reinstallation problems, which severely degrade the network performance. This requires a comprehensive policy for timely eviction of inactive flow entries to avoid overflows and optimally maintain flow tables usage. To this end, many studies have been proposed, but none of them have suggested detailed eviction strategy for UDP flows. This paper proposes a UDP flow eviction strategy which periodically updates the statistical information of UDP flows through reinforcement learning and utilizes it to evict inactive UDP flows. This eviction strategy is combined with the existing TCP flow eviction method to form an eviction system that takes into account the protocol-specific characteristics of the flow. Through three traffic-based experiments, we found that the proposed system reduces the number of overflow occurrences by 27% and flow entries reinstallation by 28%, compared to the random and FIFO policies, resulting in 15% reduction in control signaling overhead.

Keywords—Software-defined networking, Reinforcement learning, Q-learning, OpenFlow

I. INTRODUCTION

Software-defined networking (SDN) is an emerging networking paradigm that separates data plane from control plane and centralizes control functions on a logically centralized SDN controller [1]. The separation of control and data planes in SDN enables programmable network applications and flexible network management through a standardized communication between controller and software-defined data plane entities (*i.e.*, switches and routers). The switches handle packets approaching the network through one or more flow tables within each switch. A flow table is a set of flow entries that contains information about how incoming flow packets are matched and what actions are performed on them [2]. The controller adds, deletes, or updates flow entries in switches by exchanging standard protocol messages, and consequently the approaching packets are forwarded after matching with the corresponding flow entry.

Switches usually use a Ternary Content-Addressable Memory (TCAM) [3] due to its high speed but suffers from its limited capacity, and this prevents the flow table from accommodating sufficient flow entries. This leads to flow table overflow, which means that flow entry for incoming flows cannot be added. Resultantly, switch fails to match packets of the incoming flows to any entry (a.k.a. table miss). In the case of a table miss, communication between the controller and the switch is inevitable to handle that packet, and the network control overhead increases significantly. In addition, flow table overflow can be exploited to attack the network by crimpling the switches capability to admission more flows [4]. To avoid network degradation due to flow table overflow, flow entries of inactive flows must be timely evicted.

An eviction policy that determines which flow entry is to be evicted has significant impact on network performance. For example, the following may happen. Flows can be divided into long-lasting elephant flows with large payload and mice flows with inverse properties. If a flow entry of a continuing elephant flow is evicted to install a new mice flow entry, the elephant flow will request the reinstallation of flow entry. The reinstallation of flow may cause an unexpected delay, recurrence of control overhead, and overflows [5]. Furthermore, there is a risk that unexpected termination of the connection results in TCP window shrinkage, leading to overall degradation of network performance [6]. Therefore, it is important to proactively manage the number of installed flow entries to prevent overflow, and define a comprehensive policy to determine which flow entries must be evicted before and after overflow.

Flow table management is a well-studied research topic, and multiple recent studies have presented effective flow eviction policies for flow table management. These studies can be largely divided into two groups. There are studies that use traditional algorithms such as FIFO and random selection to determine the next flow entry to be evicted in the event of overflow. Then there

This work is partly supported by the Ministry of Education, IITP, and NRF, Korea, under the High-Potential Individuals Global Training Program (IITP-2019-0-01579), AI Graduate School Support Program (No.2019-0-00421), and mid-career support program (NRF-2020R1A2C2008447).

are studies that apply supervised machine learning algorithms to classify flow entries as active or inactive, or maintain optimal usage of flow tables through Reinforcement Learning (RL).

Characteristics of TCP can be exploited to form an effective early eviction policy for TCP flows [7]. In contrast, UDP flows offer no such information which can be used to create an early eviction policy for them. To the best of our knowledge, a single study has handled UDP flows through a simple delayed entry installation policy [7]. In this paper, we argue that UDP is an important protocol for future networks and can take more diverse forms in its use. In particular, QUIC, a new UDP-based protocol by Google, is adopted as HTTP3 standard [8]. This calls for a comprehensive and efficient flow entry management policy to manage UDP and TCP flows collectively.

This paper proposes an early eviction mechanism for UDP flows and combines it with existing mechanism for TCP flows to create a comprehensive flow table management system. Challenge in designing an eviction strategy for UDP flows is to determine the completion of UDP flows, as there is no explicit flow termination signaling in the UDP protocol. To this end, we propose a RL-based dynamic monitoring mechanism in SDN controller that periodically collects statistical information of UDP flows from the switches and evicts the inactive UDP flow entries as earliest as possible. To minimize the control overhead due to periodic collection, RL learns the characteristics of the flow and dynamically adjusts the monitoring period. In the emulated experiment environment, OpenFlow is used as standard communication protocol between the controller and the switches. Experimental results show that the proposed early eviction mechanism of UDP flows reduces the number of overflows by 27%, and reduces the reinstallations of flow entries by 28% in comparison to FIFO and random eviction policies, implying that the proposed mechanism adequately selects the flow entries of UDP flows for eviction.

II. Background And Related Work

A. SDN Flow Table Management and Its Limitations

SDN enables program-based centralized network control plane with well-defined APIs by separating it from the data plane. SDN controller in the control plane communicates with data plane through standardized protocols. One of those protocols is OpenFlow which is well established and most commonly used in industry and academia. Switches in the data plane handle packets approaching the network through one or more defined flow tables within them. A flow table can consist of a number of flow entries that specify rules for handling and managing flows. A flow entry consists of match fields, instruction, counters, etc., Fig. 1 shows the detailed structure. Match fields defines matching criterion for the flow packets, instruction defines how matched packets are to be handled, and counters records the statistical information of the flow.

Through OpenFlow, SDN controller installs flow entries in the flow tables of the switches to define the rules for forwarding

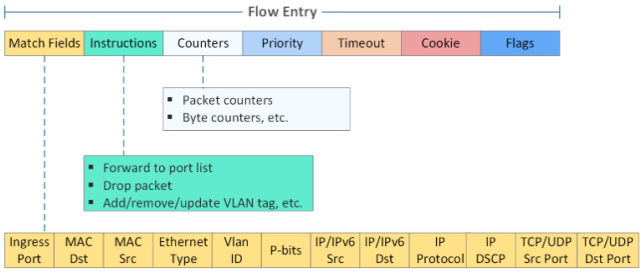

Fig. 1. Structure of a flow entry in a switch flow table.

incoming packets belong to the flows. In particular, when an incoming packet of a flow fails to match any flow entry in the flow table, the switch forwards it to the SDN controller for further processing by using OpenFlow Packet_In message. SDN controller defines a forwarding rule for remaining packets of the flow by adding a flow entry in the switch flow table through Flow_mod message. Update or removal of a flow entry is also done through Flow_mod message with additional parameters.

Other than Flow_mod message, there are two methods for automatic removal of a flow entry; hard timeout and idle timeout. In the case of hard timeout, a timeout value for a flow entry is specified at the time of its installation. After expiration of the hard timeout, the flow entry is automatically removed, despite that corresponding flow is still active or there is enough capacity in the flow table. Consequently, hard timeout eviction policy causes sharp increase in reinstallations of flow entries. Idle timeout of a flow entry defines a period in which if no packet of the corresponding flow arrives, the flow entry is expired. This either leads to overflow due to high idle timeout value, or to high number of reinstallations due to low idle timeout value. Hard and idle timeout flow entry eviction methods in OpenFlow are inefficient for flow table management, and there is a need for more intricate mechanism which can accurately and timely detect the inactivation of flow entries to execute the eviction.

One of the ways to design an efficient flow table management system is to exploit the characteristics of different protocols. In particular, explicit session termination characteristic of TCP flows can be used to determine the inactivation instant of the corresponding flow entries. Similar or any other method is not applicable for UDP flows, as it is designed to be a lightweight and simple protocol. Hence, continuous tracking of UDP flow entries is required for timely eviction. This elevates the control signaling cost due to exchange of status request and response messages between switches and SDN controller after every monitoring period. Increasing the monitoring period to reduce the control signaling cost effects the timely detection of flow entry inactivation, and reducing the monitoring period increases the cost. Hence, the monitoring period policy needs to be carefully crafted.

B. Selected Related Studies in Literature

Early studies on flow entry eviction explore the algorithms like FIFO and random from cache replacement strategies and

apply them for flow table management when overflow occurs [9]. In FIFO, the oldest flow entry in the flow table is selected for eviction, and any flow entry is randomly selected in the case of random. Trigger for eviction is provided by the occurrence of overflow. Later, same algorithms are applied to proactively avoid the occurrence of overflow [10], and in this case eviction trigger is based on a certain threshold value over flow table capacity. Another study uses session termination of TCP flows through FIN/RST flags for flow entry selection and eviction trigger [7].

Recently emerging machine learning techniques are frequently used to improve different aspects of network management. RL and supervised learning algorithms are equally used in various studies to handle different issues related to networks. These algorithms are also used in various ways for solving different aspects of flow table management. For instance, supervised learning techniques such as classification and regression can be used to solve flow table management problems with the given raw data. An earlier study classifies flows as active or inactive using supervised learning and evicts the inactive flows in case of flow table overflow [12]. Another study borrows the similar idea but extends the previous work by proactively evicting the inactive flow entries through predictive method [13]. Performances of these studies are dependent on quality and quantity of data for offline learning, and it is not always available. Moreover, performance can be degraded if the online traffic pattern deviates from the patterns in the learned data.

RL is used to overcome the data dependency and offline learning limitations. Unlike supervised learning, RL does not require offline learning through dataset. Instead, RL maximizes cumulative rewards by learning what behavior learning agent should choose while interacting with dynamic systems. Investigation of various RL algorithms for optimal flow table usage is presented in [11]. However, pre-installation of flow entries must precede the arrival of actual flows [11], and causes unnecessary flow table occupation. Therefore, we propose a RL based scheme focusing on UDP flows which timely evicts the flow entries of completed flows to adjust the incoming flows.

III. Proactive Eviction of UDP Flows

In this paper, we propose an early eviction strategy for UDP flows using RL. Furthermore, the proposed UDP eviction method is combined with an existing TCP flow eviction mechanism [7] to establish a comprehensive flow table management system.

A. Proactive Flow Eviction Mechanism

Lack of explicit connection termination in UDP flows makes it hard to detect the inactivation of corresponding flow entries for timely eviction. However, in SDN, the controller can require a switch to provide statistical information about any of the installed flow entries. The statistical information includes duration, idle time, and the number of matched packets of the flow. This paper uses these stats at the controller to identify an approximate trend of UDP flows and evict them if the trend is towards inactivation. For this purpose, the controller sends a flow stat request to a switch for the packet_count of corresponding flow entries of UDP flows after every sampling period. The reply message from the switch includes packet_count of all the flow entries of UDP flows, and the controller performs necessary tasks on each entry.

The way in which controller processes every entry in the received flow stat reply message is shown in Fig. 2. The first step is to check if packet_count for the entry from previous sampling period already exists in the record or not. In case there is no previous information, a new field for the flow entry is created in the record and its packet_count is stored. In case the packet_count from the previous sampling period exists in the record, packet_count difference between current and previous sampling periods is taken. A positive difference value indicates that the flow is active, and its current packet_count is updated in the record. Zero difference value shows that the flow is inactive, and controller evicts the corresponding flow entry from its record and from the switch by sending an eviction message. This process is executed for each entry in the response message after every sampling period to avoid overflow by proactively evicting inactive flow entries of UDP flows.

Furthermore, the proposed system handles the flow table overflow by performing priority based flow entry eviction. The priority of flow entry of UDP flows is determined based on stats recorded by the controller. The rate of increase in packet_count indicates how much the packet_count has increased compared to the previous sampling period. The flow entry with highest rate of increase is assigned with highest priority and the flow entry with lowest rate is assigned lowest priority. At the occurrence of overflow when an installation of a new flow entry is required, the proposed system evicts the flow entry with the lowest priority to minimize the probability of reinstallation.

Effectiveness of the proposed system is dependent on the continuous and timely update of the flow entry stats, and to achieve this, sampling period needs to be appropriately short. However, number or traffic rate of UDP flows is not always high in the network. Small sampling period causes unnecessary control overhead when traffic rate of UDP flows is low. In addition to the control overhead, the sampling period also effects the performance of the proposed system. If the period is too short, an entry of an active UDP flow may be evicted early, causing table miss for later coming packets of the flow. In other words, this results in the process of sending Packet_In messages to the controller for reinstallation of the flow entry. In contrast, if the period is too long, an entry of an inactive flow that already has no packets to match occupies space in the flow table for a long time and causes wastage of valuable flow table space. To have an effective sampling period with minimal overhead, we design an adaptive periodic sampling method that works on the SDN controller. In this method, sampling period for collecting UDP flows stats from a switch is continuously adjusted according to the network environment using RL.

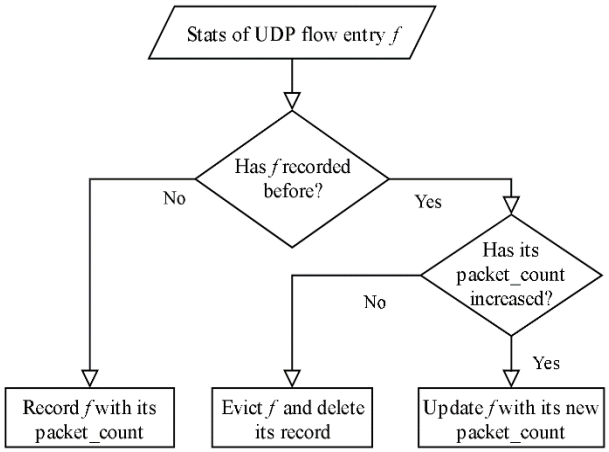

Fig. 2. Logic of proactive UDP flow deletion

B. Design of Learning Framework

In the proposed system, RL is used to dynamically resize the sampling period, so the period value becomes the most critical parameter in the learning framework. The goal of learning framework is to increase the usage of flow tables from a long-term perspective by minimizing the number of overflow occurrences and the reinstallation of previously installed flow entries. Thus, the proposed learning framework is designed to maximize table hit rates without any terminal state. Any RL framework consists of three elements: states, actions, and rewards. The learning agent receives a given state of the environment, it takes the relevant action and transfers to another state, thereby receiving rewards to maximize long-term cumulative rewards. In the proposed system, the state, action, and reward spaces of the RL framework are defined as follows.

- States: The state space of the proposed model represents the size of the sampling period, where the unit size is 500 ms. It ranges from 500 ms to 5,000 ms with total of 10 states. The condition space of the specified sampling period may be indicated as $s_i = (sampling\_period_i)$
- Actions: The action space of the proposed model includes three options. (i) increase the sampling period by unit size; (ii) decrease the sampling period by unit size; and (iii) maintain the sampling period. For example, actions that increase the sampling period can be expressed as $a_i^{increase} = (sampling\_period_{increase})$
- Rewards: The flow table hit rate is measured after the elapse of time T since the last action taken by the learning agent. The calculation of table hit rate excludes the table misses that occur at the initial arrival of the flows, as they are inevitable. Rewards given to learning agents are based on this measured percentage of table hits. Three kinds of rewards are given. If the measured hit rate is higher than the hit rate before the action, assign a reward of 1, and assign reward of -1 if it is low. 0 reward is assigned if there is no change in the hit rate. At a time t, reward can be expressed as follows:

$$r_t = \begin{cases} 1, & hit\_ratio_{t-1} < hit\_ratio_t \\ 0, & hit\_ratio_{t-1} = hit\_ratio_t \\ -1, & hit\_ratio_{t-1} > hit\_ratio_t \end{cases}$$

Based on the above-mentioned learning framework design, the proposed system uses the temporal difference learning algorithm, Q-learning. The reason for using the Q-learning algorithm is that it updates the Q value of each action at a given time step, regardless of the model for the environment [14]. The proposed learning agent interacts with the environment and repetitively performs learning process every T second. In each step, the agent selects the action to take in the given state s_t. The agent selects actions on its own based on an action-selection policy aimed at maximizing the expected rewards. Once the action is selected, reward is given accordingly, and the following state determines the sampling period for the next T second. Finally, the Q value is updated according to the action taken by the agent, and the above processes are repeated for each step. Q value of action a in state s is expressed as $Q(s, a)$, and it is updated with the following formula after rewards are collected.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma\, maxQ(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

where α ($0 < \alpha \leq 1$) is the learning rate, which determines how much information the agent obtains overrides the existing information. γ ($0 \leq \gamma \leq 1$) is the discount rate that defines the importance of future rewards and ensures the convergence of the cumulative rewards received by agents performing continuous tasks. In this paper, α and γ parameters are set to 0.1 and 0.9, respectively, with agent learning period (T) as 6 s.

Algorithm 1: Q-Learning algorithm

Pre-condition:

Initialize Q table with small random number

Initialize state s_t

Procedure:

1: **for** every T seconds **do**
2: Get action a_t from s_t using ε-*greedy* policy
3: **if** $\varepsilon > \varepsilon_{min}$
4: $\varepsilon = \varepsilon$ *(1 – decay rate)*
5: Take action a_t on the s_t and receive reward r_t,
6: Sample new state s_{t+1} after applied action a_t
7: Update $Q_t \leftarrow Q_t + \alpha(r_{t+1} + \gamma maxQ_{t+1} - Q_t)$
8: **end for**

The Q-Learning algorithm uses the epsilon greedy policy as an action-selection policy. It selects action with the highest Q value as the next action by default, but action can also be selected randomly with a constant probability. This allows the algorithm to have a chance to explore the action space during the process of finding actions that can maximize rewards and prevent them from falling into the local loop. The probability of randomly selecting an action is continuously reduced as the learning process repeats, as shown in Algorithm 1. This decay process lasts until ε reaches the preset extension rate, ε_{min}, after which random actions are selected with a fixed probability. We have set ε_{min} to 0.1, and decay rate to 0.995.

IV. Evaluation Results and analysis

A. Implementation

The proposed UDP flows eviction mechanism is emulated by using Mininet [15], a Python-based SDN framework, and Ryu [16] SDN controller. The basic network control operations and learning agent are defined through Python script in the Ryu controller. Emulated network uses OVS switches v2.5.5, and OpenFlow v1.3 as standard interface protocol between the controller and the swtiches.

The RL-based UDP flow eviction mechanism in SDN controller learns from five different type of messages received from the switches. Packet_In from a switch is used to determine the arrival of a new flow or an ongoing flow due to early eviction of its corresponding flow entry. Flow_Stats_Reply is a response to the controller's flow information request to update the table, and Table_Stats_Reply is used by learning agent to select an action and calculate the corresponding reward. Eviction of a flow entry is confirmed through received Flow_Removed message, and flow table overflow is detected through Error_Tablefull message.

We have conducted experiments on a network topology consisting of a controller, a switch, and 30 hosts. A single switch topology is chosen to have fair comparison against target methods, as overall performance of the proposed system only increases with more switches in the network. Extension of the proposed system for multiple switches is easily possible through a separate record and Q-table for each switch. In order to emulate flow table overflows, the flow table size in the switch is set to accommodate up to 50 flow entries. All the hosts in the network act as both client and server to facilitate traffic generation for performance evaluation.

Experiments use flow-based traffic generated on each host by using iperf. Three different traffic types are used in the experiments for evaluation purposes, where each type generates total of 1,500 flows. Traffic types differ from each other based on different ratios of TCP and UDP flows. For example, traffic type T3U7 consists of 30% TCP flows and 70% UDP flows. Configurations of each traffic type are detailed in Table I. In each type, total 1,500 flows are sequentially generated, where interval between generated flows is exponentially distributed. The ratio of elephant flows (*i.e.*, long duration flows with large payload) and mice flows (*i.e.*, small duration flows with small payload) in all the traffic types is set to 5% and 95%, respectively. The average size of elephant flows is set to about 30 MB with packet size ~30 KB, and average size of mice flows is set to about 300 KB with packet size ~1,470 B. For an experiment, traffic consisting of elephant and mice flows of TCP and UDP protocols is generated without any specific order.

TABLE I. Simulation traffic configuration

Traffic	T3U7	T5U5	T7U3
Total flows	1.5K	1.5K	1.5K
TCP rate	30%	50%	70%
UDP rate	70%	50%	30%

B. Experimental Results

The proposed UDP flow eviction system is evaluated using three performance metrics, namely overflow occurrences, flow reinstallations, and aggregate control overhead. As the goals and strategy of existing RL-based flow entry management scheme [11] are in contrast with the proposed eviction mechanism, it is difficult to make a fair comparison based on above mentioned performance metrics. Therefore, we compare the proposed eviction mechanism with baseline proactive eviction schemes (Random and FIFO) to evaluate its performance. Random and FIFO are generally reactive eviction schemes, where flows are evicted only in case of overflow occurrence. To have fair comparison with proposed eviction system, random and FIFO schemes are modified to proactive flow evictions. It is achieved by implementing a threshold on flow table capacity, and both target schemes evict flows as per their policies once the threshold is crossed.

The flow table overflow happens regardless of the proactive eviction system in place, due to higher flow arrival rate than sampling period. An effective flow eviction system cannot prevent overflow occurrences but can significantly reduce them by selecting inactive flow entries to be evicted. Normalized number of overflow occurrences for all the three eviction methods with different traffic types are show in the Fig. 3. Normalized value for a particular method with a certain traffic type is obtained by taking a ratio between an overflow occurrence in the method and aggregate overflow occurrences for all methods. Results in Fig. 3, show that the proposed eviction system significantly reduces overflow occurrences in comparison to both random and FIFO eviction schemes. The lowest overflow occurrence is achieved by the proposed eviction system for the case of traffic type with 70% UDP flows. This highlights the fact that the proposed eviction mechanism aggressively evicts the flow entries of UDP flows to create space in the flow table for incoming flows. FIFO eviction schemes caused the most overflows, because it evicts entries of active elephant flows without taking into account any attributes of the flows. Almost similar results for random and FIFO eviction schemes in the case of traffic type T5U5 can be attributed to random selection probabilities of random selection scheme.

Reinstallation of a flow entry happens when the in place eviction mechanism removes a flow entry of an active flow. Resultantly, table miss occurs for the following packet of the flow, and it is sent to the controller in the form of Packet_In message. This not only increases the control signaling overhead and controller load, but also effects the QoS of the flow. Hence, the reinstallation performance metric represents the significance of flow entry selection for eviction, where lower reinstallations implies better selection mechanism. Fig. 4 illustrates the normalized number of reinstallations performed by each eviction scheme for different traffic types. Normalized values for reinstallations are calculated through similar method as overflow occurrences.

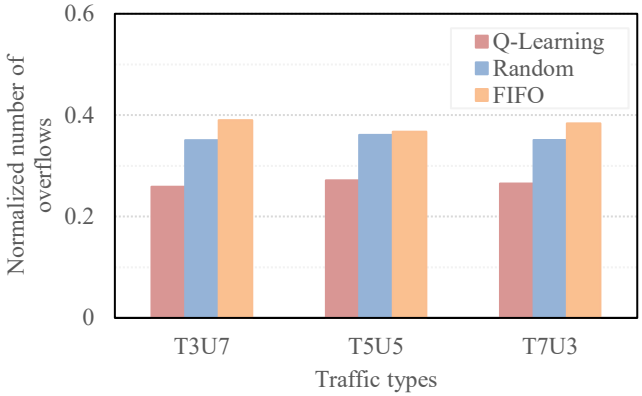

Fig. 3. Comparison of overflow occurrences with different traffic types.

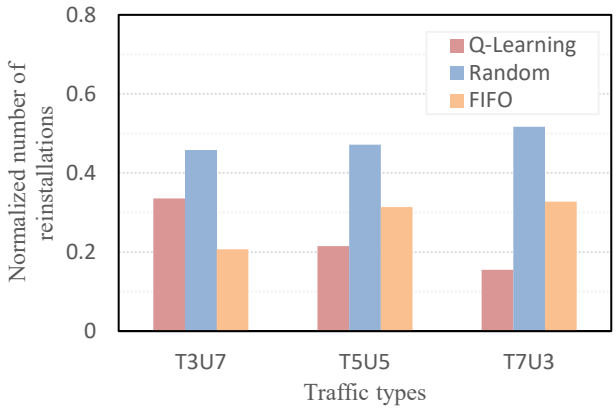

Fig. 4. Flow entries reinstallations comparison between proposed and target eviction mechanisms.

The results in Fig. 4 shows that FIFO scheme outperforms the proposed eviction system only in the case of T3U7 traffic type. This is because FIFO is dependent on the order of flows in which they enter the network. If the beginning of T3U7 traffic mostly consists of mice flows, then FIFO correctly selects flow entries of inactive flows almost every time. Conversely, if the beginning of T3U7 traffic mostly consists of elephant flows then almost every time flow entry selected by FIFO is for active flow. For this reason, FIFO based eviction policy has un-deterministic performance in arbitrary network traffic. Whereas the selection mechanism of proposed eviction system is agnostic to network traffic conditions. Moreover, the proposed eviction system benefits from RL characteristic to maximizes the total sum of rewards, including rewards to be gained later, by choosing actions in consideration of the future. This improves the performance of learning agent over time which can reduce reinstallations and increase system performance from a long-term perspective.

An eviction mechanism with significant overhead is not practical to be used in real networks regardless of its performance. Hence, control overhead is an important performance metric. Control overhead of an eviction mechanism is defined as total number of OpenFlow messages exchanged between switch and the controller due to overflows and reinstallations. This also includes the Flow_Stats request

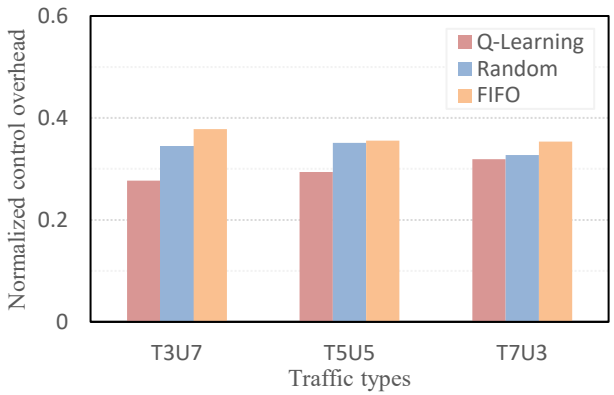

Fig. 5. Control overhead comparison with different traffic types.

and response messages for flow stats collection purpose in the proposed eviction system. We have measured and normalized the control overhead of proposed eviction system, random, and FIFO for three traffic types, and the presented results in Fig. 5 show that the proposed eviction system has the lowest control overhead. This is an expected result based on the outcomes of overflow and flow reinstallation experiments, as the proposed mechanism has the lowest overflow occurrences and flow reinstallations. More importantly, results in Fig. 5 reveals that RL based adaptive sampling period in the proposed system reduces the otherwise high collection overhead. Its applicability in real networks is advocated by the fact that even with the added stats collection overhead, its cumulative overhead is lower than random and FIFO for all three traffic types.

V. Conclusion

This paper presents a proactive flow entry eviction mechanism for UDP flows in SDN. It is achieved by sampling stats of UDP flows, where the sampling period is adaptively controlled using RL. The eviction decision on a flow entry is made based on the packet count difference in current and previous sampling period. We have combined the proposed eviction mechanism with the TCP flow entries eviction in [7] to have a comprehensive eviction system for SDN flow table management. As there are no explicit studies on UDP flow eviction strategies available in literature, the performance of the proposed eviction system is evaluated against conventional random and FIFO target schemes. The preliminary results show that the proposed system reduces the overflow occurrences and flow entries reinstallation by 27% and 28%, respectively. Control overhead analysis reveals that even with stats collection the number of control messages generated by the proposed system are 15% lower than the two target schemes. We are further evaluating the proposed eviction system against different metrics with real traffic traces, and will present them in our next article. Moreover, other characteristics of UDP traffic are under investigation that can be added to improve the RL state space.

Acknowledgment

The third and fourth authors are the co-corresponding authors.

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” in the SIGCOMM Computer Communication Review (CCR) 2008.

[2] “OpenFlow Switch Specification Version 1.3.0”, Open Networking Foundation 2012.

[3] TCAMs and Openflow: What every practitioner must know. [Online]. Available:https://www.sdxcentral.com/articles/contributed/sdn-openflow-tcam-need-to-know/2012/07/, 2012.

[4] J. Leng, Y. Zhou, J. Zhang, and C. Hu, “An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flowtable overflow in software-defined network,” CoRR, vol. abs/1504.03095, 2015.

[5] M. Ku´zniar, P. Pereˇs´ıni, and D. Kosti´c, “What you need to know about sdn flow tables,” in International Conference on Passive and Active Network Measurement. Springer, 2015, pp. 347–359.

[6] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, “Star: Preventing flow-table overflow in software-defined networks,” Computer Networks, 2017.

[7] S. Shirali-Shahreza and Y. Ganjali, "Delayed Installation and Expedited Eviction: An Alternative Approach to Reduce Flow Table Occupancy in SDN Switches," in *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1547-1561, Aug. 2018.

[8] Bishop, Mike. "Hypertext transfer protocol version 3 (HTTP/3) ." *Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-20* (2019).

[9] A. Zarek, Y. Ganjali, and D. Lie, “Openflow timeouts demystified,” Univ. of Toronto, Toronto, Ontario, Canada, 2012.

[10] 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, ser. DEBS ’14. New York, NY, USA: ACM, 2014, pp. 177– 188. [Online]. Available: http://doi.acm.org/10.1145/2611286.2611301.

[11] Ting-Yu Mu, Ala Al-Fuqaha, Khaled Shuaib, Farag M. Sallabi, and Junaid Qadir, “SDN Flow Entry Management Using Reinforcement Learning,” in *ACM Transactions on Autonomous and Adaptive Systems,* 13, 2, Article 11, 23 pages, Nov. 2018.

[12] H. Yang and G. F. Riley, "Machine Learning Based Flow Entry Eviction for OpenFlow Switches," 2018 27th International Conference on Computer Communication and Networks (ICCCN), 2018, pp. 1-8.

[13] H. Yang and G. F. Riley, "Machine Learning Based Proactive Flow Entry Deletion for OpenFlow," *2018 IEEE International Conference on Communications (ICC)*, Kansas City, MO, 2018, pp. 1-6.

[14] Watkins, Christopher JCH, and Peter Dayan, "Q-learning," *Machine learning* 8.3-4 (1992): 279-292.

[15] Mininet – an instant virtual network on your laptop (or other PC). [Online]. Available: http://www.mininet.org/, February 2017.

[16] Ryu SDN Framework, [Online]. Avaialable: https://github.com/osrg/ryu.