Maximizing SDN Flow Table Efficiency with Dynamic Timeout Allocation and Proactive Eviction

Begüm Arslan, Elif Kızılkaya, Beytullah Yiğit, Fatih Alagöz
Dept. of Computer Engineering, Bogazici University, Istanbul, Turkey
Email: {begum.arslan@std, beytullah.yigit@std, elif.kizilkaya@std, fatih.alagoz@}bogazici.edu.tr

2024 15th International Conference on Network of the Future (NoF) | 979-8-3503-7776-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/NoF62948.2024.10741357

***Abstract*—Software-Defined Networking (SDN) represents a transformative paradigm that decouples the data plane from the control plane in network architecture. Through the utilization of an OpenFlow controller, this framework populates flow tables with necessary flow entries. Flow tables are inherently constrained by factors such as power consumption and cost, which underscores the need for adept management algorithms to optimize their utilization. In this study, we introduce *VITARPEM* utilizing two approaches: dynamic idle timeout allocation and proactive eviction. By incorporating dynamic idle timeout allocation, contingent upon flow table occupancy, we have achieved a reduction in packet-in count of up to %8.5 compared to the latest research findings. Moreover, our proactive eviction algorithm has demonstrated a diminished rate of rejected flows, thereby augmenting overall network performance. In scenarios characterized by limited flow table sizes, the number of rejected flows decreased to %43 of the closest competitor's count.**

***Index Terms*—SDN, TCAM, OpenFlow, dynamic timeout, proactive eviction**

I. Introduction

Software-Defined Networking (SDN) is an innovative network paradigm that decouples control and data planes. This separation enables programmable network operations, fostering rapid advancements in the field. Recently, SDN has been the focus of extensive research and investment, particularly in data center networks [1]–[5], as well as in transport networks where dynamic management of forwarding rules is crucial. For example, Petrosino et al. [6] have explored energy-efficient forwarding in T-SDNs, highlighting the need for adaptive strategies that also consider bandwidth constraints and quality of service, further expanding the relevance and application of SDN technologies beyond traditional settings. These networks require a communication protocol between software (control plane) and hardware (data plane) due to their separation. Generally, the OpenFlow protocol serves as the interface between these planes. A critical component of OpenFlow switches is the flow tables [7]. These tables contain information for each flow, dictating the processing and routing of associated packets. Flow tables are implemented using Ternary Content Addressable Memory (TCAM), which enables constant-time flow entry lookups (O(1)) regardless of table size [8]. However, TCAMs have limited memory size, typically accommodating only thousands of entries. This limitation is due to constraints in silicon size, expensiveness, and high power consumption [8]. Consequently, significantly expanding the flow table size is impractical and costly.

In such networks, each incoming packet is compared with existing table entries. If a match is found, the specified policy is applied; otherwise, a $packet-in$ message is sent to the controller for processing [7] and then a flow rule is added to the table. Table overflow occurs when the table is full, preventing new entries and resulting in dropped packets, namely rejected flows [9]. Efficient utilization of TCAM space is crucial to avoid negatively impacting network performance and overburdening the controller. There are two main strategies for this: setting appropriate timeout mechanisms and proactive eviction.

OpenFlow offers two types of timeouts: idle timeout and hard timeout. A hard timeout removes a flow after a specified time, whereas an idle timeout does so after a period of inactivity (no matching packets) [10]. Typically, fixed timeout values are assigned to all entries as flow table management strategies. However, this approach is sub-optimal in SDN due to limited table sizes and varying flow characteristics. Large timeouts can lead to space wastage and potential table overflow, while small timeouts can trigger excessive $packet-in$ messages, straining the controller. Furthermore, considering table occupancy when calculating timeouts can be beneficial, as fixed timeouts do not account for occupancy ratios. When occupancy is low, flow entries should remain longer to minimize table misses. Conversely, when occupancy is high, it is beneficial to remove the flows used infrequently or have been idle for a long time.

OpenFlow also allows evicting an active flow by sending $flow\_removed$ message from controller to the switches. However, flow removal can degrade network performance and increase $packet-in$ messages. Therefore, selecting which flows to evict is crucial.

This paper proposes a dual approach, VITARPEM, which combines dynamic idle timeout adjustment with proactive eviction. When occupancy ratios are low, we rapidly increase idle timeouts using $packet-in$ count; when occupancy is higher, we cautiously increment the timeouts or use the same timeouts based on flow statistics. We use a dynamic t_{max} to limit excessive table usage of the flows.

Additionally, we calculate *ActiveFlowMetrics* for flows based on their usage frequency to guide proactive eviction. Periodically, we assess table occupancy and, upon reaching a high threshold, remove the flows with the lowest frequency of use until reaching a lower threshold for the table occupancy. Simulation results indicate that, compared to the widely used fixed timeout mechanism, *VITARPEM* achieves an average packet-

in count reduction of approximately 7%, ensuring efficient utilization of network resources and minimizing packet loss.

The paper is structured as follows: Section II reviews related work, providing an overview of existing solutions and techniques in flow table management and discussing their limitations, which our work aims to address. Section III details our methodology and algorithm design. Section IV details the process and outcomes of the parameter tuning simulations. Section V presents simulation results, showcasing the effectiveness of *VITARPEM* in reducing $packet - in$ count and rejected flows compared to *AFTM* [9], *TSMM* [11] algorithms, and fixed timeout mechanism. Finally, Section VI concludes the paper, summarizing our contributions and suggesting directions for future research.

II. Related Work

In recent years, significant research attention has been directed towards optimizing flow table management in SDNs, primarily due to its constrained capacity. These methods have predominantly concentrated on strategies such as dynamic timeout mechanisms, which adapt based on flow characteristics, and proactive eviction mechanisms. Some approaches have even integrated machine learning techniques to augment eviction or timeout allocation strategies.

Shen et al. [9] introduces a dynamic idle timeout mechanism combined with proactive eviction, utilizing three modules: caching, timeout assignment, and eviction. The cache module records flow statistics, which are then utilized in the timeout assignment module to determine idle timeouts for specific flows. Periodically, the eviction module removes long-lived flows using dynamic threshold values. However, these evictions can cause premature removal of frequently used flows. In contrast, our algorithm considers both the hit count and the duration a flow has remained in the flow table, which helps in preserving flows that arrive frequently. Hit counts occur when a flow is received and it does not generate a $packet - in$ message but is processed immediately. This approach is similar to tracking the packets per second for each flow. Additionally, the proposed method, *AFTM*, does not incorporate table occupancy in its timeout decisions, potentially leading to situations where the table remains highly occupied without adjusting the timeouts accordingly. Our method improves upon this by incrementally adjusting timeouts based on initial flow arrivals and current table occupancy, ensuring a more balanced and efficient utilization of the flow table.

Li and Huang's two-stage timeout mechanism for SDN switches optimizes flow table utilization by assigning dynamic timeouts and using an Inactive Flow Queue (IFQ) for potential reactivation of flow rules [12]. While this method adapts to flow behavior, its reliance on accurate timeout predictions and the complexity of managing a secondary queue could introduce overhead and impact network performance. Since our method VITARPEM integrates dynamic timeout adjustments and proactive eviction directly, it simplifies the algorithm by avoiding secondary storage and reducing operational overhead.

Panda et al. [13] employs dynamic hard timeout allocation coupled with LRU-based proactive eviction. This method targets flows whose idle timeout (t_{idle}) equals the maximum timeout value t_{max}. However, this approach may yield suboptimal results, particularly when numerous flows have idle times less than t_{max}. Additionally, the use of hard timeouts can result in flow removals occurring just before new packets arrive, thereby increasing table misses. Furthermore, the lack of consideration for table occupancy when assigning timeouts may lead to excessive timeouts, exacerbating table occupancy issues. Additionally, the absence of controls in the proactive eviction mechanism could result in significant fluctuations in table occupancy, leaving the table either too full or too empty, thus not optimally utilizing the table capacity. To mitigate these issues, our algorithm incorporates a lower bound on the flows to be evicted, ensuring a more stable and efficient flow table management.

SmartTime by Vishnoi et al. [14] uses a dynamic approach to manage TCAM space in SDNs through adaptive timeouts and random proactive eviction. However, this random eviction might not always target the least necessary flows, leading to potential inefficiencies and increased network overhead. Our proactive eviction method targets less frequently used flows for eviction, which potentially leads to more effective TCAM management. Moreover, the reliance on continuous network condition monitoring to adjust timeouts adds complexity and may impact system responsiveness and scalability.

Studies in [1], [15], and [16] primarily focus on proactive eviction strategies. In reference [15], Wu, Qiao, and Chen utilize an LRU-based flow table management algorithm that keeps many active entries and evicts the oldest unmatched flow entry when full. While this method helps reduce the number of messages sent to and from the network controller and shortens wait times for processing, it might remove relevant flows because it only considers the age of entries. In contrast, our approach, VITARPEM, enhances this model by integrating not only eviction based on time but also considering the frequency of flow entries, enabling a more efficient management of the flow table that aligns more closely with network demands. In references [1] and [16], the studies utilize machine learning models to classify flow entries as active or inactive, enhancing flow table management in SDNs. This categorization allows for dynamic eviction based on the activity status of flows, minimizing table overflow by prioritizing the removal of inactive entries. However, these approaches heavily rely on the accuracy of the predictive models and require substantial offline training, which could lead to inefficiencies under rapidly changing network conditions or new traffic patterns not reflected in the training data. In contrast, our method VITARPEM addresses these limitations by using a more adaptive mechanism that integrates dynamic timeout adjustments with heuristic-based eviction, reducing dependency on historical data.

The IHTA algorithm [8] dynamically allocates both idle and hard timeouts based on packet inter-arrival times and TCAM capacity, targeting flow rules with the highest hard

timeouts for eviction. However, its fixed threshold approach may lead to inefficient space management under changing network conditions. Contrastingly, our solution improves on this by adjusting eviction criteria dynamically based on flow characteristics and table occupancy offering more adaptive flow table management and reducing unnecessary evictions.

Huang et al. [3] utilized a Markov model in their approach to manage flow table efficiency in SDNs. By identifying entries with the lowest probability of being matched, these entries are proactively evicted. In contrast, VITARPEM not only focuses on the probability of match but also incorporates metrics to determine flow eviction such as frequency of use, aiming to reduce unnecessary evictions and maintain essential flows.

The "TimeoutX" method by Zhang et al. [10] addresses SDN scalability by dynamically setting timeouts based on traffic characteristics and flow table utilization, reducing table misses and blocked packets. However, it relies heavily on accurate traffic predictions and extensive historical data, increasing system complexity and overhead. In contrast, VITARPEM uses a simpler heuristic-based approach, reducing reliance on historical data and computational resources, making it more adaptable for volatile network environments.

Xu et al.'s [17] CPD framework proactively manages potential table overflows by predicting network traffic in SDNs. However, it does not feature dynamic timeout adjustments, which may limit its adaptability to varied traffic conditions. Conversely VITARPEM integrates both proactive eviction and dynamic timeout adjustments, thereby offering efficiency in managing flow tables under diverse network scenarios.

The method proposed in [11] is tailored for software-defined satellite networks and employs a 2-step algorithm. Initially, it dynamically assigns idle timeouts based on the number of $packet-in$ messages received for each flow while also considering table occupancy. The exponential increase in timeouts ensures that flow rules are not prematurely removed from the table, allowing periodically incoming flow rules to persist. However, [11] lacks a maximum timeout constraint, potentially leading to flows remaining in the table for extended periods. Additionally, the absence of proactive eviction may result in high table occupancy levels early on in the process.

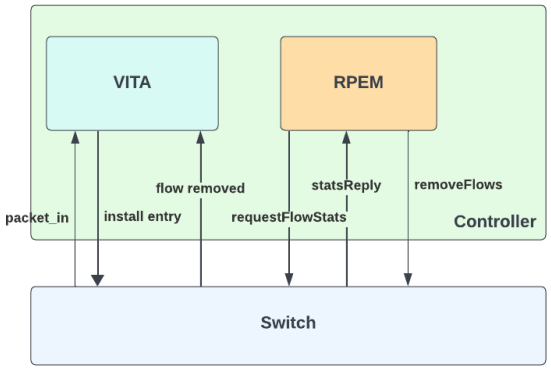


Fig. 1: *VITARPEM* Architecture

Notation	Description
t_{init}	Initial idle timeout value for flows
t_{max}	Max idle timeout value for flows
$\min t_{\text{max}}$	Minimum t_{max} value.
$t_{\text{max\_restore}}$	Initial t_{max} value. It is also used to restore t_{max} when table occupancy is below $TCAM_{lowerbound}$
$t_{\text{lastRemoved}_k}$	Time when flow k is removed from the flow table, either due to eviction or timeout.
t_{packetIn_k}	Time when flow k enters the flow table.
$t_{\text{lastDuration}_k}$	Duration that flow k remained in flow table.
n_{packetIn_k}	Number of times flow k has entered flow table.
$TCAM_{lowerbound}$	Lower bound for table occupancy ratio used in VITA.
$TCAM_{upperbound}$	Upper bound for table occupancy ratio used in VITA.
$TCAM_{limit}$	Flow table capacity for the switch.
$TableOcc$	Current table occupancy ratio.
coef_u	Coefficient for decreasing t_{max} value when the flow table occupancy is above $TCAM_{lowerbound}$.
B	Constant that represents a fixed amount by which the value of t_{max} is reduced when the flow table occupancy is above $TCAM_{lowerbound}$.
$low\_threshold$	Low threshold for table occupancy used in RPEM.
$high\_threshold$	High threshold for table occupancy used in RPEM.
T	Period that RPEM algorithm runs.

table I: Description of Parameters

III. Versatile Idle Timeout Assignment and Responsive Proactive Eviction Method (VITARPEM)

The algorithm, as illustrated in Figure 1, consists of *Versatile Idle Timeout Assignment (VITA)* and *Responsive Proactive Eviction Method (RPEM)* modules. As $packet-in$ messages are received, *VITA* algorithm is activated. The allocation of idle timeouts utilizes the flow table. Periodically, if the table occupancy is above the $high\_threshold$, *RPEM* starts pulling flow table statistics and calculates *ActiveFlowMetric*s accordingly. It then proceeds to delete entries with the smallest *ActiveFlowMetric* values until the occupancy drops to the $low\_threshold$.

Algorithm 1 Versatile Idle Timeout Assignment

Require: t_{init}, $t_{\max}$, $t_{\text{lastRemoved}_k}$, t_{packetIn_k}, $t_{\text{lastDuration}_k}$, n_{packetIn_k}, $t_{\text{max\_restore}}$, min$t_{\max}$
Require: $TCAM_{lowerbound}$, $TCAM_{upperbound}$, coef$_u$, B
Ensure: $Timeout\ T$
1: **for** each packet-in message arrives **do**
2: Record the arriving time t_{packetIn_k} of flow k
3: **if** not found in data table **then**
4: $n_{\text{packetIn}_k} = 1$ ▷ number of packet-in is 1
5: $T = t_{\text{init}}$ ▷ initialize the idle time with t_{init}
6: **else**
7: $n_{\text{packetIn}_k} = n_{\text{packetIn}_k} + 1$ ▷ increase n_{packetIn_k}
8: **if** TableOcc $\leq TCAM_{lowerbound}$ **then**
9: $t_{\max} = t_{\text{max\_restore}}$
10: $T = \min(t_{\text{init}} \times 2^{n_{\text{packetIn}_k}}, t_{\max})$
11: **else if** TableOcc $\leq TCAM_{upperbound}$ **then**
12: $t_{\max} = \min(t_{\max} \times \text{coef}_u - B, \text{min}t_{\max})$
13: **if** $t_{\text{packetIn}_k} - t_{\text{lastRemoved}_k} \leq t_{\text{lastDuration}_k}$ **then**
14: $T = \min(t_{\text{lastDuration}_k} + t_{\text{packetIn}_k} - t_{\text{lastRemoved}_k}, t_{\max})$
15: **else**
16: $T = t_{\text{lastDuration}_k}$
17: **end if**
18: **else if** TableOcc $> TCAM_{upperbound}$ **then**
19: $T = 1$
20: **end if**
21: **end if**
22: **end for**
23: **return** T

A. Versatile Idle Timeout Assignment

The *VITA* algorithm is activated upon receiving a packet-in message. It employs four methods to set idle timeouts using statistics from the data table as illustrated in Algorithm 1. The data table stores the following metrics for each flow: n_{packetIn_k}, t_{packetIn_k}, $t_{\text{lastDuration}_k}$, $t_{\text{lastRemoved}_k}$.

$n_{\text{packetIn}k}$ serves as a metric to track how many times flow rules re-enter the flow table after their initial departure. The term t_{packetIn_k} is the timestamp of the packet-in event. $t_{\text{lastRemoved}_k}$ indicates the most recent time a flow was removed from the flow table. The $t_{\text{lastDuration}_k}$ represents the total duration a flow spent in the table during its most recent presence. Each parameter plays a crucial role in understanding and managing the life cycle of flows within the network infrastructure. For example, a flow with a low $t_{\text{lastDuration}_k}$ but high n_{packetIn_k} might indicate that the flow rule is frequently re-entering the table and therefore should be assigned a higher idle timeout.

Upon a flow entering the flow table, its arrival time, denoted as $t_{\text{packetIn}k}$, is logged in the data table (Line 2). If the flow enters the flow table for the first time (Line 3), it is assigned an initial idle timeout of t_{init} (Line 5), and n_{packetIn_k} is initialized to 1 (Line 4).

If a flow rule already exists in the data table, meaning it has previously entered and been removed from the flow table, n_{packetIn_k} is increased by one (Line 7). The table occupancy will then be evaluated for idle timeout allocation.

When the flow table is not heavily occupied, assigned idle timeout increases rapidly, doubling with each new packet, as expressed by the formula $2^{n_{\text{packetIn}_k}}$ (Line 10). Because idle timeout is increasing exponentially flow rules stays longer if they arrived in the flow table after being removed in the table when there is enough space. Idle timeout is assigned with an exponentially increasing value because flow rules that reenter the flow table are more active than others and should stay longer in the flow table. $packet - in$ count from the data table helps assign longer timeouts for recurring flows, up to a maximum limit, $t_{\max}$. This value is designed to prevent the flow table from getting too full. We designed $t_{\max}$ to be dynamic, having its maximum value when table occupancy is below $TCAM_{lowerbound}$ (Line 9).

Between the lower bound and upper bound thresholds (Line 11-18), the idle timeout is set according to the flow's inter arrival time. As shown in line 12, $t_{\max}$ is dynamically decreased based on table density. $mint_{\max}$ value represents the lowest value to which $t_{\max}$ can be decreased. If a flow re-enters the flow table in a time shorter than its last duration, it is assigned an idle timeout equal to the sum of its last duration and its interarrival time, as outlined in lines 13 and 14.

Whenever a flow reappears after a significant delay, the timeout resets to the previous value from the data_table, calculated by the difference $T_{\text{packetIn}} - T_{\text{lastRemoved}}$ as line 16 details.

When the table occupancy surpasses the upper bound, a minimal idle timeout of 1 is imposed.

Flow Metrics Collector algorithm runs whenever a flow is removed. Flow data is collected for idle timeout allocation. Inputs of VITA is logged in data table. Additionally, the information exchange between the controller and the switch within the VITA algorithm consists solely of the flow add message. This message is a standard component across all flow management strategies, ensuring that no extra messages are required for the VITA algorithm beyond what is universally necessary for initiating flow control actions. This streamlined approach minimizes information exchange overhead, enhancing system efficiency. In terms of time complexity, each operation within the VITA algorithm including conditional checks and arithmetic calculations, executes in constant time due to the absence of iterative or recursive components. Although the function is invoked each time a flow is added to the network, its performance remains consistent due to these characteristics, ensuring that the execution time does not depend directly on the number of active flows but on the frequency of flow additions.

Algorithm 2 Flow Metrics Collector

Require: t_{packetIn_k}
1: **for** each flow_removed message arrives for flow k **do**
2: Record the arriving time $t_{\text{lastRemoved}_k}$ of flow k
3: $t_{\text{lastDuration}_k}$ = $t_{\text{lastRemoved}_k}$ - t_{packetIn_k}
4: **end for**

B. Responsive Proactive Eviction Method

When the flow table reaches its capacity, network performance can deteriorate significantly. Incoming flows face the risk of being dropped or may necessitate the eviction of existing flows. The primary goal is to ensure that the flow table never becomes full. To this end, we periodically run a proactive eviction algorithm as shown in Algorithm 3 at intervals denoted by T. Within *RPEM*, when table occupancy exceeds $high\_threshold$, the controller issues a stats request namely OFPFlowStatsRequest to the switch using OpenFlow API for hit count of flows. Although crucial, this process increases the information exchange overhead, particularly in dynamic environments with a high number of flow updates. Upon receiving the statistics, the algorithm calculates frequency of use named *ActiveFlowMetric* for each flow, based on the hit count from flow statistics and the packet-in time from the data table.

ActiveFlowMetric is calculated as follows:

$$\text{ActiveFlowMetric} = \frac{\text{total\_hits}}{t_{\text{current}} - t_{\text{packet-in}}} \tag{1}$$

total_hits describes the times when a flow received by switch and matched with the flow table. t_{current} represents the current time and $t_{\text{packet-in}}$ is the same parameter used in idle timeout allocation which is the arrival time of the packet-in message.

The algorithm continues to remove flows from the proactive eviction table, which contains flows paired with their calculated *ActiveFlowMetrics*, specifically targeting those with the lowest values until the occupancy drops below the $low\_threshold$. For each selected flow rule, a delete command is sent to the switch, effectively removing the flow causing an additional information exchange overhead. (lines 6, 7). However, this overhead is managed strategically: proactive eviction is only triggered when the table occupancy surpasses the set upper threshold, ensuring that the overhead is justified by the need to maintain optimal network performance and flow table utilization.

In terms of computational complexity, the proactive eviction function operates with a time complexity of $O(k(logN))$, where k is the number of flows removed, N represents the total entries in the proactive eviction table. This logarithmic complexity in relation to the number of flows is primarily due to the heap operations involved in selecting the least active flows. Despite this complexity, the triggering of this function ensures that it only runs when a certain condition holds, balancing computational overhead with network management needs.

Both *VITA* and *RPEM* offer tunable parameters, allowing adaptation to the specific characteristics of the network. This flexibility ensures that network performance is optimized while preventing table overflow and the associated negative consequences.

Algorithm 3 *RPEM* Running Every T Seconds

Require: $low\_threshold$, $high\_threshold$
1: **if** TableOcc $\geq$ high_threshold **then**
2: Request flow rule stats from the switch
3: Wait until the reply comes from the switch
4: **if** TableOcc $\geq$ high_threshold **then**
5: **while** TableOcc $>$ low_threshold **and** proactive_eviction_table not empty **do**
6: pop one flow from proactive_eviction_table ▷ pop the one with lowest *ActiveFlowMetric*
7: remove flow from flow table ▷ send delete message to switch
8: **end while**
9: **end if**
10: **end if**

IV. Parameter Tuning and Analysis

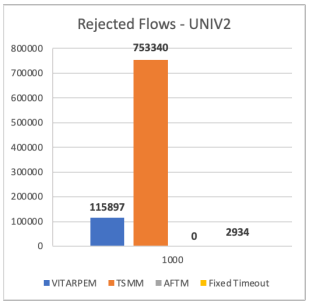
A. Experimental Setup

In order to optimize our algorithm for various network conditions, we conducted experiments to determine the optimal values for its parameters. We selected a subset of UNIV1 dataset and conducted simulations with flow table sizes of 150 and 300. For each parameter, including mint_{max}, t_{max}, coef$_u$, $TCAM_{upperbound}$, $TCAM_{lowerbound}$, $low\_threshold$, and $high\_threshold$, we systematically varied their values and observed their impact on the results. By evaluating the performance of the algorithm with different parameter settings, we aimed to identify the configurations that yield the best outcomes in terms of network performance metrics.

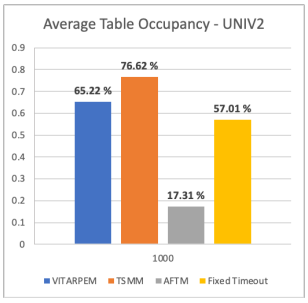
B. Sensitivity Analysis

1) **TCAM$_{\text{upperbound}}$**:Our initial $TCAM_{upperbound}$ for setting the timeout to 1 was 0.95. However, during parameter tuning, we found that a high $TCAM_{upperbound}$ resulted in a higher number of rejected flows, which is undesirable. For larger table sizes, the rejected flow numbers were 119 for the value 0.95, while they were 0, 2, and 20 for the values 0.7, 0.8, and 0.9, respectively. Therefore, we decided to decrease it. It's also important to minimize the number of packet-ins. For smaller table sizes, decreasing the $TCAM_{upperbound}$ to 0.7 helped improve the results. For smaller tables, the number of packet-ins is 3577 and 4061 for the values 0.7 and 0.95, respectively. However, for larger table sizes, reducing the $TCAM_{upperbound}$ led to an increase in packet-ins, making it less favorable. We observed an increase from 2394 to 3759 when increasing the parameter value from 0.7 to 0.95. This is because, after reaching this $TCAM_{upperbound}$, idle timeouts are set to 1, causing flows to be removed from the table more quickly and resulting in frequent packet-ins. Thus, we determined that a $TCAM_{upperbound}$ of 0.8 strikes a good balance between small and large table sizes. This parameter can be further optimized based on the specific resources and needs of a network.

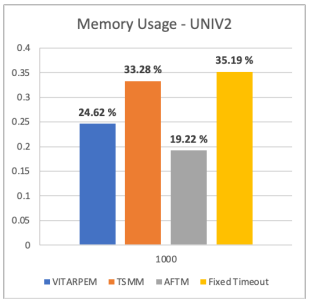
2) **TCAM$_{\text{lowerbound}}$**: Initially, we set the $TCAM_{lowerbound}$ to 0.6. If this threshold is not passed, idle timeouts are increased faster for re-entering flows. A high $TCAM_{lowerbound}$ resulted in more rejected flows because flows stayed longer in the flow table, which also led to a higher number of packet-ins for a smaller flow table. We observed an increase from 35 to 119 when increasing the parameter value from 0.4 to 0.6 in terms of rejected flows. However, changing $TCAM_{lowerbound}$ did not have much effect when the table size was larger. No flows were rejected for all parameter values. Therefore, we decided to set the $TCAM_{lowerbound}$ to 0.4.
3) **high_threshold**: The $high\_threshold$ is used as a trigger mechanism to start evicting flows when table occupancy reaches this level. This threshold did not have much effect on results when the table size was larger. In all values, no flows were rejected. A high $high\_threshold$ resulted in a high number of rejected flows because the table was fuller, while a very low $high\_threshold$ led to a higher number of packet-ins due to suboptimal table usage for a smaller table size. For example, at the value 0.90, 3424 flows were rejected, while 1682 flows were rejected at the value 0.7. We decided to set the $high\_threshold$ to 0.85 as a balanced value.
4) **low_threshold**: $low\_threshold$ ensures that flow rules are evicted from the flow table until the table occupancy reaches this threshold. Having a high $low\_threshold$ means that flows are not evicted sufficiently, leading to a quicker fill-up of the table and a higher number of rejected flows. Conversely, lowering the $low\_threshold$ too much makes the flow table too empty, increasing the packet-in count. For example, the number of rejected flows was 2256 and the number of packet-ins was 4881 at the parameter value 0.55, while those values were 3657 and 3681, respectively, at the value 0.55. Therefore, we decided to set the $low\_threshold$ to 0.55 as a balanced value.
5) **t$_{\text{max}}$**: We use the $t_{\max}$ value when the table size is below $TCAM_{lowerbound}$, allowing the idle timeout to be assigned as 2^n. Based on our experiments, we decided to keep $t_{\max}$ at 32. We observed values of 4299, 9968, and 10052 for the number of rejected flows in successive runs for smaller table sizes at the value 32. For the value 64, we observed these values: 9834, 10836, and 9469. Therefore, the simulation results are not consistent enough to provide further insights.
6) **mint$_{\text{max}}$**: When table occupancy is between the $TCAM_{upperbound}$ and $TCAM_{lowerbound}$, $t_{\max}$ is decreased up to a minimum value, mint$_{\max}$. Initially, mint$_{\max}$ was set to 15. Decreasing this value to 10 helped reduce the number of rejected flows from 2463 to 1982 because flows would stay in the table for a shorter time period. The number of rejected flows decreased from 2463 to 1982. Additionally, the number of packet-ins decreased, possibly because flows were staying longer than necessary initially. Increasing mint$_{\max}$ had the opposite effect, resulting in more rejected flows. The number of rejected flows increased. The number of rejected flows increased from 2463 to 6194 when increasing mint$_{\max}$ to 25. We also experimented with decreasing mint$_{\max}$ further, but this led to an increase in packet-ins at a certain point because flow rules were re-entering the flow table even though they were assigned a shorter idle timeout. The packet-ins were 9215 for mint$_{\max}$ value of 25, while it was 3790 for the value of 10.
7) **coef$_u$**: Changing coef$_u$ had no noticeable effect on the results. It did not lead to any rejected flows for larger table sizes when we changed the parameter value from 0.5 to 0.9. It followed a similar trend for the smaller table size. We used the value of 0.75.



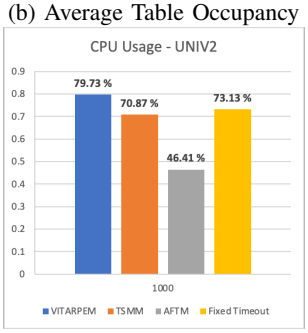
(a) Rejected Flows



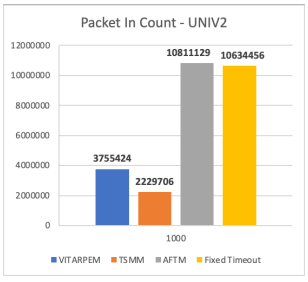
(b) Average Table Occupancy



(c) Average Memory Usage

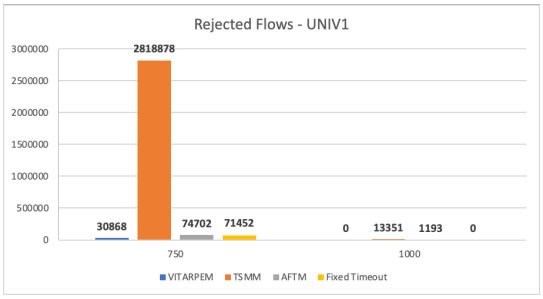


(d) Average CPU Usage

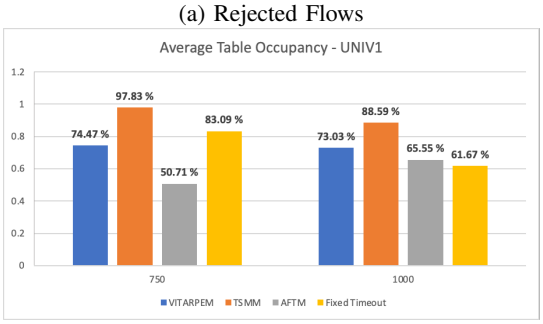


(e) Packet In Count

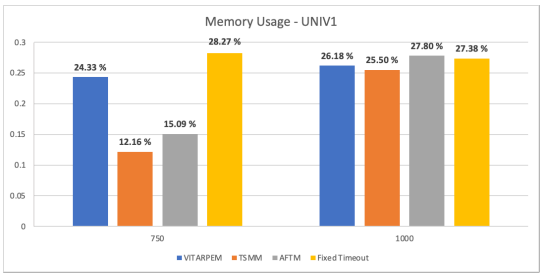
Fig. 2: UNIV2 Simulation Results with 1000 flow table size



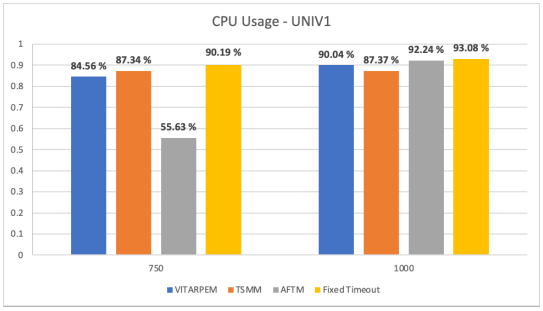
(a) Rejected Flows



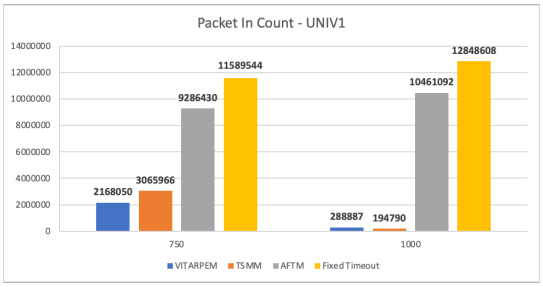
(b) Average Table Occupancy



(c) Average Memory Usage



(d) Average CPU Usage



(e) Packet In Count

Fig. 3: UNIV1 Simulation Results with 750 and 1000 flow table sizes

V. Experimental Results

A. Experimental Setup

To evaluate our algorithm's performance, we are using UNIV1 and UNIV2 network packet traces collected from two university data centers by the authors of [18] . These traces is widely used in studies such as [9] and [19]. While UNIV1 has a 67.4% TCP flow percentage, UNIV2 has 8.70%, consisting mostly of UDP flows [1]. For this analysis, we leveraged the entire UNIV1 trace and one-third of the UNIV2 trace.

We compare our algorithm with existing studies: *AFTM: An adaptive flow table management scheme for OpenFlow Switches* [9] and *Timeout Strategy-based Mobility Management for Software Defined Satellite Networks (TSMM)* [11]. We are comparing the first part of the [11] which is *TSMM* algorithm that adopts a dynamic idle timeout mechanism while considering the table occupancy. However, this algorithm lacks a proactive eviction mechanism. We excluded the handover handling portion of *TSMM* since our simulations are not based on satellite networks. *AFTM* [9] contains both idle timeout allocation and proactive eviction. In addition, we are comparing our algorithm with the fixed timeout mechanism with an idle timeout of 3 seconds and a hard timeout of 5 seconds.

We assess the performance of each algorithm using the given packet trace, focusing on flow table sizes of 750 and 1000 for UNIV1 and only 1000 for UNIV2. Our experimental setup was based on the Mininet [20] simulation environment, with RYU [21] as the controller and employing OpenFlow version 1.3 as the communication protocol. We used a single switch and multiple hosts to simulate network traffic from the datasets, utilizing the tcpreplay tool to replay the captured traffic accurately. We are conducting our simulations in a virtual environment that has 8 allocated processors and 10 GB of memory. The system utilizes an Intel(R) Core(TM) i7-10870H CPU operating at 2.20 GHz.

B. Evaluation Metrics

We will be comparing our algorithms with the following metrics:

1) **Rejected Flows:** The number of flow requests denied due to table overflow.
2) **Average Table Occupancy:** The average occupancy level of the flow table over time.
3) **Average Memory Usage:** The average memory consumption by the control plane.
4) **Average CPU Usage:** The average CPU load of the control plane.
5) **Packet-in Count:** The number of packet-in messages sent to the controller.

C. Comparative Analysis

The fixed timeout approach, with its constant timeout settings, initially maximizes utilization due to its simplicity. However, this approach exhibits less adaptability, especially as the table size increases, leading to decreased utilization in

scenarios with larger table capacities, such as those observed in the UNIV1 data trace as shown in Figure 3. In contrast, *VITARPEM* employs dynamic idle timeout modifications to optimize table utilization effectively and prevent unnecessary flow drops. This method proves particularly advantageous in environments where the fixed timeout approach falls short, demonstrating *VITARPEM*'s adaptability in managing different network conditions. Due to their more cautious flow management strategies, *TSMM* [11] and *AFTM* [9] perform poorly in UNIV1, likely as a result of premature evictions and inadequate adaptation to network conditions in smaller table sizes. *TSMM* [11] has a high average table occupancy, as shown in Fig. 3b, resulting in a high number of rejected flows. *VITARPEM* follows a stable utilization rate of 73.03% across various table sizes.

The proactive eviction technique of *VITARPEM* in the UNIV1 dataset reduces rejected flows by dynamically adjusting to traffic patterns. As shown in Fig. 3a, at a table size of 750, *VITARPEM* outperforms other approaches and achieves zero rejections at a table size of 1000. In comparison, the rejection rates of *AFTM* [9] are similar to those of the fixed timeout approach at smaller table sizes; however, at larger table sizes, *AFTM* rejects more flows than the fixed timeout method. *AFTM* excels in minimizing rejected flows in the UNIV2 data trace, where it rejects no flows at all (See Fig. 2a). Following closely behind, the fixed timeout mechanism rejects only 2,934 flows, while *TSMM* shows significantly higher rejection rates. *VITARPEM* manages 115,897 rejected flows. Despite recording fewer rejected flows in both UNIV1 and UNIV2, *AFTM* and the fixed timeout mechanism exhibit substantially higher numbers of packet-in events—more than twice the amount in UNIV2 and up to 45 times higher in UNIV1 compared to *VITARPEM*. This underscores the effectiveness of *VITARPEM*'s proactive eviction mechanism in retaining more necessary flows than the other approaches. By maintaining a lower packet-in count, *VITARPEM* demonstrates its ability to prioritize the retention of high-frequency and critical flows. Additionally, it reduces the control plane overhead by decreasing the frequency of packet-in events, which typically require processing by the controller. This strategy helps improve the system's responsiveness and supports more stable network operations.

Fig. 2 shows that, both *TSMM* [11] and *VITARPEM* perform well where packet rates and data volumes are higher. *TSMM* [11] adjusts the timeout based on the number of packets per flow, achieving a better packet-in count, especially in UNIV2 compared to other approaches (See Fig. 2e). It should be noted that flow table entries are more active in UNIV2 due to the high percentage of UDP flows, which is uncommon in real networks [1]. Although *VITARPEM* handles more packets than *TSMM* [11], its proactive eviction and dynamic idle timeout modifications help maintain a relatively low packet-in count. Notably, *TSMM* rejects a larger number of flows in both traces than *VITARPEM* (See Fig. 3a and 2a) , which may indicate that *TSMM* retains flows longer than necessary, leading to unnecessary resource occupation and increased flow rejections. For example, *TSMM* [11] exhibits approximately six times the number of rejected flows compared to *VITARPEM*, as depicted in Fig. 2a. In contrast, *VITARPEM*'s lower rejection rate suggests a more balanced approach, optimizing resource utilization and adapting efficiently to traffic changes.

The fixed timeout mechanism and *AFTM* [9] show less flexibility, as evidenced by their higher packet-in counts. Regarding packet-in counts, *VITARPEM* demonstrates excellent performance, recording the lowest packet-in counts at smaller table sizes in UNIV1 as illustrated in Fig 3e. In UNIV2 and at larger table sizes in UNIV1, *TSMM* [11] performs slightly better (See Fig. 2e); however, as noted earlier, it also rejects a significantly higher number of flows(See Fig. 3a and 2a).

Continuing our analysis, we evaluate the execution times of the algorithms for managing 100 flows, using flow installation time as a proxy to underscore the distinctions among their performances. The Fixed Timeout Mechanism, operating without a dynamic algorithm, exhibited the fastest installation time at 0.016549 seconds, reflecting its straightforward, non-adaptive approach. Following closely, VITARPEM's VITA component achieved an execution time of 0.018772 seconds, highlighting its ability to efficiently manage flow setups despite its dynamic operations. In contrast, TSMM and AFTM recorded slower times of 0.023401 and 0.020092 seconds respectively. These findings affirm that VITARPEM not only effectively balances flow management but also maintains a competitive edge in execution speed, which is crucial in high-performance networks where rapid response to traffic dynamics is essential.

To sum up, *VITARPEM* performs well for different table sizes in the UNIV1 data trace and table size 1000 in UNIV2 data trace thanks to its proactive eviction technique, which dynamically adjusts idle timeouts in response to traffic patterns. Because of its flexibility, our approach reduces rejected flows and keeps table occupancy at an optimal level. Despite recording higher CPU and memory usage percentages (See Fig. 3c and 3d), it is important to note that these simulations were conducted on a machine with limited processing power and memory capacity. Under typical conditions with more robust hardware, the increased resource demands of *VITARPEM*'s aggressive flow control tactics would likely be mitigated, enhancing its overall suitability for challenging network environments.

VI. Conclusion and Future Work

In SDN, the controller plays a pivotal role in managing the network by installing flow rules on switches. Given the inherent limitations in flow table capacity, efficient flow table management is crucial. In our paper, we introduce an innovative algorithm that synergizes proactive eviction with dynamic idle timeout allocation, thereby refining the optimization of flow table capacity. Our algorithm demonstrates a substantial reduction in the packet-in count, particularly when compared with fixed timeout methods and the *AFTM* [9] algorithm. It also effectively lowers the number of rejected flows, especially relative to the *TSMM* [11] algorithm. Notably, our method

maintains efficient table occupancy rates, achieving better table utilization especially at smaller table sizes.

The simulation results support the effectiveness of our algorithm, showcasing its ability to maintain an equilibrium across various performance metrics. This balanced performance, coupled with the potential for further improvements, positions our algorithm as a promising solution for flow table management in SDN environments.

In the future, we plan to use machine learning techniques to improve our proactive eviction mechanism, helping us make better decisions when removing flows. We also hope to apply this approach to satellite networks, where it could improve flow management despite the challenges of high latency and changing network structures.

REFERENCES

[1] H. Yang, G. F. Riley, and D. M. Blough, "Stereos: Smart table entry eviction for openflow switches," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 377–388, 2020.

[2] K. Park, S. Sung, H. Kim, and J. il Jung, "Technology trends and challenges in sdn and service assurance for end-to-end network slicing," *Computer Networks*, vol. 234, p. 109908, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623003535

[3] G. Huang and H. Y. Youn, "Proactive eviction of flow entry for sdn based on hidden markov model," *Frontiers of Computer Science*, vol. 14, pp. 1–10, 2020.

[4] Y. Maleh, Y. Qasmaoui, K. El Gholami, Y. Sadqi, and S. Mounir, "A comprehensive survey on sdn security: threats, mitigations, and future directions," *Journal of Reliable Intelligent Environments*, vol. 9, no. 2, pp. 201–239, 2023.

[5] S. W. Turner, M. Karakus, E. Guler, and S. Uludag, "A promising integration of sdn and blockchain for iot networks: A survey," *IEEE Access*, vol. 11, pp. 29 800–29 822, 2023.

[6] A. Petrosino, G. Sciddurlo, G. Grieco, A. Shah, G. Piro, L. A. Grieco, and G. Boggia, "Dynamic management of forwarding rules in a t-sdn architecture with energy and bandwidth constraints," in *Ad-Hoc, Mobile, and Wireless Networks. ADHOC-NOW 2020*, ser. Lecture Notes in Computer Science, L. Grieco, G. Boggia, G. Piro, Y. Jararweh, and C. Campolo, Eds., vol. 12338. Cham: Springer, 2020, pp. 3–15.

[7] M. Abbasi, S. Maleki, G. Jeon, M. R. Khosravi, and H. Abdoli, "An intelligent method for reducing the overhead of analysing big data flows in openflow switch," *IET communications*, vol. 16, no. 5, pp. 548–559, 2022.

[8] B. Isyaku, M. B. Kamat, K. b. Abu Bakar, M. S. Mohd Zahid, and F. A. Ghaleb, "Ihta: Dynamic idle-hard timeout allocation algorithm based openflow switch," in *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 2020, pp. 170–175.

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

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

[11] T. Li, H. Zhou, H. Luo, W. Quan, Q. Xu, G. Li, and G. Li, "Time-out strategy-based mobility management for software defined satellite networks," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 319–324.

[12] X. Li and Y. Huang, "A flow table with two-stage timeout mechanism for sdn switches," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 1804–1809.

[13] A. Panda, S. S. Samal, A. K. Turuk, A. Panda, and V. C. Venkatesh, "Dynamic hard timeout based flow table management in openflow enabled sdn," in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, 2019, pp. 1–6.

[14] 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: Association for Computing Machinery, 2014, p. 177–188. [Online]. Available: https://doi.org/10.1145/2611286.2611301

[15] D. Wu, L. Qiao, and Q. Chen, "Research and implementation of lru-based flow table management for onboard switch," in *2020 Prognostics and Health Management Conference (PHM-Besançon)*, 2020, pp. 300–303.

[16] H. Yang and G. F. Riley, "Machine learning based flow entry eviction for openflow switches," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, pp. 1–8.

[17] J. xu, L. Wang, C. Song, and Z. Xu, "Proactive mitigation to table-overflow in software-defined networking," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 00 719–00 725.

[18] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 267–280. [Online]. Available: https://doi.org/10.1145/1879141.1879175

[19] T. Li, H. Zhou, H. Luo, I. You, and Q. Xu, "Sat-flow: Multi-strategy flow table management for software defined satellite networks," *IEEE Access*, vol. 5, pp. 14 952–14 965, 2017.

[20] "Mininet," http://mininet.org/, [Accessed in 03.01.2024].

[21] "RYU SDN Framework," https://ryu-sdn.org/, [Accessed in 03.01.2024].