ADAPTIVE TIMEOUT & EVICTION ORCHESTRATOR FOR SOFTWARE-DEFINED NETWORKING

by

Efe Ertekin

M.S., Computer Engineering, Boğaziçi University, 2025

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2025

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Professor Fatih Alagöz, for his support and invaluable guidance throughout my research. His knowledge and experience were enlightening during the tough moments.

I am also deeply thankful to my co-supervisor, Dr. Beytüllah Yiğit, for his suggestions and support over the years. I had a chance to sharpen my thinking through his critical comments. He helped a lot for me bring greater depth to this study. In addition, I am grateful to the juries for their valuable time and feedback.

ABSTRACT

ADAPTIVE TIMEOUT & EVICTION ORCHESTRATOR FOR SOFTWARE-DEFINED NETWORKING

Flow table management plays a vital role in enhancing the scalability, efficiency, and overall performance of Software-Defined Networking (SDN). In this thesis, an innovative mechanism called Adaptive Timeout Assignment and Eviction Orchestrator (ATEO) is presented. Multi-layered timeout assignment module enhanced with Z-score-based flow classification, a reactive heuristic-driven eviction mechanism, and a dynamic threshold adjuster module to adapt to various network conditions are introduced as a comprehensive flow management strategy with ATEO. The timeout assignment module of ATEO intelligently modifies timeout values by analyzing the statistical characteristics of flows. The Z-score-based flow classification module differentiates between short-lived "mice" flows and long-lived "elephant" flows. This classification helps ATEO to dynamically allocate more persistent timeouts to frequently accessed or critical flows. On the other hand, the heuristic-based eviction module identifies and removes flow rules that are expected to have a lower impact in the future. This allows ATEO to create space for the newcomer flows that are potentially more important. In addition, it ensures that the flow table is populated with high-priority flows which minimizes the unnecessary packet-in messages and the controller overhead. Additionally, ATEO's proposed dynamic threshold adjuster monitors the occupancy of the flow table and the changes in the number of rejected flows. It uses these data to generate a near real-time response. Thresholds that are used in timeout and eviction processes are updated to keep the number of rejected flows lower while having high Quality of Service (QoS). It significantly reduces the rejected flows while keeping the packet-in count competitive with other algorithms. The results show that ATEO achieved an efficient flow table management which is essential for modern network infrastructures.

ÖZET

YAZILIM TANIMLI AĞ İÇİN UYARLANABİLİR ZAMAN AŞIMI & TAHLİYE ORKESTRATÖRÜ

Akış tablosu yönetimi, Yazılım Tanımlı Ağlar (SDN) için ölçeklenebilirliği, verimliliği ve genel performansı artırmada hayati bir rol oynar. Bu çalışmada, yenilikçi bir mekanizma olan Uyarlanabilir Zaman Aşımı Atama ve Atılma Düzenleyicisi (ATEO) sunulmuştur. Z-puanı tabanlı akış sınıflandırması ile geliştirilen çok katmanlı zaman aşımı atama modülü, tepki veren sezgisel yaklaşıma dayalı bir atılma mekanizması ve çeşitli ağ koşullarına uyum sağlayabilen dinamik eşik ayarlayıcı modülü ile birlikte ATEO kapsamlı bir akış yönetim stratejisi sunar. ATEO'nun zaman aşımı atama modülü, akışların istatistiksel özelliklerini analiz ederek zaman aşımı değerlerini akıllıca değiştirir. Z-puanı tabanlı akış sınıflandırma modülü, kısa ömürlü "fare" akışları ile uzun ömürlü "fil" akışlarını ayırt eder. Bu sınıflandırma, ATEO'nun sık erişilen veya kritik öneme sahip akışlara daha kalıcı zaman aşımları atamasına olanak tanır. Öte yandan, sezgisel yaklaşıma dayalı atılma modülü, gelecekte daha az etkili olması beklenen akış kurallarını tespit edip kaldırır. Bu sayede ATEO, potansiyel olarak daha önemli olan yeni gelen akışlar için yer açar. Ayrıca, akış tablosunun yüksek öncelikli akışlarla dolu kalmasını sağlar, bu da gereksiz "packet-in" mesajlarını ve denetleyici üzerindeki yükü azaltır. Buna ek olarak, ATEO'nun önerdiği dinamik eşik ayarlayıcı, akış tablosunun doluluk oranını ve reddedilen akış sayısındaki değişiklikleri izler. Bu verileri kullanarak neredeyse gerçek zamanlı yanıtlar üretir. Zaman aşımı ve atılma süreçlerinde kullanılan eşik değerleri, yüksek Hizmet Kalitesi (QoS) sağlarken reddedilen akışların sayısını düşük tutmak amacıyla güncellenir. Reddedilen akışları önemli ölçüde azaltırken, "packet-in" sayısını da diğer algoritmalarla rekabet edebilir seviyelerde tutmuştur. Sonuçlar, ATEO'nun modern ağ altyapıları için gerekli olan verimli bir akış tablosu yönetimi sağladığını kanıtlamaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES vii
LIST OF TABLES viii
LIST OF SYMBOLS ix
LIST OF ACRONYMS/ABBREVIATIONS xi
1. INTRODUCTION 1
2. RELATED WORK 6
3. ATEO: SYSTEM ARCHITECTURE AND IMPLEMENTATION 18
3.1. Dynamic Threshold Adjustment Based on Rejected Flows 20
3.2. Z-Score Detection for Elephant Flows and Heuristic Importance Calculation 22
3.3. Reactive Eviction 25
3.4. Timeout Assignment 26
4. RESULTS 29
4.1. Experimental Setup 29
4.2. Evaluation Metrics 31
4.2.1. Packet-In Count 31
4.2.2. Rejected Flows 32
4.2.3. Flow Table Utilization 32
4.2.4. Computation Overhead 33
4.3. Algorithmic Constants and Rationale 33
4.4. Comparative Analysis 35
5. CONCLUSION AND FUTURE WORK 42
REFERENCES 43

LIST OF FIGURES

Figure 1.1. Traditional Network & Software-Defined Network. 1

Figure 1.2. SDN Controller & Data Plane Structure. 2

Figure 1.3. Packet Arrival Pattern Examples. 3

Figure 3.1. Architecture of ATEO. 19

Figure 4.1. Topology of Simulation. 30

Figure 4.2. Packet-in Counts at 1000, 1250, and 1500 Table Entry Sizes (UNIV1). 36

Figure 4.3. Packet-in Counts at 250, 500, and 750 Table Entry Sizes (UNIV1). 37

Figure 4.4. Packet-in Counts at 500, 750, and 1000 Table Entry Sizes (UNIV2). 38

Figure 4.5. Rejected Flows at 1000, 1250, and 1500 Table Entry Sizes (UNIV1). 39

Figure 4.6. Rejected Flows at 250, 500, and 750 Table Entry Sizes (UNIV1). . 40

Figure 4.7. Rejected Flows at 500, 750, and 1000 Table Entry Sizes (UNIV2). 41

LIST OF TABLES

Table 2.1. Comparison of Flow Table Management Algorithms in SDN. . . . 14

Table 4.1. Comparison of 3-Second Idle Timeout vs. 3-Second Idle with 5-Second Hard Timeout Mechanisms (UNIV1, Table Size = 750). . . 31

Table 4.2. Performance for Varying Δ (First 10% of UNIV1, Table Size = 500). 35

Table 4.3. Performance for Varying *b_value* (First 10% of UNIV1, Table Size = 1000). 35

LIST OF SYMBOLS

avg_byte_count	Average byte count across flows
avg_hit_count	Average hit count across flows
b_value	Decay factor for exponential timeout reduction
byte_count	Total bytes transferred in a flow
current_occupancy	Current utilization level of the flow table
current_rejected_flows	Number of rejected flows during the current interval
decay_factor	Factor used to decay *last_timeout* for mice flows
delta_rejected_flow	Difference in rejected flows between intervals
eviction_high_threshold	Maximum allowed table occupancy
eviction_interval	Time interval between eviction checks
eviction_low_threshold	Lower threshold for flow table occupancy
heuristic_importance	Value representing the importance of a flow
hit_count	Number of times a flow has been matched in the flow table
idle_timeout	Final timeout assigned to the flow before installation
last_timeout	Previously assigned timeout for a flow
max_rejected	Maximum number of rejected flows observed so far
min_heap	Stores heuristic importance of flows
minimum_t_max	Lowest possible value of *t_max*
minimum_timeout	Minimum allowed timeout value
monitoring_interval	Time interval between flow stats requests
new_delta	Updated Δ value
predicted_timeout	Calculated timeout value for a flow
prev_rejected_flows	Number of rejected flows in the previous interval
ratio_rejected	Ratio of rejected flows to the max observed
std_dev_bytes	Standard deviation of byte counts
std_dev_hits	Standard deviation of hit counts
t_init	Initial idle timeout for new flows
t_max	Maximum idle timeout value

timeout_duration_ratio	Ratio of idle timeout to flow duration
timeout_high_threshold	Upper bound for adaptive timeout control
timeout_low_threshold	Lower bound for adaptive timeout control
tmax_restore_value	Reset value of t_max when flow table occupancy is low
total_hits	Total number of hits received by a particular flow
z_score_bytes	Z-Score of the byte count of a flow, used in elephant flow detection
z_score_data	Dataset storing previous statistics of observed flows for Z-Score analysis
z_score_hits	Z-Score of the hit count of a flow, used in elephant flow detection
z_threshold	Threshold for elephant flow classification
Δ	Scaling factor for threshold updates
λ	Exponential decay term used in t_max adjustment

LIST OF ACRONYMS/ABBREVIATIONS

AI	Artificial Intelligence
AHTM	Adaptive Hard Timeout Method
ATEO	Adaptive Timeout & Eviction Orchestrator
CPU	Central Processing Unit
DL	Deep Learning
DQN	Deep Q-Network
HQTimer	Hybrid Q -Learning-Based Timeout Mechanism
ICMP	Internet Control Message Protocol
IoT	Internet of Things
LRU	Least Recently Used
ML	Machine Learning
ONF	Open Networking Foundation
QoS	Quality of Service
RPEM	Responsive Proactive Eviction Method
SDN	Software-Defined Networking
SDSN	Software-Defined Satellite Networks
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TSMM	Timeout Strategy-based Mobility Management
UDP	User Datagram Protocol
VITA	Versatile Idle Timeout Assignment
VITARPEM	Versatile Idle Timeout Assignment & Responsive Proactive Eviction Method

1. INTRODUCTION

Before merging with the Linux Foundation, goals of Open Networking Foundation (ONF) were the development, standardization, and commercialization of software-defined networking (SDN). According to ONF, SDN is a new network architecture that allows for direct programming and separates network control from forwarding [1]. This definition identifies two characteristics of SDN: programmability on the control plane and decoupling of the control and data planes.

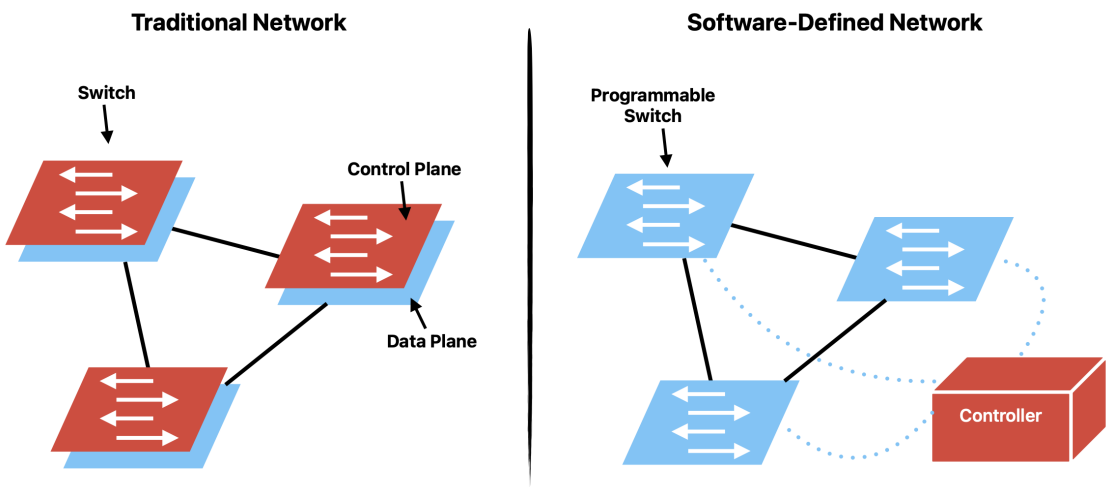


Figure 1.1. Traditional Network & Software-Defined Network.

SDN decouples the data plane, that is in charge of forwarding traffic, from the control plane, which determines where the packet should go [2]. This separation enables centralized management of the network and makes policy changes easy [3] compared to traditional networking systems require every switches to get updated for policy change as seen in Figure 1.1. In addition, optimizing routes and adapting to network changes dynamically can be achieved. When a packet enters the SDN-enabled network, it is examined against the flow table entries to determine the appropriate action. The flow table is a component that contains a set of rules that match packet headers to specific actions, for instance, forwarding, dropping, or modifying the packets [4]. Each entry of the flow table generally includes match fields, counters, and a set of instructions in

them. For example, if there is no match with the new flow, the data plane can redirect this flow to the controller through the southbound interface as a “packet-in” message to learn what it should do next [5]. After the controller receives a packet-in, it processes the information and decides on the appropriate action (e.g., forwarding the packet, dropping it, or setting a new rule). Then, it sends back a “flow-mod” message to the switch to install a new flow entry for subsequent packets [6]. This process enables SDN to react to new flows in near real-time and improves the flexibility of the network by making changes programmatically inside the controller. The simple demonstration of the SDN-enabled network can be seen in Figure 1.2.

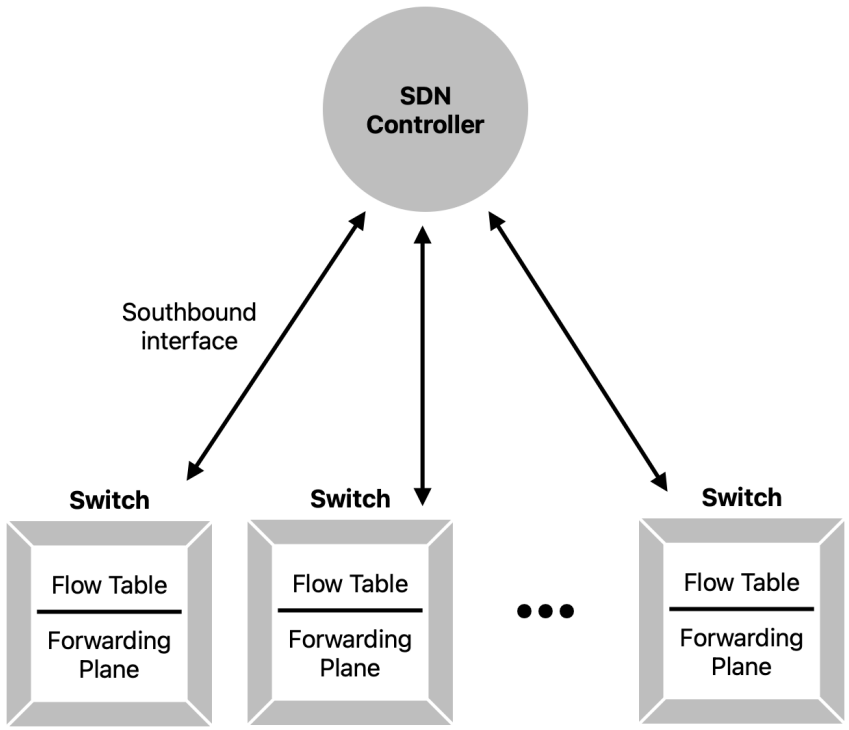


Figure 1.2. SDN Controller & Data Plane Structure.

A flow table is a key component within the SDN data plane component that stores information about handling network traffic based on defined rules or policies. Practically, the flow table serves as a lookup table [7]. Incoming packets are compared against the pre-defined matching fields inside the flow table such as IP addresses, port numbers, or other header attributes. The flow table has entries that establish flow

rules, including matching criteria and associated actions. Entries specify what should happen next if a packet matches. This structure enables the controller to manage flows dynamically based on real-time network conditions and needs. By updating flow entries proactively, the controller can adapt to traffic changes, optimize overall network performance, and enforce security. As a result, programmability, flexibility, and the ability to implement custom strategies that were not feasible with traditional networking paradigms are introduced in SDN.

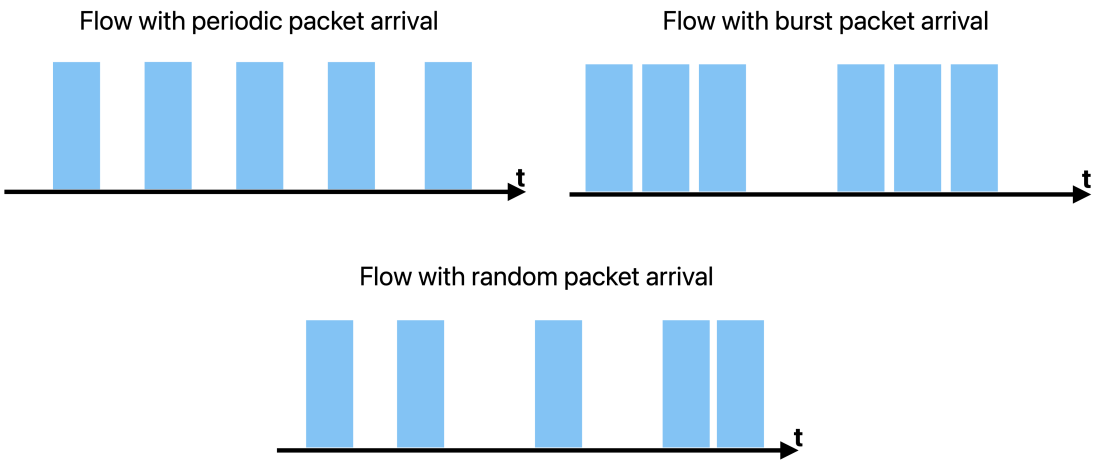


Figure 1.3. Packet Arrival Pattern Examples.

Even with these advantages, efficient flow table management is still a technical challenge to overcome. Limitations like finite flow table sizes, rapid entry timeout expiration dates, different packet arrival patterns, and the overhead caused by frequent table misses make it crucial to design smart flow table management systems. Dealing with various packet arrival patterns is challenging for flow table management solutions because the proposed systems should create a dynamic response to highly variable and often unpredictable traffic behaviors. Packet arrival patterns can vary significantly [8] in terms of volume, timing as seen in Figure 1.3, and source-destination pairs. This diversity of patterns makes the design of the controller problematic. Sudden bursts or irregular flows can easily overwhelm the controller and may lead to packet loss. In addition, frequent updates on flow tables can lead to overhead on both the controller and the switches, causing a further complication in real-time traffic management in an

SDN environment.

One of the fundamental challenges in SDN is efficient usage of the flow table [9]. Ternary Content Addressable Memory (TCAM) is the component that stores flow table entries, and it is an expensive [10] and power-hungry component [11]. Due to these characteristics of TCAM, it is not feasible to store a large number of flow entries inside TCAMs. When the table reaches full capacity, new flow entries cannot be installed, leading to rejected flows [12].

To prevent problems like this from happening, flow table management systems are developed. A timeout value defines the duration of a flow entry that remains in the flow table before being removed [13]. Timeout has two different versions. A hard timeout removes a flow entry after a certain duration [14]. In contrast, idle timeout removes a flow entry if no packets match the flow within the specified time [14]. If the timeout for a flow entry is given too short, this can increase the number of visits to the controller (packet-in count), possibly causing controller overhead. However, assigning excessively long timeouts to flow entries causes them to persist in the table for extended periods, potentially leading to table overflow and an increased number of rejected flows [15]. These systems try to assign ideal timeout values to each flow entry to prevent table overflow and unnecessary rejected flows.

These scenarios make efficient timeout and eviction algorithms important. Eviction is the process of removing the right flow entries based on a policy to make some room for new flows. However, identifying ideal flow entries and how many should be evicted is a challenge in the eviction process. For instance, if the eviction mechanism removes the frequently-used flows from the table, this will lead to higher controller overhead by increasing the number of packet-ins. Handling these conditions requires balancing responsiveness with resource constraints. Without convenient mechanisms, excessive controller overhead and unreasonable rejected flows might occur, which decrease the performance of the network.

In modern networks, the system usually experiences dynamic workloads with varying traffic patterns [16]. And as a result, the balancing act requires real-time adaptability for these networks. The flow management systems with static configurations are not optimal to account for real-time changes in traffic behavior. For this reason, the need for smart flow management algorithms is justified.

Various default settings have been used for timeout in different studies, ranging from 5 seconds in NOX [17] and Floodlight to 20 and 60 seconds in DevoFlow [18]. On the other hand, the lack of an efficient intelligent mechanism to adaptively alter the timeout assigned to flow entries has resulted in inefficient use of flow table resources and excessive controller overhead. A better approach to the timeout mechanism needs to be addressed and this is where ATEO comes into play.

ATEO offers multiple modules to address these issues. Dynamic timeout assignment module to decide proper timeout for flows, reactive eviction module to remove the flows expected to be the least needed, and lastly threshold adjustment module to keep the system stable under different network conditions. The mixture of these modules forms a closed-loop system that has the adaptability to different network conditions by adjusting each other's approach. ATEO's core principle is to combine data-driven flow analysis with lightweight computational overhead to achieve effective SDN control. With the presence of flow activity analysis components such as Z-score-based flow type detection, data-driven timeout assignment module, and threshold-aware eviction process, ATEO presents automated flow table management. The modular design of ATEO enhances the general effectiveness of flow table management by reducing the controller workload.

2. RELATED WORK

Adequate flow table management remains one of the most critical challenges in SDN due to the limited size of TCAM. TCAMs are responsible for storing flow entries on switches and the expense of using high-capacity TCAMs is not feasible due to their high prices. The inability to scale flow tables because of enormous expenses, the traffic that has been increasing day by day is getting harder to handle over time. To overcome these issues, researchers aim to improve timeout configurations and eviction strategies inside the flow table using the controller.

Early adaptations mostly relied on fixed timeout values that were not flexible enough under different traffic patterns which led to excessive controller overhead and premature eviction of actively used flows. In addition, if the fixed timeouts are given high, the flow table reaches its maximum in an instant and at that point, the surge of the rejected flows can be observed. The increased rejected flows lead to lower Quality of Service (QoS) due to the packet losses. This limitation surged adaptive solutions. These adaptive solutions make some calculations to find optimal timeout values for flows and proper candidate flows for the eviction process using real-time metrics such as flow frequency, packet size, duration of the flows, etc.

Most studies have centered on timeout mechanisms that change according to the flow's behavior and reactive eviction mechanisms to free up space by trying to keep the crucial flow entries inside the flow table. These timeouts can be chosen as idle, hard, or both at the same time. Solely relying on hard timeout might cause problems such as early removal of an active flow. You need to guess the flow's active time accurately and assign the hard timeout based on that guess. The idle timeout handles this by resetting the timeout every time the flow appears. On the other hand, the idle timeout may lead to the premature removal of rules for flows with infrequent but periodic traffic like heartbeat signals. Ultimately, choosing the timeout type for your solution depends on the trade-offs you are willing to accept.

In addition to static and adaptive approaches, more recent research has introduced machine learning techniques such as reinforcement learning and deep neural networks to predict optimal timeout values for newcomer flows and eviction strategies by learning from historical values. However, these real-time machine learning models have drawbacks, such as high CPU/Memory usage problems.

This section provides an overview of these algorithms, analyzing their design principles, and their trade-offs to achieve competitive results in SDN environments.

The researchers proposed an enhanced flow table management system in [19] to reduce the number of table-miss packets (packet-in count) in SDN environments by employing a caching algorithm based on the Least Recently Used (LRU) strategy. In traditional OpenFlow switches, expired flow entries are deleted immediately. On the other hand, this proposed method retains expired entries in a cache within the flow table if there is available space. With this, the chances of reusing these flows increase and minimize the unnecessary interactions with the controller. The paper proposes a solution that integrates two components to achieve effective flow table management. The first component is the LRU-based caching algorithm module. This module stores the expired flow entries inside the flow table and assigns them an age value. Each cached entry's age increases in time sequentially. The eviction policy uses these ages to detect the removal prioritization. The least recently used is chosen for removal when the space is needed inside the flow table. The cache size is dynamically adjusted inside the table according to the observed vacancies in the flow table over time to handle the different traffic patterns without risking overflow. The second module is vacancy-based packet forwarding. The goal of this module is to optimize the packet forwarding paths based not only on distance but also considers the available cache space in each switch. Flow entries are distributed to switches to have more vacant space and achieve better load balancing. The lack of an intelligent timeout assignment and eviction modules makes ATEO a more comprehensive flow table management solution overall.

In [20], the paper addresses the challenge of optimizing flow table utilization in SDN by creating a dynamic hard timeout allocation mechanism. The authors concentrate on the limitations of the static timeout values and their inefficiency that leads to increased TCAM usage. This paper proposed a mechanism based on the inter-arrival time of packets and the predictability of flows to solve these challenges. Calculating the average inter-arrival time of the last N packets and dynamically adjusting the timeout value using an exponential scaling factor are the main features of their method. In detail, their solution begins by categorizing flows as predictable or unpredictable according to their packet patterns. For the newcomer flows, there is an initial hard timeout value for flows that is 1 second. After receiving the N number of packets, the system calculates the average inter-arrival time of the packets and updates the new timeout value for a flow based on this. The timeouts are capped at 11 seconds to keep them under control. The goal is to keep the predictable flows longer inside the flow table and try to get rid of the sporadic flows in a short time with this dynamic adjustment. In addition, when the current occupancy of TCAM exceeds a certain threshold which is 90%, the system starts the eviction and uses the LRU policy to remove the flow entries. The purpose of this approach is to balance responsiveness and memory efficiency. The primitive eviction system and fixed thresholds might not assist the system enough to keep up with the different traffic patterns. ATEO both offer smart eviction and reactive threshold adjustment modules to handle varying traffic conditions.

In [21], the paper introduces a dynamic timeout mechanism that adjusts the given timeouts for flows based on the real-time network conditions to improve the flow table utilization in SDN. The method is implemented with the Ryu controller that uses OpenFlow 1.0. It has four major interconnected modules in it to achieve effective flow table management. The first one is the monitoring information module. The main goal of this module is to gather system-level data such as the number of active flow entries, byte counts on ports, and the current CPU load of the controller using OpenFlow messages. These collected data are used to form a foundation for the decision-making process. The second module is the timeout adjustment module. This module uses predefined thresholds to check the occupancy of the flow tables and takes

action to give proper timeouts to newcomer flows. It classifies the flows according to their protocol type and gives a timeout based on this knowledge. It assigns the shortest timeouts to ICMPs, moderate ones to UDPs, and the longer timeout values to TCPs. When a certain level of occupancy is exceeded, it assigns a default timeout value to all flows. The third module is the flow table addition module. It handles the creation of new flow entries using OpenFlow structures. The last module is the active deletion module. This module is triggered only when the controller load exceeds a certain threshold. ICMPs are the first ones that are evicted from the table, followed by UDPs, and lastly TCPs during this process. The lack of adaptive threshold values and primitive eviction mechanism makes this solution incomplete compared to ATEO.

Based on the number of packets arriving at two sample time points and the flow history information base, TimeoutX in [22] estimates the flow type, including the flow's duration and size. A timeout is chosen adaptively for each flow entry based on traffic statistics, flow type, and flow table occupation. TimeoutX assigns dynamic timeout for flows based on the combination of the traffic characteristics, flow types, and current flow table utilization ratio rather than using a fixed timeout for all flow entries. The classification of flows falls into four categories in TimeoutX based on the duration and volume of the flows: short-small, short-large, long-small, and long-large. The general flow of the mechanism starts with the collection of statistics and information on historical flows that are used in other operations. The timeout assignment part uses this collection to estimate the flow's characteristics and uses the current table utilization ratio to assign a new timeout. The flow type prediction is based on the outcomes of two sampling time points. In a dynamic network, this might result in false positives. Because in a dynamic network, choosing the two sampling time points is on its own a significant problem to solve and the paper does not propose a smart mechanism. These time points can lead to different flow type results for the same flow. In addition, due to the lack of a reactive eviction algorithm, TimeoutX cannot provide a whole-package solution for flow table management.

In [23], Adaptive Hard Timeout Method (AHTM) tries to maximize the utilization of the flow table by maximizing the timeouts of flow entries to reuse the flow table effectively. Their solution involves modeling the flow table as an M/G/C/C queueing system, where flows are treated as customers and flow table entries as service stations. AHTM has two components in it. For a given timeout, AHTM-Analyzing computes truncated times, representing the total number of times a flow has been removed from the table and blocking probability which is the likelihood that a new flow is blocked due to no space in the table for a given timeout, and AHTM-Designing determines the optimal timeout based on performance requirements. The method is implemented as a lightweight SDN application on the POX controller, using OpenFlow messages to record flow arrival intervals and service times, and is evaluated through simulations to demonstrate its effectiveness. Unlike AHTM, ATEO offers reactive eviction and threshold adjuster modules to create a more comprehensive algorithm.

In [24], Hybrid Q -Learning-Based Timeout Mechanism (HQTimer) addresses the challenges of SDN by proposing a hybrid Q-learning-based approach for the timeout assignment mechanism. HQTimer utilizes a hybrid timeout mechanism that is boosted with Q-learning-based logic. The hybrid timeout strategy specifically solves the rule dependency problem that is caused by wildcard rules in OpenFlow. It assigns an idle timeout value to table-miss rules and no timeout to their dependent rules. The safely removal of the dependent rules is achieved after the expiration of the associated table-miss rule while preserving the forwarding semantics without requiring modifications to the OpenFlow protocol with this mechanism. It dynamically adjusts timeout values according to traffic scenarios and data plane performance metrics such as table-hit rate, overflow number, and rule dependency. Deep Q-Network (DQN) is employed by HQTimer to model the timeout decision-making as a reinforcement learning problem. The state includes features like a flow rule's cumulative lifetime and hit counts. The action is selecting an appropriate timeout value between 1 and 10 seconds. The reward is based on maximizing the hit efficiency which is the number of table hits per unit of time. Deep Q-Network is used to learn optimal timeout policies while maximizing hit efficiency and minimizing flow table overflow. In addition, this whole system is

designed to operate with minimal control overhead by using passive information from flow-removed and packet-in messages. It avoids the heavy active statistics polling from switches. The model is periodically updated via experience replay which ensures that the learning process remains stable and efficient. However, it lacks a reactive eviction module. HQTimer safely handles timeout expirations and dependent rule removals, but it does not propose an active flow eviction process, unlike ATEO which potentially limits its responsiveness under sudden traffic changes.

In [25], SmartTime proposes a proactive eviction strategy combined with an adaptive timeout heuristic to optimize flow table management in SDN. Their first step is to assign a short initial idle timeout to all flow entries, guaranteeing that short-lived or non-repeating flows are quickly evicted from the flow table. Particularly, SmartTime assigns a fixed idle timeout of 100 milliseconds for every new flow. They modified the Open vSwitch to be able to give milliseconds as a timeout value because current OpenFlow protocols' minimum timeout support is 1 second. This value ensures the flow's immediate removal if it is short-lived or non-repeating. The main goal is to prevent unnecessary TCAM space usage. The assigned timeout grows exponentially with the recurrence of long-lived flows. When a flow comes back to the switch again and it is currently not in the flow table, SmartTime doubles the previously assigned timeout value to allow the flow to stay inside the TCAM longer than before. However, this exponential growth is capped at a maximum idle timeout of 10 seconds to prevent inefficiencies when long-lived flows become inactive and remain inside the flow table due to the higher timeout values. To fix these problems, SmartTime also proposes a mechanism to reset the idle timeout values back to the minimum value when a flow shows poor performance. This performance is based on the flow's staying idle for a disproportionate amount of time compared to its active period. In addition, SmartTime also proposes an eviction mechanism. When the usage of the TCAM crosses the high threshold of 95%, SmartTime triggers the eviction mechanism and starts removing the flow entries randomly to open up some space for newcomer flows. This random eviction policy is chosen because it performs better than the FIFO eviction. However, SmartTime lacks a threshold adjuster, which makes this algorithm less responsive to

different network traffic.

Load-aware feedback control module in the controller and a cache-based timeout prediction model are the two main features of Intelligent Timeout Master [26]. The cache-based timeout prediction model determines an appropriate idle timeout for each flow according to the characteristics of each flow rule. The cache part keeps the expired flow entries and uses them to calculate the packet interval of a flow. When a flow reappears, the controller updates its cached timeout with a heuristic formulation that is based on the interval between the new packet-in event and the last expiration time. With this, the flows that have longer interval times get longer timeout values. In addition, timeout values are capped at 11 seconds to avoid over-allocation. The load-aware feedback control module is designed to decrease flow table overflow by periodically adjusting the maximum timeout depending on the current utilization of the flow table through feedback control. The system uses two different thresholds: start and peak. These thresholds are used to divide the flow table utilization into safe, warning, and reserved zones. The controller monitors the flow table occupancy at regular intervals and adjusts the global maximum timeout value based on the occupancy. If the occupancy is close to the peak threshold while still inside the warning zone, it reduces the maximum timeout value to remove the flows faster. And it restores the maximum value when the occupancy drops back to the safe zone. For the reserved zone, it takes an aggressive action and reduces the maximum timeout value at most. However, the usage of fixed thresholds in a load-aware feedback control module is not ideal for divergent network scenarios. Hence, ATEO adjusts its thresholds according to the increase in rejected flow numbers to keep up with changes in network traffic.

Timeout Strategy-based Mobility Management (TSMM), introduced in [27], focuses on the mobility management of software-defined satellite networks (SDSN). Heuristic algorithm of TSMM alters idle timeouts during satellite handovers to reduce the overloading problem of the flow table caused by unexpired flow entries. TSMM runs in three phases: a “small start” stage to explore the capacity of the flow table, a “fast decrease” stage to ensure space for incoming flows, and a “handover-based adjustment”

stage to arrange timeout values with the remaining connection duration of satellite links. In the small start stage, flows are assigned with low idle timeout values such as a second, and the timeout value is increased exponentially as more flows arrive to make sure that TCAM space is not saturated early. When the current occupancy of the flow table exceeds 95%, the fast decrease stage is triggered and all idle timeout values are fixed to 1 second to free up some space in TCAM. In the final stage, the evaluation of the remaining connection time for each flow based on satellite handover schedules is done. With this, entries that are related to expired or irrelevant paths are removed proactively from the flow tables. Despite its improvements in SDSN scenarios, TSMM has some major shortcomings such as fixed thresholds or advanced flow prioritization, which are key features in ATEO for boosting scalability and efficiency in diverse network conditions.

The proposed approach in [28] introduces an algorithm that combines dynamic idle timeout assignment and proactive eviction mechanisms under VITARPEM. Newly allocated idle timeouts are decided by the algorithm according to the current occupancy of the flow table and traffic characteristics. Versatile Idle Timeout Assignment (VITA) is the module dedicated to timeout assignment. This module adjusts the idle timeout values based on multiple parameters such as the number of packet-in messages received for a flow, its re-appearance frequency, and the current table occupancy. VITA increases the given timeout values for flows when there is an abundance of resources such as low table occupancy. When the opposite conditions are on the stage, it assigns lower idle timeouts. The main goal is to keep flows longer when there is enough space inside the flow table, and when the space is constrained, it tries to remove the flows faster. This ensures more efficient resource utilization overall. The eviction mechanisms of VITARPEM focus on removing the low-impact flows prioritized by the metrics of flow usage to reduce the controller overhead and improve the utilization of the flow table. The Responsive Proactive Eviction Method (RPEM) triggers periodically and calculates a metric for each flow based on its total hit count and active duration. Flows with low metric values are prone to be evicted faster than the higher metric ones. When a certain threshold is exceeded by the current table occupancy, the eviction process

starts. Eviction continues until the table occupancy falls below a predefined lower threshold. This prevents large-scale evictions and stabilizes the system. With these mechanisms, VITARPEM addresses the other timeout mechanisms' inabilities effectively, yet it does not integrate flow classification techniques like Z-score detection for elephant flows and an adaptive threshold adjuster mechanism to handle rapid changes in network conditions properly.

ATEO addresses these drawbacks by using more adequate flow classification methods and threshold adjusters that change according to the current network traffic to further optimize the timeout allocation and eviction policies in flow table management. The comparison of the flow table management algorithms that are mentioned in this section is seen in Table 2.1.

Table 2.1. Comparison of Flow Table Management Algorithms in SDN.

Enhanced Flow Table LRU [19]	*Advantages:* • Reduces packet-in count. • Age-based eviction prioritization. • Dynamic cache size handling. *Disadvantages:* • Needs more memory in flow table. • No learning component.
Dynamic Hard Timeout [20]	*Advantages:* • Adaptability for various traffic. • Uses packet patterns to categorize flows. *Disadvantages:* • Static thresholds. • Simple LRU eviction.

Table 2.1. Comparison of Flow Table Management Algorithms in SDN. (cont.)

Algorithm	*Advantages and Disadvantages*
Dynamic Timeout SDN [21]	*Advantages:* • Protocol-aware timeout assignment. • Avoids controller overload. *Disadvantages:* • No adaptive thresholding. • Primitive eviction strategy.
TimeoutX [22]	*Advantages:* • Flow-type-aware timeout. • Uses historical flow data. *Disadvantages:* • No reactive eviction. • Sampling point issues may cause false predictions.
AHTM [23]	*Advantages:* • Mathematical modeling of flow behavior. • Lightweight POX implementation. *Disadvantages:* • Lacks real-time adaptability. • No reactive eviction.
HQTimer [24]	*Advantages:* • Learns timeout policies via reinforcement learning. • Avoids protocol modification. *Disadvantages:* • No active eviction. • Complex implementation.

Table 2.1. Comparison of Flow Table Management Algorithms in SDN. (cont.)

Algorithm	*Advantages and Disadvantages*
SmartTime [25]	*Advantages:* • Proactive eviction. • Efficient short-flow handling. *Disadvantages:* • Random eviction under high load. • No threshold adjuster.
Intelligent Timeout Master [26]	*Advantages:* • Dual-module design. • Caches expired entries for reuse. *Disadvantages:* • Fixed thresholds reduce flexibility. • No per-flow eviction tuning.
TSMM [27]	*Advantages:* • Tailored for SDSN environments. • Efficient during satellite handovers. *Disadvantages:* • Limited general applicability. • Focused mainly on mobility, not general traffic.
VITARPEM [28]	*Advantages:* • Dynamically adjusts idle timeouts. • Proactively evicts low-impact flows. *Disadvantages:* • No flow classification (e.g., mice/elephant). • Lacks adaptive threshold.

Table 2.1. Comparison of Flow Table Management Algorithms in SDN. (cont.)

Algorithm	*Advantages and Disadvantages*
ATEO	*Advantages:* • Z-score model distinguishes mice/elephant flows. • Reactive eviction using importance heuristics. • The thresholds are updated dynamically. *Disadvantages:* • Higher implementation complexity. • Requires monitoring for near-real-time analytics.

3. ATEO: SYSTEM ARCHITECTURE AND IMPLEMENTATION

In this section, ATEO's dynamic flow management system is examined. ATEO utilizes a combination of adaptive threshold adjustment, heuristic-based flow prioritization for eviction, and intelligent timeout assignment to optimize controller overhead and reduce rejected flow numbers. ATEO dynamically adjusts flow table thresholds based on real-time rejected flow trends, reactively evicts flows using *heuristic_importance* prioritization, and assigns timeout according to the flow's characteristics and the current table occupancy. Additionally, a Z-score-based mechanism is used to distinguish elephant flows from others to provide more information about the flow for timeout assignment and eviction operations.

ATEO is designed to use the limited flow table resources in SDN environments dynamically while keeping the controller burden at reasonable levels. The lightweight yet effective decision-making logic offered by ATEO adapts to the current network conditions to keep the computational burden low and QoS high by causing fewer or no rejected flows inside the system. By making changes inside the controller based on the current network traffic, ATEO manages to prevent most overflow scenarios or under-utilization of the flow table. For ATEO, preventing overflow scenarios is more crucial than under-utilization of the flow table because QoS is the main concern of this solution.

The heuristic-based eviction mechanism focuses on keeping the important flows inside the flow table and the importance of the flow is calculated based on historical behaviors that are captured in real-time. This mechanism allows the system to retain the high-impact flows inside the flow table while removing the low-impact ones to improve the overall traffic handling and flow table management. This leads to fewer packet-in counts and less controller overhead. Unlike static eviction strategies, ATEO's reactive approach can adapt to the traffic changes in the system and adapt to it in real-

time.

The timeout assignment module of ATEO dynamically computes the timeout values that are given to the flows rather than using a static value. Flows that have higher importance and activity receive longer timeouts to avoid quick removal. If these flows are removed quickly, the number of packet-ins might increase a lot, and the controller overhead might be impossible to manage. On the other hand, insignificant ones are assigned lower timeouts to make sure that they do not take the space for a long time inside the flow table. In addition, this module adapts to the occupancy of the flow table to re-adjust its measures.

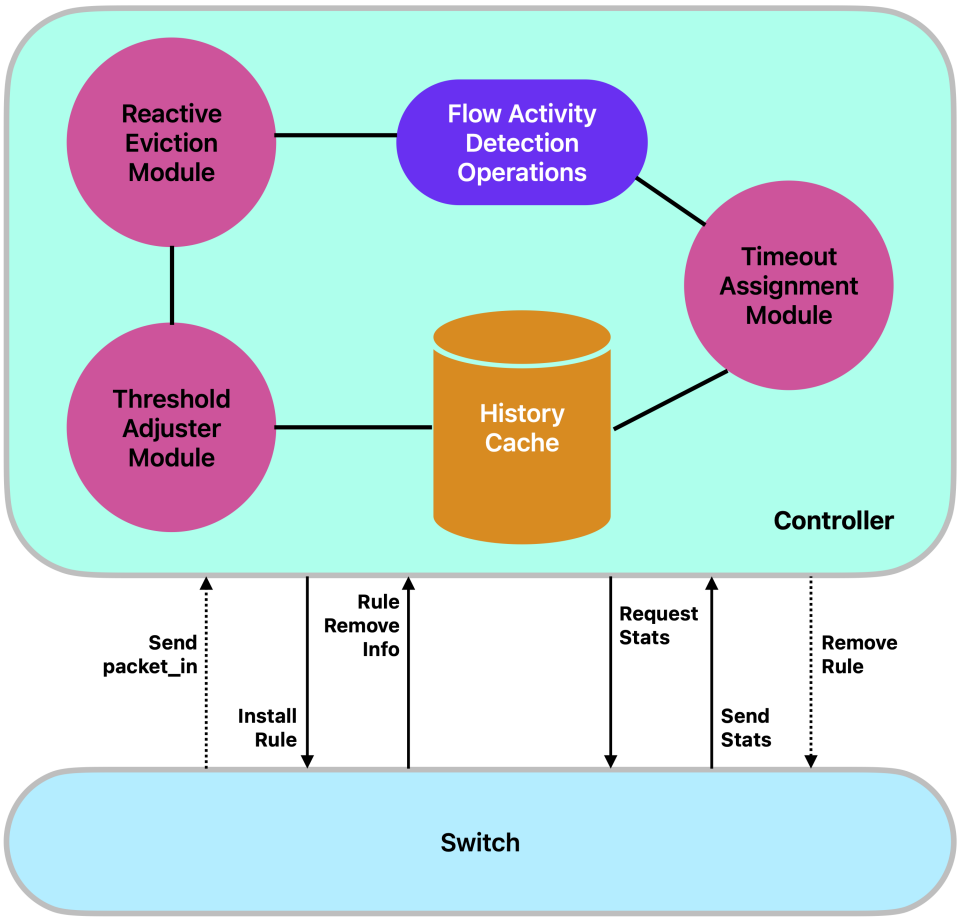


Figure 3.1. Architecture of ATEO.

Overall, ATEO's reactive modules work together to increase the effectiveness of the system as seen in Figure 3.1. They reinforce each other's effects and ensure that they all adapt to the changes in the network conditions. In this chapter, these modules are explained in detail.

3.1. Dynamic Threshold Adjustment Based on Rejected Flows

In SDN, the increase in rejected flows may cause service disruptions due to packet loss. Minimizing them ensures efficient flow table management, optimal resource use, and improved QoS. To avoid causing rejected flows, the system should be able to react to various conditions. When the flow table is on the verge of being full, the system should evict some flow entries. Using a fixed threshold that initiates the eviction process when the current flow table occupancy exceeds this threshold is a simple solution, but it does not account for varying network situations. Traffic conditions can change regularly, and optimizing the fixed threshold by hand is not feasible. The dynamic threshold adjustment mechanism adjusts the flow table's thresholds by auditing the changes in the number of rejected flows while updating the scaling factor according to the history cache of delta rejected flow numbers every *eviction_interval* seconds. The update process for the scaling factor helps to react to network traffic faster. For instance, when the current number of changes in the rejected flows is higher than the previously known highest number of rejected flows' changes in a turn that is kept inside the history cache, the scaling factor is updated to its highest possible value to give the system a time to adjust to the network's packed traffic. However, if there is no increase in the number of rejected flows, the mechanism assigns the highest possible threshold values to make sure that the flow tables are utilized efficiently.

Monitoring the changes in the number of rejected flows and adjusting thresholds according to these changes in real time plays a huge role in sustaining the network performance under different traffic patterns. In addition, the system is capable of preventing unnecessary flow evictions and optimizing the handling of existing ones by applying various rules during the timeout assignment process. The continual observation

loop enhances stability by understanding the current traffic patterns and maintaining an updated scaling factor for finer granularity in generating response.

Algorithm 1 Dynamic Threshold Adjustment Mechanism.

Require: current_rejected_flows, prev_rejected_flows, eviction_high_threshold, Δ, history_cache

Ensure: eviction_low_threshold, timeout_high_threshold, timeout_low_threshold

1: delta_rejected_flow ← current_rejected_flows − prev_rejected_flows
2: Append delta_rejected_flow to history_cache
3: max_rejected ← MAX(history_cache)
4: **if** max_rejected > 0 **then**
5: ratio_rejected ← delta_rejected_flow / max_rejected
6: **else**
7: ratio_rejected ← 0
8: **end if**
9: **if** delta_rejected_flow > 0 **then**
10: new_delta ← Δ + Δ × MIN(1, ratio_rejected)
11: **else**
12: new_delta ← Δ
13: **end if**
14: eviction_low_threshold ← eviction_high_threshold − (2× new_delta)
15: timeout_high_threshold ← eviction_high_threshold − new_delta
16: timeout_low_threshold ← eviction_high_threshold − (3× new_delta)
17: **return** eviction_low_threshold, timeout_high_threshold, timeout_low_threshold

In the beginning, the delta of the rejected flow number by comparing the total number of flow numbers of the previous loop with the current one is calculated using this equation ($delta_rejected_flow = current_rejected_flows - prev_rejected_flows$) and then appends this obtained value to the history cache to keep track of rejected flows' deltas (lines 1-2). The algorithm 1 obtains the known maximum number of rejected flows from this history cache (*max_rejected*) and calculates a ratio (lines 3-8), $ratio_rejected = \frac{delta_rejected_flow}{max_rejected}$, scaled by a predefined constant, Δ. If *delta_rejected_flow* is positive, ($new_delta = \Delta + \Delta * min(1, ratio_rejected)$); if it is negative, $new_delta = \Delta$ (lines 9-13). Thresholds are formulated as follows: the eviction's high threshold remains unchanged, the eviction's low threshold is calculated as $eviction_high_threshold - (2 \times new_delta)$, the timeout's high threshold is formulated as $eviction_high_threshold -$

new_delta, and the timeout's low threshold is found with *eviction_high_threshold* $-$ ($3 \times$ *new_delta*) (lines 14-16). These updates enable precise and adaptive control of resource management based on real-time rejection trends.

This formulation enables the system to adjust its thresholds based on the live network conditions instead of dictating static threshold values. Aggressive eviction of flows or maximization of table utilization, whichever is more appropriate at the time is permitted by the flexibility of the thresholds. Further, the presence of the history cache that keeps track of the previous changes in rejected flow numbers helps to reduce the risk of overreaction to temporary traffic spikes.

Overall, this mechanism delivers a solution to achieve adaptive design to deal with different traffic patterns in real-time by having a reactive threshold adjustment algorithm to keep the number of packet-in and rejected flow.

3.2. Z-Score Detection for Elephant Flows and Heuristic Importance Calculation

Around 80% of the internet traffic is carried out by only a small number of connections labeled as elephants, while the remaining large number of connections are labeled as mice and are small in size or lifetime [29] [30]. To carry most of the internet traffic, elephant flows have high traffic volume and persistence over time [31]. On the other hand, mice flows have the majority of packet flows with their short-lived connections over the internet [32]. Thus, accidental removal of the elephant flow rule from the table can lead to higher packet-in counts due to the nature of elephant flows. The Z-Score mechanism is dedicated to identifying elephant flows, which are network flows with significantly higher activity compared to mice flows. The activity is based on the number of hit counts and byte counts. The flow activity metrics are collected in every *monitoring_interval*. If the flow has been seen in the system before, the statistics of the previous occurrence of the flow are added to the dataset called *z_score_data*. The Z-Score mechanism that is presented in Algorithm 2 is triggered every minute

to process the data that are currently inside *z_score_data*. The flow's both Z-Score values *z_score_hits* and *z_score_bytes* are calculated during this operation using these formulas $z_score_hits = \frac{hit_count - avg_hit_count}{std_dev_hits}$ and $z_score_bytes = \frac{byte_count - avg_byte_count}{std_dev_bytes}$ (lines 2-3). *avg_hit_count*, *std_dev_hits*, *avg_byte_count*, and *std_dev_bytes* respectively represent the average hit count, standard deviation of hit counts, average byte count, and standard deviation of byte counts for all flows within *z_score_data*. A flow is only classified as an elephant when these criteria are met: the flow's hit count and byte count exceed their respective averages and its Z-Scores for both metrics surpass *z_threshold* (lines 4-5).

Algorithm 2 Z-Score Classification.

Require: avg_hit_count, avg_byte_count, std_dev_hits, std_dev_bytes, z_threshold
Ensure: z_score_data

1: **for** each flow f in z_score_data **do**
2: $z_score_hits \leftarrow \frac{(hit_count) - (avg_hit_count)}{std_dev_hits}$
3: $z_score_bytes \leftarrow \frac{(byte_count) - (avg_byte_count)}{std_dev_bytes}$
4: **if** $z_score_hits > z_threshold$ and $z_score_bytes > z_threshold$ **then**
5: f is Elephant
6: **else**
7: f is Mice
8: **end if**
9: **end for**

The usage of two different criteria enables the Z-Score mechanism to minimize the number of false positives for elephant flow detection. Even a flow somehow manages to have higher hit counts, if it doesn't have a high byte size, it's not going to be labeled as an elephant, and vice versa. The higher bandwidth and processing resource usage nature of elephant flows makes the detection of them crucial in SDN environments. By labeling them correctly, the prevention of higher packet-in numbers and rejected flows can be achieved.

The heuristic importance function in Algorithm 3 provides smart tips for a controller to prioritize the flows based on their nature. For elephant flows, a higher number of hit counts and a large amount of byte sizes justify their retention in the flow table.

Algorithm 3 Heuristic Importance Calculation.

Require: flow_table
Ensure: min_heap of (flow, heuristic_importance)

for each flow f in flow_table **do**
  if f is Elephant **then**
    $heuristic_importance \leftarrow \frac{total_hits}{duration}$
  else ▷ Mice flow
    $timeout_duration_ratio \leftarrow \frac{idle_timeout}{duration}$
    $heuristic_importance \leftarrow \frac{total_hits}{timeout_duration_ratio}$
  end if
  Insert (f, $heuristic_importance$) into min_heap
end for
return min_heap

The eviction of elephant flows may cause a higher number of packet-in and rejected flows as discussed before. In addition, the usage of the ratio of total hit count to the duration of the current flow that is under examination, the system captures the intensity of this flow and decides whether it's eligible to stay inside during the eviction process.

In contrast, the longevity of the mice flows is rewarded rather than their impact on the system. If a mouse flow manages to stay active for longer than its assigned idle timeout, it is counted as relevant and the system tries to keep it inside the flow table as long as possible. The relevancy of it increases when the mouse flow stays longer inside the flow table. This logic effectively gives it credit for surviving potential eviction cycles. A differentiation like this aligns with the intention to minimize the number of packet-in and rejected flows.

Likewise, the usage of a min-heap to store flows and their heuristic importance allows the SDN controller to effectively determine which flows should be removed first during reactive eviction. Constant access is possible with this data structure when getting the least important flow at any time, which is pretty crucial while the system is dealing with a heavy traffic load. The mixture of both Z-score-based classification and this heuristic mechanism builds a great foundation for data-driven eviction strategies

that are responsive to different network traffic loads.

3.3. Reactive Eviction

The reactive eviction mechanism is designed to manage the occupancy of the flow table by evicting flows based on *heuristic_importance* prioritization. When the current occupancy is higher than the *eviction_low_threshold*, *Heuristic Importance Calculation* function is called in every eviction loop interval to populate the *min_heap* which guides the eviction of flows. The flows with the lowest *heuristic_importance* values are the first ones evicted during this process. The eviction process is triggered when the flow table's occupancy exceeds the *eviction_high_threshold* and enters a loop that repeatedly evicts flows. The evicted flow is removed from the flow table and the occupancy level of the flow table is updated during this operation. When the new flow table occupancy falls below the *eviction_low_threshold*, eviction operation stops to prevent removing too many flows which leads to a higher number of packet-ins.

Algorithm 4 Reactive Eviction Mechanism.

Require: flow_table, eviction_low_threshold, eviction_high_threshold
Ensure: Updated flow_table with evicted low-priority flows

```
1: if flow_table_occupancy > eviction_low_threshold then
2:     min_heap ← HeuristicImportanceCalculation(flow_table)
3:     if flow_table_occupancy > eviction_high_threshold then
4:         while flow_table_occupancy > eviction_low_threshold do
5:             (f) ← ExtractMin(min_heap)
6:             Remove f from flow_table
7:             Update flow_table_occupancy
8:         end while
9:     end if
10: end if
11: return flow_table
```

This mechanism enables the system to maintain a balance between performance and availability inside the flow table by removing the less important flows in a prioritized manner. The reactive eviction algorithm can anticipate the potential overflows

by using the threshold configurations. This increases the controller's responsiveness to new flow arrivals by creating space for more important flows.

The usage of *heuristic_importance* as a metric for reactive eviction enables the system to keep elephant flows inside the flow table as long as possible while evicting the least important mice flows to keep the table available for the newcomers. This logic improves the quality of the whole system by keeping the flow table unclogged, especially at times when the network is heavily loaded.

Overall, the reactive eviction mechanism works with both thresholds and flow importance. The heuristic calculation function's regular execution ensures that the prioritization of flows remains up-to-date with recent network activity without overwhelming the controller, and the newly adjusted threshold values are used to start reactive eviction operation just in time. The combination of these mechanisms leads to the desired availability inside the flow table while keeping the number of packet-in and reject flows down as much as possible.

3.4. Timeout Assignment

The algorithm in Algorithm 5 implements a practical dynamic timeout mechanism. The main goal of this mechanism is to assign lower timeout values to insignificant flows, and higher timeout values to important flows. The calculation of the importance of the flows is mentioned in section 3.2 as the flow activity. Higher activity means higher importance. Likewise, the occupancy of the flow table dictates the timeout assignment process. For instance, when the table occupancy is high, the new timeout values tend to become lower than the previously assigned to prevent the flow table from overflowing. On the other hand, when the occupancy of the table is low, the trend of the new timeout values increases to keep the number of packet-in counts under control. If packets stay longer inside the flow table, packet-in counts get lower because when the same type of flow arrives in the system, the flow table will already have that rule in it and act on it accordingly. No need to visit the controller to get the rule for that flow.

Previously unseen flows are assigned the initial idle timeout (lines 1-4). The newcomer flows are granted base-level participation in the flow table by these lines to ensure they are not prematurely evicted before their behavioral pattern is analyzed. When the flow table occupancy is low, the algorithm reserves *t_max* as a timeout for elephant flows and increases the timeouts logarithmically according to the last known *total_hits* number of every mouse flow (lines 5-11). This increase enables the mice flows that have consistent traffic patterns to gain more persistence inside the flow table gradually by preventing quick timeout spikes. With this, frequently seen mice flows can stay longer on the table. As a result, mice flows that are considered recurring can avoid re-installation via packet-ins which leads to higher controller load and less efficiency overall in the system.

During the increase of table occupancy, *t_max* decays exponentially according to the current *table_occupancy* which leads to lower timeout values for elephant flows, and a similar decay approach is held for mice flows by decreasing their last assigned timeout values with *decay_factor* (lines 12-20). Degrading the lifespan of both flow types as the need for resources increases, this adaptive design provides a balance between holding important flows inside and freeing up some space when needed. In addition, the exponential decay enables the system to respond effectively to rapid changes in traffic changes to ensure that the adjustments for timeouts remain proportional to the harshness of the congestion.

During high-occupancy scenarios, the algorithm starts giving elephant flows *minimum_t_max* and mice *minimum_timeout* clearing the table aggressively to avoid an increase in the number of rejected flows (lines 21-28). The idea behind this strategy is to create a safeguard against sudden traffic bursts that might lead to overflow of the flow table which causes rejected flows that have the potential to disrupt the packet communication. The strategy of the aggressive cleanup under the high-pressure creates the needed space for newcomer flows that have the potential to be more important flows than the evicted ones to sustain the overall network effectiveness. Moreover, this last-resort logic is critical for maintaining the stability of the flow table and preventing

the bottlenecks caused by saturation.

Algorithm 5 Dynamic Timeout Assignment.

Require: total_hits, table_occupancy, flow_type, t_init, t_max, timeout_low_threshold, timeout_high_threshold, b_value, minimum_tmax, minimum_timeout, tmax_restore_value, last_timeout

Ensure: predicted_timeout

if flow_not_seen_before **then**
  $predicted_timeout \leftarrow t_init$
  return $predicted_timeout$
end if
if $table_occupancy \leq timeout_low_threshold$ **then**
  $t_max \leftarrow tmax_restore_value$ ▷ Restore t_max when occupancy is low
  if $flow_type =$ *"elephant"* **then**
    $predicted_timeout \leftarrow t_max$
  else
    $predicted_timeout \leftarrow \min(t_init \times \log_e(1 + total_hits), t_max)$
  end if
else if $timeout_low_threshold < table_occupancy \leq timeout_high_threshold$ **then**
  $\lambda \leftarrow e^{-b_value \times ((table_occupancy) - (timeout_low_threshold))}$
  $t_max \leftarrow \max(t_max \times \lambda, minimum_tmax)$
  if $flow_type =$ *"elephant"* **then**
    $predicted_timeout \leftarrow t_max$
  else
    $predicted_timeout \leftarrow \max(last_timeout \times decay_factor, minimum_tmax)$ ▷ Flow's previous timeout is $last_timeout$
    $predicted_timeout \leftarrow \min(predicted_timeout, t_max)$
  end if
else
  $t_max \leftarrow minimum_tmax$
  if $flow_type =$ *"elephant"* **then**
    $predicted_timeout \leftarrow t_max$
  else
    $predicted_timeout \leftarrow minimum_timeout$
  end if
end if
$idle_timeout \leftarrow predicted_timeout$
if $idle_timeout < minimum_timeout$ **then**
  $idle_timeout \leftarrow minimum_timeout$
end if
return $\text{int}(idle_timeout)$

4. RESULTS

In this section, we explore the details of the experimental setup, evaluation metrics, algorithm constants with their rationale, and the comparative analysis of certain algorithms to assess the effectiveness of our proposed approach, ATEO.

The experimental setup initiates the core of our empirical analysis and provides a comprehensive report about the hardware, software tools, and platforms used during the experiments. With this section, the performance analysis of the chosen algorithms can be reproduced. The evaluation metrics are used to asses the performance and behavior of the proposed algorithms. These metrics are used as primary indicators of their strengths, limitations, and suitability for the current problem domain. In addition, the section presents the algorithmic constants with their reasoning behind the choice of their specific values based on both theoretical and empirical tuning. Finally, the comparative analysis provides a side-by-side evaluation of the performance of proposed algorithms, spotlighting the most adequate algorithms under various circumstances.

4.1. Experimental Setup

The simulations are tested on Apple's M1 Chip running 4 GB RAM allocated UTM [33] to virtualize the Ubuntu 22.04 ARM version. To assess the performance of ATEO, we used the UNIV1 and UNIV2 network packet traces, which were collected from two university data centers as described in [34]. These datasets are commonly used in research studies, including our opponents [35] and [28]. The UNIV1 trace consists of 67.4% TCP flows, while UNIV2 predominantly contains UDP flows, with only 8.70% being TCP flows [36]. In the simulation process, we used the full UNIV1 trace and the first one-third subset of the UNIV2 trace as in [28]. Mininet [37] with the Ryu [38] controller (OpenFlow1.3) simulated network topologies with Python 3.9. The topology has one switch and seven hosts, where each host replays the same .pcap traffic file that comes from UNIV1 and UNIV2 to simulate network load as seen in Figure

4.1. Flow table sizes of 250, 500, 750, 1000, 1250, and 1500 entries were evaluated for UNIV1, while 500, 750, and 1000 were tested for UNIV2.

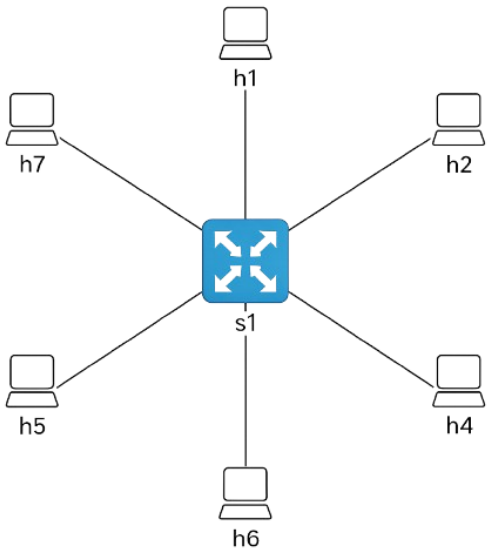


Figure 4.1. Topology of Simulation.

These table sizes were chosen to emulate the constraints of TCAM-limited switches in the real world and to create a realistic testing ground for evaluating the success of the algorithms under limited resources. By testing with different flow table sizes, we aim to show the adaptation ability of ATEO to resource levels. In addition, TCP-heavy and UDP-heavy traces were used to obtain the performance of the algorithms across different traffic characteristics. This diversity is crucial in seeing the performance of these algorithms under various network conditions. Our competitors are:

- *Fixed Timeout (3 s):* This only removes flows after a fixed idle period (3 seconds), unlike [39], which also assigns the 5-second hard timeout for their flow entries. Our simulations showed that the 3-second idle timeout alone provided better results as seen in Table 4.1.
- *VITARPEM [28]:* It efficiently utilizes dynamic timeout and eviction mechanisms which makes it a suitable competitor for benchmarking ATEO.
- *TSMM [27]:* To ensure the consistency in comparisons, this is added as in [28] as competitor.

Table 4.1. Comparison of 3-Second Idle Timeout vs. 3-Second Idle with 5-Second Hard Timeout Mechanisms (UNIV1, Table Size = 750).

Idle/Hard Timeout	*Packet-in Count*	*Rejected Flows*
3/X	410000	4500
3/5	8300000	20800

4.2. Evaluation Metrics

In SDN, the use of specific and meaningful metrics is needed during the evaluation of the performance of proposed mechanisms or architectures. These metrics are used to identify which proposed mechanism has better efficiency, scalability, and practicality under different network conditions. In this section, four primary evaluation metrics are discussed. These are commonly used to measure the performance of flow table management methods: Packet-In Count, Rejected Flows, Flow Table Utilization, and Computation Overhead. Each of these metrics provides valuable insights into how well the controller is designed and how efficiently the overall network is operating.

4.2.1. Packet-In Count

Switches send a message to the controller when they encounter a packet that does not match any existing flow entry inside their flow table. This message is called a packet-in message. This message is one of the core functions of SDN, but the poor usage of these messages can lead to overhead on the controller. An excessive number of packet-in messages may cause inadequate flow rule installation strategies and poor scalability. A high packet-in count results in increased communication between the controller and the switches, which introduces additional latency and burden to both the controller and the switch with a lot of tasks to process. The network's overall responsiveness may degrade due to this. Consequently, minimizing the packet-in count is generally desirable. The lower the packet-in count, the more independent the switches are. When the flow table has flow entries that are currently in use, the number of packet-ins tends to stay lower.

4.2.2. Rejected Flows

This metric represents the number of flows dropped during the installation process due to the flow table's occupancy at its maximum level. When the number of rejected flows is high, the overall performance of the network can be seen as low. The system is either overloaded or lacks an efficient flow table management mechanism, or both at the same time, in that case. Such rejections lead to packet drops, affect the QoS in a harmful way, and degrade the overall reliability of the network. This metric gives insights into how effectively the controller's logic is designed, as well as the utilization of the flow table's success. Ideally, the number of rejected flows should be close to zero to ensure uninterrupted service delivery in an SDN system.

4.2.3. Flow Table Utilization

This metric is used to see the utilization of the limited memory space of a switch's flow table. Due to the expensive nature of TCAMs, the SDN switches have restricted flow table capacities. This forces SDN mechanisms to use the available space in the flow table efficiently. The capability of the system to keep the utilization of the flow table as high as possible while causing almost zero rejected flows and lower packet-in message numbers is counted as effective. Making the most out of the current hardware while keeping the other metrics on their optimal levels is a crucial challenge that every flow table management mechanism tries to achieve. This success contributes to the overall performance of the system. On the other hand, an underutilized flow table may lead to degraded network performance by having higher packet-in counts and computational overhead in the controller that comes with the packet-in messages. Therefore, maintaining a high flow table utilization while keeping the packet-in count and rejected flow number low is a critical task.

4.2.4. Computation Overhead

Computation overhead refers to the processing resources consumed mostly by the controller and other network elements when executing the flow table management mechanisms. A high computational load may slow down the timeout assignment process and introduce network delays. The controller can work faster when the overhead is reduced. This metric depends especially on the timeout assignment module itself, and packet-in messages come from switches in this study. If the flow entries were not installed with near-optimal configurations in the flow table, they might lead to more packet-in counts. In the end, the timeout assignment module has to operate for the same flow over and over again. Evaluating this metric shows whether proposed SDN solutions achieved practical and performant results under realistic network conditions.

- *Packet-In Count:* A high count of packet-in messages shows that there is frequent communication between the controller and switches. This may lead to control plane overhead. Fewer is better.
- *Rejected Flows:* When the installation of a flow rule fails, rejected flow occurs. This leads to lower QoS. Fewer is better.
- *Flow Table Utilization:* Efficient use of the switch's flow table minimizes unnecessary controller interaction. Higher is better.
- *Computation Overhead:* High computational overhead can slow down calculations inside the controller. Fewer is better.

4.3. Algorithmic Constants and Rationale

ATEO uses specific constants in its threshold adjustment, eviction, and timeout assignment logic. We chose the constants and rationale based on empirical experiments and standard practices in SDN-based flow management.

- *monitoring_interval (5 s)*: Checking more frequently increases the load on switches and the controller, whereas checking less frequently can result in less accurate flow

metrics. If used hardware does not have problems running at lower intervals, opt for lower intervals.

- *eviction_interval (2 s)*: Shorter intervals increase controller load, while longer intervals can cause delayed evictions and more rejected flows. If used hardware does not have problems running at lower intervals, opt for lower intervals.
- *Δ (10%)*: A moderate value that enables measured adaptation without inducing excessive oscillation. The tested values (5, 10, and 15) show little or no difference in terms of performance as seen in Table 4.2. The table size of 500 is chosen for this experiment because it leads to a high enough packet-in count and rejected flows. The results show little to no difference, and one can choose other tested values.
- *eviction_high_threshold (90%)*: A fixed high threshold was chosen around 90-95% in [25] [21] [27]. Using lower values than these increased packet-in counts but yielded little to no reduction in rejected flows.
- *t_init = minimum_t_max (11 s)*: Theoretically chosen as sufficiently long to avoid constant re-installations, yet short enough to allow reclaiming if unused. In line with [26], which found that 88% of the flows had packet intervals of less than 11 seconds, which makes this reasonable for both minimum max and initial value.
- *t_max (32 s)*: Serves as the limit for idle timeout in seconds, as used in previous work [28].
- *b_value (2)*: Among tested values (1, 2, and 3), this provided the best balance, keeping the packet-in count lower, as seen in Table 4.3. The table size of 1000 is chosen for this experiment because it does not cause rejected flows and fills the table enough for the usage of this value. The results show that 2 is a bit better than 1.
- *minimum_timeout (1 s)*: In line with [26], which found that 76% of the flows had packet intervals of less than 1 second, making this a reasonable minimum.
- *z_threshold (2)*: Based on the statistical standard in [40], where 95% of values lie within ± 2 standard deviations, helping segregate elephant flows effectively.

Table 4.2. Performance for Varying Δ (First 10% of UNIV1, Table Size = 500).

Δ	*Packet-in Count*	*Rejected Flows*
5%	519000	2800
10%	516000	3000
15%	535000	2400

Table 4.3. Performance for Varying *b_value* (First 10% of UNIV1, Table Size = 1000).

b-value	*Packet-in Count*	*Rejected Flows*
1	14300	0
2	13700	0
3	19800	0

4.4. Comparative Analysis

In this section, the comparison between ATEO and competitor flow management algorithms is presented comprehensively. These results highlight both the advantages and disadvantages of each solution while examining their performance under different flow table entry sizes. The crucial performance metrics are packet-in counts, the total number of rejected flows, and the utilization of the flow table. All of these metrics are used to evaluate the adaptability and efficiency of these solutions. In addition, these metrics provide insight into how each algorithm performs the flow table management under varying traffic conditions. Further, CPU and memory usage levels are assessed to see the computational overhead differences between the algorithms. By comparing ATEO's performance against other algorithms in different scenarios, the evaluation aims to show ATEO's ability to maintain competitive controller overhead while achieving lower packet-in count and rejected flows. With this analysis, the strengths of ATEO are made evident in contrast to more narrowly focused solutions.

Figure 4.2 and 4.3 shows that ATEO manages to decrease the number of packet-ins across different table sizes, which helps to reduce controller overhead in the control plane. In the UNIV1 dataset, ATEO accomplishes noteworthy reductions of up to

76.19% in packet-in counts compared to TSMM and *58.33%* compared to VITARPEM for flow table sizes 750 and 1000. Lower packet-in counts might contribute to energy savings and better utilization of controller resources due to the lower number of operations required to keep the system up. In addition, ATEO's performance consistency across different table sizes presents robustness in various real-world SDN environments.

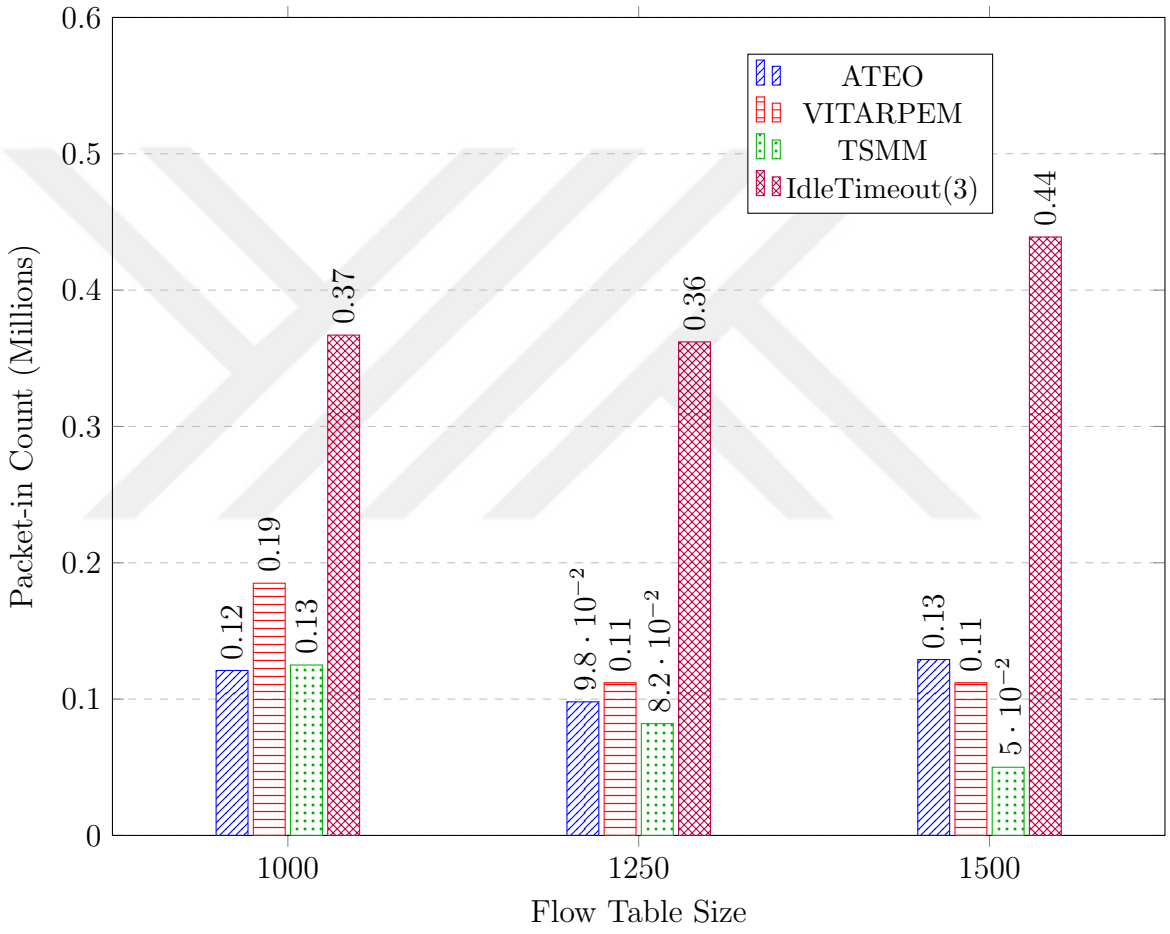


Figure 4.2. Packet-in Counts at 1000, 1250, and 1500 Table Entry Sizes (UNIV1).

On the other hand, ATEO produces a higher packet-in count at a flow table size of 500 and performs worse than its competitors. However, ATEO shows more resilience in extra resource-constrained scenarios. At a flow table size of 250 entries, ATEO's packet-in count is only *7.82%* higher than the best result achieved by VITARPEM, making ATEO a competitive algorithm again under such restricted conditions. These results indicate that ATEO may exhibit some inefficiencies at moderate table sizes regarding packet-in count. However, it compensates with its competitive performance

under more severe resource constraints. You will see in the ongoing results how ATEO compensates for the slight lack of performance in packet-in counts compared to its rival mechanisms. The effectiveness and adaptability of ATEO are further reinforced by its remarkably lower rejected flow numbers, as shown in the rejected flow results section.

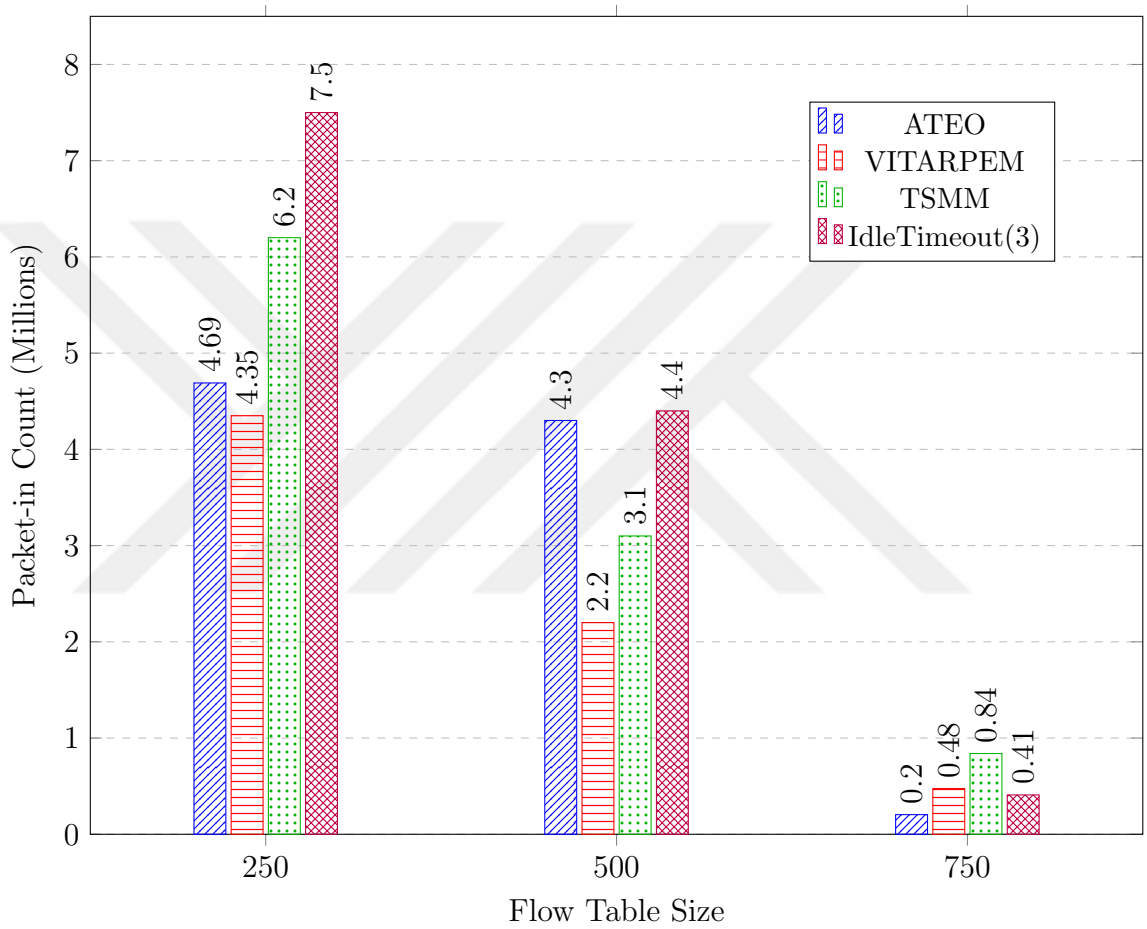


Figure 4.3. Packet-in Counts at 250, 500, and 750 Table Entry Sizes (UNIV1).

For the UNIV2 dataset, ATEO keeps performing better by achieving a lower number of packet-in counts *56.76%* and *54.29%* compared to VITARPEM and TSMM, respectively, for the flow table size of 750. Even though ATEO leads to slightly higher packet-in counts for the flow table size of 1000, its overall performance remains competitive as shown in Figure 4.4.

The most prosperous part of ATEO is reducing the number of rejected flows. In the UNIV1 dataset for a table size of 500 entries, ATEO achieves up to *99.9%* fewer rejections than TSMM and *99.78%* than VITARPEM, see Figure 4.5 and 4.6.

A similar trend is observed with a table size of 250 entries. ATEO achieves *98.92%* fewer rejections than its most capable competitor VITARPEM. Compared to TSMM, ATEO produces *99.24%* fewer rejected flows.

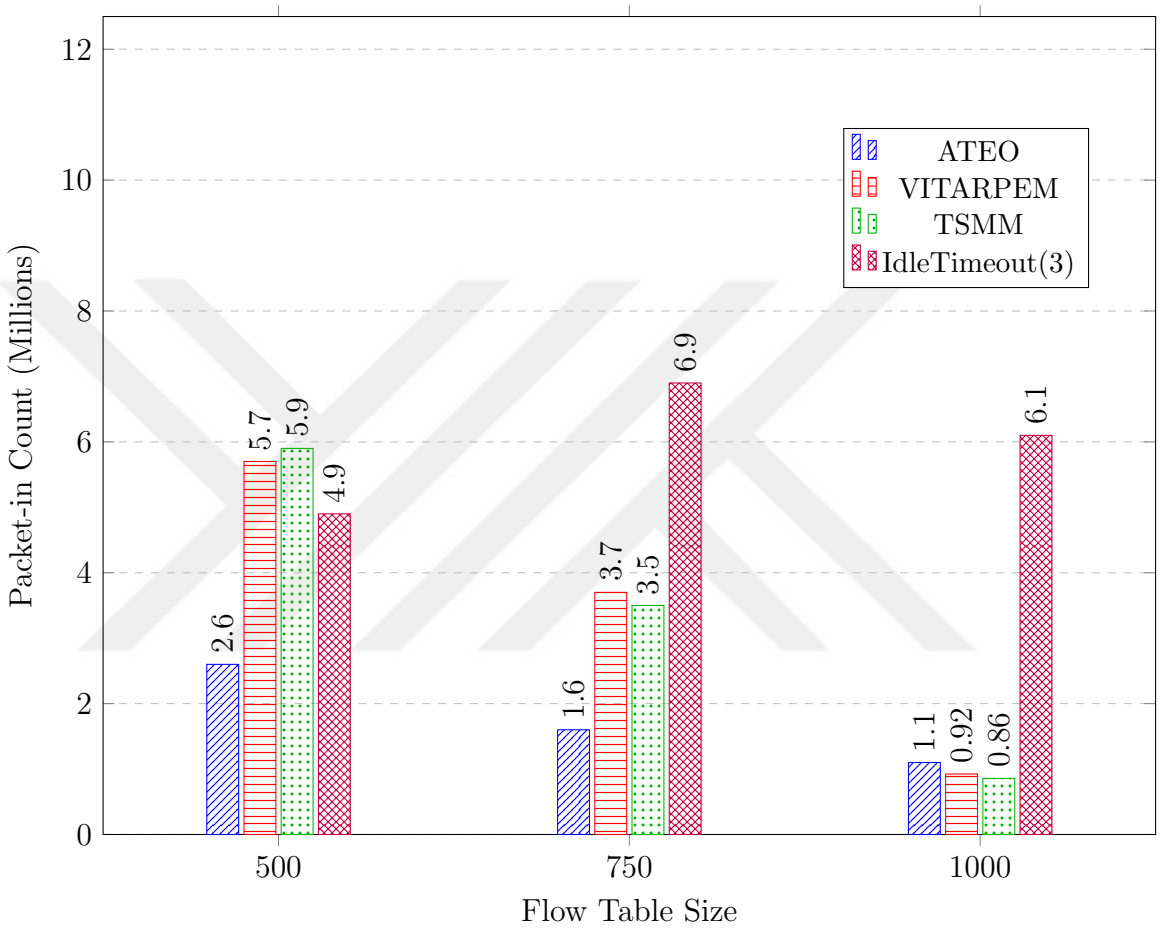


Figure 4.4. Packet-in Counts at 500, 750, and 1000 Table Entry Sizes (UNIV2).

However, a table size of 750 entries shows smaller differences between algorithms, but ATEO still achieves the best result by a significant margin. The 750-entry table size indicates that we are approaching the optimal flow table size for these algorithms, since all the results are approaching 0. With a table size of 1000 entries, only TSMM has rejected flows. In contrast, table sizes of 1250 and 1500 entries show no differences between the algorithms in terms of rejected flows, with all results being 0. Thus, finding which algorithm performs better when using these table sizes is not possible.

In the UNIV2 dataset for a table size of 750 entries (one-third of the dataset, see Figure 4.7), ATEO preserves its superiority by rejecting *99.2%* fewer flows than

TSMM and *95.7%* fewer than VITARPEM. For the 1000-entry table size, ATEO still performs better than its competitors. It achieves up to *97.76%* fewer rejected flow numbers compared to TSMM and *34.8%* than VITARPEM. These results highlight ATEO's ability to maintain competitive performance without causing excessive flow drops when near-optimal resources are given to the system. On the other hand, under resource-constrained systems, ATEO performs significantly better than its competitors.

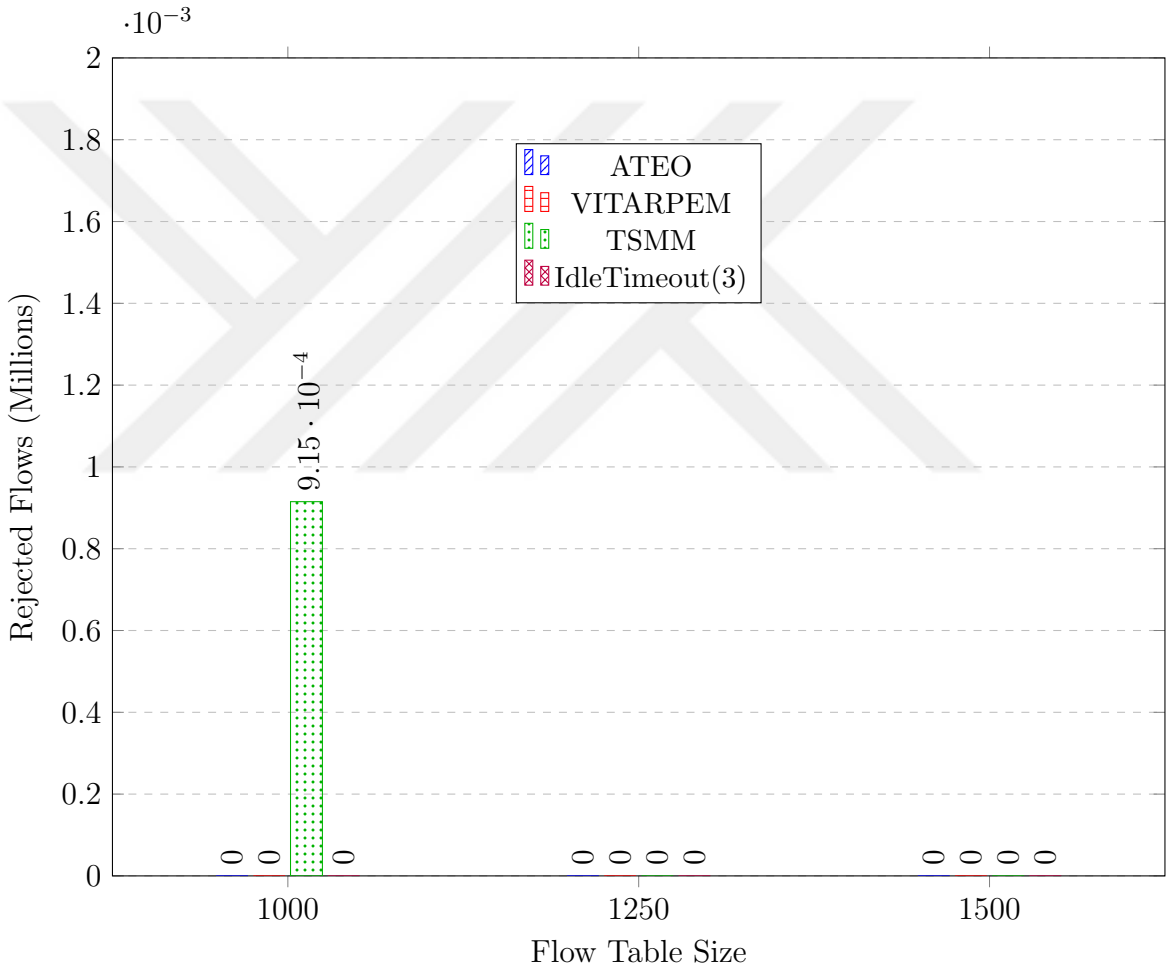


Figure 4.5. Rejected Flows at 1000, 1250, and 1500 Table Entry Sizes (UNIV1).

Evaluation of flow table utilization rates in UNIV1 and UNIV2 datasets shows a moderate performance of ATEO; ATEO achieves a utilization rate in the range of *74%-50%* similar to VITARPEM, which achieves *75%-66%* utilization rate. The fixed idle timeout mechanism with a *78%-57%* utilization rate and TSMM achieves a higher utilization rate of *85%-77%*, which is the better flow table utilization across other algorithms. Although it manages to utilize the flow table better, the number of

packet-ins and rejected flows is too high compared to ATEO and VITARPEM.

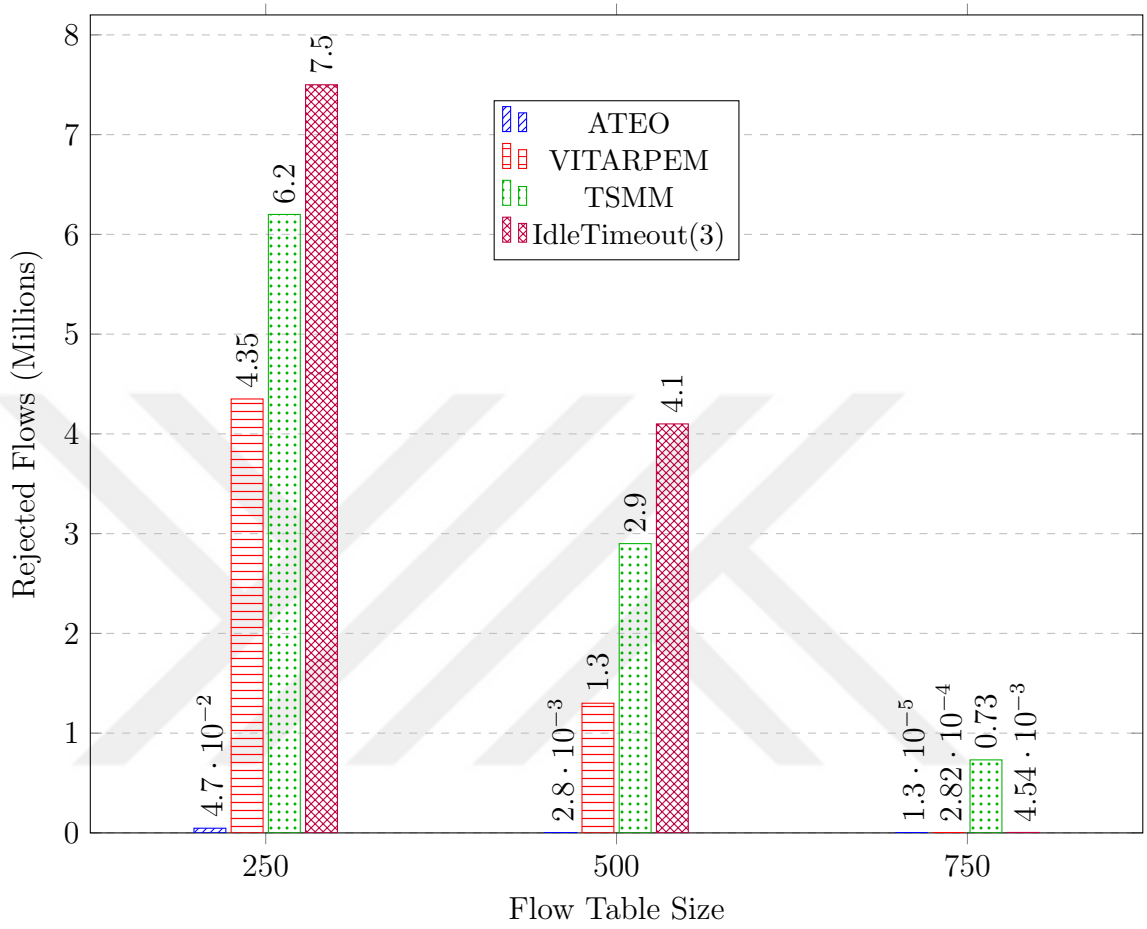


Figure 4.6. Rejected Flows at 250, 500, and 750 Table Entry Sizes (UNIV1).

The average CPU usages of VITARPEM and TSMM are *12.13%* and *12.03%* respectively, while ATEO is falling a bit behind every other algorithm with *12.19%* CPU usage. On the other hand, the fixed idle timeout uses a slightly lower CPU compared to TSMM at *11.79%*. Regarding memory usage, VITARPEM uses the most across other algorithms with *15.33%*. ATEO follows VITARPEM with slightly lower memory usage *13.78%* while the fixed idle timeout mechanism operates at *12.77%*. TSMM, with a memory usage of *13.19%*, shows comparable memory usage to the fixed idle timeout mechanism. Such a small difference in both CPU and memory utilization is not significant.

ATEO has a better overall performance in almost all metrics compared to its closest competitors, such as VITARPEM and TSMM. ATEO reduces the number of

rejected flows in lower flow table sizes and packet-in counts impressively compared to TSMM and the fixed timeout mechanism in both datasets and different flow table sizes, thanks to its reactive eviction algorithm and threshold adjuster working in tandem. Although VITARPEM achieves a slightly lower packet-in count in certain scenarios, it suffers from significantly higher rejected flows in lower table sizes, making ATEO the more reliable and balanced solution for different resource configurations and network traffic conditions. These results prove that ATEO has an outstanding ability to adapt to different traffic patterns as well as resource variants by providing better QoS to the system by lowering the number of rejected flows and reduced controller overhead in various scenarios by decreasing the packet-in count.

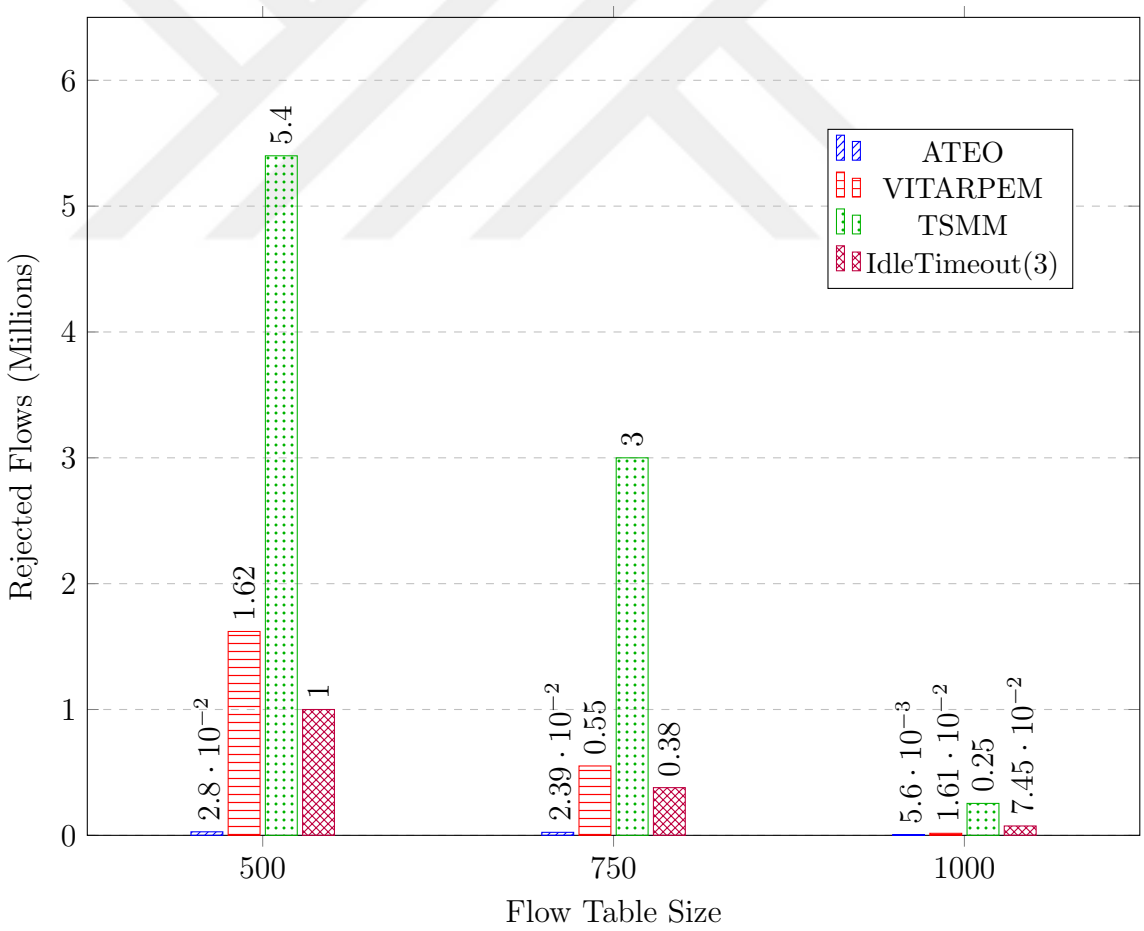


Figure 4.7. Rejected Flows at 500, 750, and 1000 Table Entry Sizes (UNIV2).

5. CONCLUSION AND FUTURE WORK

In scenarios when the current network handles critical information, keeping the number of rejected flows is crucial. ATEO, our proposed solution for flow table management in SDN, addresses these issues through its threshold adjuster, heuristic-based eviction mechanism, and dynamic timeout assignment module. The approach of ATEO helps the system keep the number of rejected flows near zero while keeping the consumption of resources competitive with its opponents. Even though ATEO leads to a slightly higher packet-in count under limited flow table capacity, this trade-off is justified by the dramatic reduction in rejected flows, which makes ATEO suitable for mission-critical systems. The dramatic reduction in rejected flows justify the slightly higher packet-in count under limited flow table capacity for ATEO.

Likewise, the modular design of ATEO provides seamless integration with existing SDN architectures. It also enables flexibility and scalability as network needs evolve. The less computational power-hungry design also makes ATEO viable for constrained environments.

In the future, to augment the capabilities of ATEO, machine learning techniques can be applied to it. Dynamic timeout assignment can be built on top of real-time machine learning mechanisms which can enable the more precise and more network traffic-oriented timeouts for flows. In addition, ATEO can be adapted for cases like satellite and IoT networks. By addressing different scenarios, we aim to extend the applicability of ATEO.

These improvements not only open doors for self-adaptive flow management systems but also assure that ATEO remains relevant as networking paradigms shift toward AI-driven models. Finally, ATEO lays a solid foundation for future research.

REFERENCES

1. Sloane, T., "Software-Defined Networking: The New Norm for Networks - Open Networking Foundation", Accessed on February 12, 2025.

2. Feamster, N., J. Rexford and E. Zegura, "The Road to SDN: an Intellectual History of Programmable Networks", *SIGCOMM Computer Communication Review*, Vol. 44, No. 2, p. 87–98, Apr. 2014.

3. Sandhya, Y. Sinha and K. Haribabu, "A Survey: Hybrid SDN", *Journal of Network and Computer Applications*, Vol. 100, pp. 35–55, 2017.

4. Isyaku, B., M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar and F. A. Ghaleb, "Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey", *Future Internet*, Vol. 12, No. 9, 2020.

5. Paliwal, M., D. Shrimankar and O. Tembhurne, "Controllers in SDN: A Review Report", *IEEE Access*, Vol. 6, pp. 36256–36270, 2018.

6. Kim, T., T. Lee, K.-H. Kim, H. Yeh and M. Hong, "An Efficient Packet Processing Protocol Based on Exchanging Messages Between Switches and Controller in OpenFlow Networks", *2013 10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)*, pp. 1–5, Melville, NY, USA, 2013.

7. Yang, T., A. X. Liu, Y. Shen, Q. Fu, D. Li and X. Li, "Fast OpenFlow Table Lookup with Fast Update", *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 2636–2644, Honolulu, Hawaii, USA, 2018.

8. Li, J., X. Ma, J. Zhang, J. Tao, P. Wang and X. Guan, "Mining Repeating Pattern in Packet Arrivals: Metrics, Models, and Applications", *Information Sciences*, Vol. 408, pp. 1–22, 2017.

9. Paliwal, M. and K. K. Nagwanshi, "Effective Flow Table Space Management Using Policy-Based Routing Approach in Hybrid SDN Network", *IEEE Access*, Vol. 10, pp. 59806–59820, 2022.

10. Ullah, Z., M. K. Jaiswal, R. C. Cheung and H. K. So, "UE-TCAM: an Ultra Efficient SRAM-based TCAM", *TENCON 2015-2015 IEEE Region 10 Conference*, pp. 1–6, IEEE, Macao, China, 2015.

11. Banerjee, S. and K. Kannan, "Tag-In-Tag: Efficient Flow Table Management in SDN Switches", *10th International Conference on Network and Service Management (CNSM) and Workshop*, pp. 109–117, Rio de Janeiro, Brazil, 2014.

12. Mimidis-Kentis, A., A. Pilimon, J. Soler, M. Berger and S. Ruepp, "A Novel Algorithm for Flow-Rule Placement in SDN Switches", *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 1–9, Paris, France, 2018.

13. Liang, H., P. Hong, J. Li and D. Ni, "Effective Idle_Timeout Value for Instant Messaging in Software Defined Networks", *2015 IEEE International Conference on Communication Workshop (ICCW)*, pp. 352–356, London, United Kingdom, 2015.

14. Isyaku, B., 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", *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 170–175, Penang, Malaysia, 2020.

15. Isyaku, B., K. B. Abu Bakar, F. A. Ghaleb and S. Sulaiman, "Performance Evaluation of Flowtable Eviction Mechanisms for Software Defined Networks Considering Traffic Flows Variabilities", *2022 IEEE 12th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 71–75, Penang, Malaysia, 2022.

16. Botta, A., A. Dainotti and A. Pescapé, "A Tool for the Generation of Realistic Net-

work Workload for Emerging Networking Scenarios", *Computer Networks*, Vol. 56, No. 15, pp. 3531–3547, 2012.

17. Gude, N., T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown and S. Shenker, "NOX: Towards an Operating System for Networks", *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 3, pp. 105–110, 2008.

18. Curtis, A. R., J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks", *Proceedings of the ACM SIGCOMM 2011 Conference*, pp. 254–265, Toronto, Ontario, Canada, 2011.

19. Kim, E.-D., Y. Choi, S.-I. Lee and H. J. Kim, "Enhanced Flow Table Management Scheme with an LRU-based Caching Algorithm for SDN", *IEEE Access*, Vol. 5, pp. 25555–25564, 2017.

20. Panda, A., S. S. Samal, A. K. Turuk, A. Panda and V. C. Venkatesh, "Dynamic Hard Timeout Based Flow Table Management in Openflow Enabled SDN", *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, pp. 1–6, Vellore, India, 2019.

21. Ma, Z., G. Zhao, C. Lin, H. Luo, Y. Zhong, S. Li, Q. Wu, Z. Mo and Z. Zhang, "The Research on Flow Table Optimization Based on Dynamic Timeout Mechanism", *2019 IEEE International Conference on Power Data Science (ICPDS)*, pp. 1–5, IEEE, Taizhou, China, 2019.

22. Zhang, L., S. Wang, S. Xu, R. Lin and H. Yu, "TimeoutX: An Adaptive Flow Table Management Method in Software Defined Networks", *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, San Diego, California, USA, 2015.

23. Zhang, L., R. Lin, S. Xu and S. Wang, "AHTM: Achieving Efficient Flow Table

Utilization in Software Defined Networks", *2014 IEEE Global Communications Conference*, pp. 1897–1902, Austin, Texas, USA, 2014.

24. Li, Q., N. Huang, D. Wang, X. Li, Y. Jiang and Z. Song, "HQTimer: A Hybrid Q -Learning-Based Timeout Mechanism in Software-Defined Networks", *IEEE Transactions on Network and Service Management*, Vol. 16, No. 1, pp. 153–166, 2019.

25. Vishnoi, A., 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*, DEBS '14, p. 177–188, Association for Computing Machinery, New York, NY, USA, 2014.

26. Zhu, H., H. Fan, X. Luo and Y. Jin, "Intelligent Timeout Master: Dynamic Timeout for SDN-based Data Centers", *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 734–737, Ottawa, Canada, 2015.

27. Li, T., H. Zhou, H. Luo, W. Quan, Q. Xu, G. Li and G. Li, "Timeout Strategy-based Mobility Management for Software Defined Satellite Networks", *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 319–324, IEEE, Atlanta, GA, USA, 2017.

28. Arslan, B., E. Kızılkaya, B. Yiğit and F. Alagöz, "Maximizing SDN Flow Table Efficiency with Dynamic Timeout Allocation and Proactive Eviction", *2024 15th International Conference on Network of the Future (NoF)*, pp. 10–18, IEEE, Castelldefels, Spain, 2024.

29. Guo, L. and I. Matta, "The War Between Mice and Elephants", *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pp. 180–188, Riverside, CA, USA, 2001.

30. Cheng, W., F. Ren, W. Jiang, K. Qian, T. Zhang and R. Shu, "Isolating Mice and Elephant in Data Centers", *arXiv preprint arXiv:1605.07732*, 2016.

31. Papagiannaki, K., N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian and C. Diot, "A Pragmatic Definition of Elephants in Internet Backbone Traffic", *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, IMW '02, p. 175–176, Association for Computing Machinery, New York, NY, USA, 2002.

32. Ajmone Marsan, M., G. Carofiglio, M. Garetto, P. Giaccone, E. Leonardi, E. Schiattarella and A. Tarello, "Of Mice and Models", M. Ajmone Marsan, G. Bianchi, M. Listanti and M. Meo (Editors), *Quality of Service in Multiservice IP Networks*, pp. 15–32, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

33. "UTM", `https://mac.getutm.app/`, accessed on January 22, 2025.

34. Benson, T., A. Akella and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild", *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, pp. 267–280, Melbourne, Australia, 2010.

35. Shen, Y., C. Wu, Q. Cheng and D. Kong, "AFTM: An Adaptive Flow Table Management Scheme for OpenFlow Switches", *2020 IEEE 22nd International Conference on High Performance Computing and Communications*, pp. 917–922, Fiji, 2020.

36. Yang, H., 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.

37. "Mininet", `https://mininet.org/`, accessed on January 22, 2025.

38. "Ryu SDN Framework", `https://ryu-sdn.org/`, accessed on January 22, 2025.

39. Isyaku, B., 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", *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 170–175, Penang, Malaysia, 2020.

40. Livingston, E. H., "The Mean and Standard Deviation: What Does It All Mean?", *Journal of Surgical Research*, Vol. 119, No. 2, pp. 117–123, 2004.