1.5em 0pt

CMPE 492

Begüm Arslan

Elif Kızılkaya

Advisor: Prof. Fatih Alagöz

January 14, 2024

TABLE OF CONTENTS

1. INTRODUCTION

Software-Defined Network (SDN) is an innovative network paradigm that decouples control and data planes. This separation enables programmable network operations, fostering rapid advancements in the field. Recently, SDN has been the focus of extensive research and investment, particularly in data center networks [4].
These networks require a communication protocol between software and hardware due to their separation. Generally, the OpenFlow protocol serves as the interface between control and data planes. A critical component of OpenFlow switches is the flow tables [14]. These tables contain information for each flow, dictating the processing and routing of associated packets. Flow tables are implemented using Ternary Content Addressable Memory (TCAM), which enables constant-time flow entry lookups (O(1)) regardless of table size [5]. However, TCAMs have limited memory size, typically accommodating only thousands of entries. This limitation is due to constraints in silicon size, expensiveness, and high power consumption [5]. Consequently, significantly expanding the flow table size is impractical and costly.
In such networks, each incoming packet is compared with existing table entries. If a match is found, the specified policy is applied; otherwise, a $packet-in$ message is sent to the controller for processing [14] and adding to the table. Table overflow occurs when the table is full, preventing new entries and resulting in dropped packets, namely rejected flows [1]. Efficient utilization of TCAM space is crucial to avoid negatively impacting network performance and overburdening the controller. There are two main strategies for this: setting appropriate timeout mechanisms and proactive eviction.
OpenFlow distinguishes between two types of timeouts: idle timeout and hard timeout. A hard timeout removes a flow after a specified time, whereas an idle timeout does so after a period of inactivity (no matching packets) [10]. Typically, fixed timeout values are assigned to all entries as flow table management strategies. However, this approach is suboptimal in SDN due to limited table sizes and varying flow characteristics. Large timeouts can lead to space wastage and potential table overflow, while small timeouts can trigger excessive $packet-in$ messages, straining the controller. Furthermore, fixed

timeouts do not account for table occupancy ratios. When occupancy is low, flow entries should remain longer to minimize table misses. Conversely, when occupancy is high, it is beneficial to remove the flows used infrequently or have been idle for a long time.

In OpenFlow, the controller can evict active flows. However, flow removal can degrade network performance and increase $packet - in$ messages. Therefore, selecting which flows to evict is crucial.

This paper proposes a dual approach, combining dynamic idle timeout adjustment with proactive eviction. When occupancy ratios are low, we rapidly increase idle timeouts utilizing the flow statistics; when occupancy is high, we cautiously increment timeouts based on flow statistics. We use a dynamic t_{max} to limit excessive table occupancy. Additionally, we calculate heuristic values for active flows to guide proactive eviction. Periodically, we assess table size and, upon reaching a high threshold, delete the flows with the lowest heuristic values until reaching a lower threshold for the table occupancy.

The report is structured as follows: Section III reviews related work, Sections IV and V detail our methodology and algorithm design, Section VI includes our requirements, Section VII covers our implementation and testing, Section VIII presents simulation results, and finally, Section IX concludes the report.

1.1. Broad Impact

In our project, we combine a dynamic idle timeout mechanism with proactive eviction in software-defined networks by introducing an innovative method. In this way, it helps to better data handling, resulting in a better and more stable network performance. For sectors like cloud services and online enterprises that depend on reliable Internet connections, this is very essential and useful. Additionally, by making better use of resources such as flow tables, our method can help network operators save money. Furthermore, by using less energy in big data centers, our solution can help the environment. Additionally, our method can be used in satellite networks, where reliable data handling and bandwidth management are crucial.

1.2. Ethical Considerations

In our project, where we worked on improving network performance in software-defined networks, we did not focus on ethical issues since it is beyond our project. However, we know these issues are important and understand that when you change how networks are managed, there can be concerns about keeping data safe and treating all users fairly.
Additionally, we used publicly shared data called UNIV1 in our project for testing purposes and gave references to each article we utilized throughout our project.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

In Software Defined Network (SDN), the OpenFlow protocol manages data flow using rules stored in limited-capacity Flow Tables. This capacity limitation, due to Ternary Content Addressable Memory(TCAM) constraints, can cause overflow problems, affecting the efficiency and reliability of SDN systems. Our project is creating a solution by handling a dynamic idle timeout mechanism with proactive eviction in software-defined networks.

2.2. Project Planning

2.2.1. Project Time and Resource Estimation

(i) **Weeks 1-2: Research and Paper Reading**
- Do an in-depth review of existing literature on SDN, OpenFlow, and network features.
- Scan various algorithms that perform idle timeout allocation or proactive eviction, taking into account flow characteristics or network restrictions.

(ii) **Weeks 3-5: Algorithm Design**
- Prepare an algorithm by taking dynamic timeout allocation and proactive eviction.

(iii) **Weeks 5: Establishing the simulation environment**
- Connect to Netlab computers.
- Install and configure Mininet.
- Set up Ryu controller.

(iv) **Weeks 6: Learning simulation tools**
- Receive packet-in message
- Handle flow-removal messages

- Set idle and hard timeout
- Retrieve the current status of flow table
- Determine the size of the flow table

(v) **Weeks 7-10: Writing Algorithm**

- Build data tables for the algorithm.
- Find thresholds and heuristics.
- Write the algorithm for timeout assignment and proactive eviction.
- Find datasets.

(vi) **Weeks 11: Algorithm Integration and Testing**

- Integrate datasets.
- Finalize algorithm.

(vii) **Weeks 12-14: Simulation, Test and Writing Report**

- Conduct simulation.
- Compare with other algorithms.
- Write report.

Project Plan on Github

2.2.2. Success Criteria

Algorithm Implementation: It was quite difficult to successfully implement our algorithm using the Mininet environment and the RYU controller. One of the most important success criteria was to successfully debug and achieve a stable, working implementation, since network behaviors are inherently unpredictable and complex.

Dataset Integration: A crucial component of our project was the integration of the UNIV1 dataset. We needed to complete this work in order to benchmark our algorithm and assess how well it performed in comparison to other solutions. Successfully incorporating this dataset and ensuring our algorithm could process and analyze it effectively was a significant measure of success.

Efficiency and Performance Improvement: A primary success criterion for our project was to develop an algorithm that outperforms existing strategies in terms of efficiency. Achieving a demonstrably better performance compared to competitor strategies was a fundamental objective of our project.

2.2.3. Risk Analysis

Technical Challenges: Our algorithm has been tested only within a simulated environment. Therefore, deployment in real-world may encounter unexpected technical issues.

Resource Limitations: Our algorithm uses controller memory to keep flow statistics, posing a risk of excessive memory usage. Moreover, the periodic proactive eviction call could strain the link between the switch and the controller, potentially degrading network performance.

Adaptability to Network Variability: We have designed the algorithm to be adaptive. Nonetheless, it is essential to test and investigate its compatibility across various network scenarios to ensure its robustness and effectiveness.

2.2.4. Team Work

We adapted to group work quite well. While initially scanning the articles, we divided some of the articles into two and explained them to each other through voice recordings. We also created joint notes about the articles through Notion, a note-taking application. We also used online shared resources for our algorithm. Notion[1] is our general collaborative note-taking app. We created a project on Github. We shared the issues from there. We do peer coding in places where we get stuck.

[1]`https://seemly-politician-629.notion.site/Satellite-Networkers-3b33217c60424e5e8e77b11aa49a`
`pvs=4`

3. RELATED WORK

In recent years, considerable research has focused on optimizing flow table management in SDNs due to its limited capacity. These methods focused mainly on approaches such as dynamic timeout mechanisms based on flow characteristics and proactive eviction mechanisms.

[1] introduces a dynamic idle timeout mechanism combined with proactive eviction, utilizing three modules: caching, timeout assignment, and eviction. The cache module records flow statistics, which then inform the assignment of idle timeouts to specific flows. Periodically, the eviction module removes long-interval flows using dynamic threshold values. However, this method might remove frequently used long-interval flows and does not consider table occupancy in its timeout decisions, potentially leading to rapid high table occupancy.

Reference [2] adopts dynamic hard timeout allocation along with LRU-based proactive eviction, targeting flows with $t_{idle} = t_{max}$. However, this approach may yield suboptimal results, especially when many flows have idle times less than t_{max}. Additionally, the use of a hard timeout can lead to flow removals just before new packets arrive, increasing table misses.

Studies [3], [4], and [6] focus solely on proactive eviction. [3] implements an LRU algorithm that evicts the oldest unmatched flow entry when the flow table's capacity is exceeded. [4] and [6] classify flow rules as active or inactive using machine learning models to optimize the flow table. Initially, necessary parameters are established through offline model training, followed by online flow table eviction. If no flow has an inactive probability above a certain threshold, the method defaults to LRU-based eviction.

IHTA [5] combines dynamic timeout allocation of both idle and hard timeouts with fixed values. It also evicts flow rules with the maximum hard timeout value if their packet count is below a certain threshold or if TCAM capacity exceeds another threshold. However, it lacks a removal policy for flow rules not meeting the maximum hard timeout criterion.

In [7], a Markov model is used to identify entries with the smallest matching probability. Then these entries are proactively evicted.
Research in [8] focuses on proactive eviction based on traffic characteristics. Their algorithm identifies and removes flow entries based on their likelihood of future matches. However, this approach incurs higher control-data plane bandwidth costs due to data retrieval from switches. Optimizing the sampling period is a significant challenge, leading us to believe that this algorithm may not be effective in isolation. [10]'s method is developed for software-defined satellite networks and utilizes a 2-step algorithm. First, it assigns idle timeouts dynamically based on the number of $packet_{in}$ messages of the flow while considering the table occupancy. The exponential increase ensures periodically coming flow rules are not removed from the flow table. However, [10] does not have a max timeout, which can cause a flow to be in the flow table for a long time. [10] also missing proactive eviction which might cause the flow table to be highly occupied at an earlier stage.

4. METHODOLOGY

The algorithm, as illustrated in Algorithm 1 and Algorithm 2, consists of Idle Timeout Allocation and Proactive Eviction. As packet-in messages are received, the idle timeout allocation algorithm is activated if the related entry is not already on the flow table. The allocation of idle timeouts utilizes the data_table. At intervals of T (period), if the table occupancy is above the high_threshold, the proactive eviction starts pulling flow table statistics and calculates heuristics accordingly. It then proceeds to delete entries with the smallest heuristic values until the occupancy drops to the lower_threshold.

Table 4.1. Description of Notatons

Notation	Description
t_{init}	initial idle timeout value for flows
t_{max}	max idle timeout value for flows
$t_{\text{lastRemoved}_k}$	Time when flow k is removed from the flow table, either due to eviction or timeout.
t_{packetIn_k},	Time when flow k enters the flow table.
$t_{\text{lastDuration}_k}$	Duration that flow $k$was in flow table.
n_{packetIn_k}	Number of times flow k has entered flow table.
$LowerBound$	Lower table occupancy ratio.
$UpperBound$	Upper table occupanct ratio.
$TCAM_{limit}$	Flow table capacity for the switch.
$TableOcc$	Current table occupancy ratio.

4.1. Idle Timeout Allocation

The Idle Timeout algorithm activates upon receiving a packet in message. It utilizes four methods to set idle timeouts as shown in Algorithm 1:

Initially, when a flow first enters the flow table, it is assigned a timeout of t_{init},

Algorithm 1 Dynamic Timeout Allocation

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

Algorithm 2 Flow Metrics Collector

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

n_{packetIn_k} is initialized by 1 and its arrival time, denoted as t_{packetIn_k} , is logged. This process is detailed in lines 2 to 5 of the Dynamic Timeout Allocation algorithm.

If the flow table is not heavily occupied, the idle timeout increases rapidly, doubling with each new packet, as expressed by the formula $2^{\text{packet\_count}}$. The 'packet count' is tracked in the data table, which helps assign longer timeouts for recurring flows, up to a maximum limit, t_{max}. This cap is designed to prevent the flow table from getting too full.(line 8-9)

Between the first and second occupancy thresholds(line 10-17), the idle timeout is set according to the flow's interarrival time. As shown in line 11, t_{max} is dynamically adjusted based on table density. If a new flow arrives before the current idle timeout expires, the timeout extends to cover this flow as outlined in line 12,13.

Should a flow reappear after a significant delay, the timeout resets to the previous value from the data table, calculated by the difference $T_{\text{packetIn}} - T_{\text{lastRemoved}}$ as line 15 details.

The Packet Count serves as a metric to track the number of times packets re-enter the flow table after their initial departure. The term T_{packetIn} denotes the time when a packet arrives at the flow table. $T_{\text{lastRemoved}}$ indicates the most recent moment a flow was removed from the flow table.

The $T_{\text{lastDuration}}$ represents the total duration a flow spent in the table during its last presence. This duration may not necessarily match the Idle Timeout, as the flow could have been evicted earlier. Each parameter plays a crucial role in understanding and managing the life cycle of flows within the network infrastructure. When the table occupancy surpasses the higher threshold, a minimal idle timeout of 1 is imposed.

Flow Metrics Collector algorithm runs whenever a flow removed. Flow data is collected for idle timeout allocation.

4.2. Proactive Eviction

When the flow table reaches capacity, network performance can deteriorate significantly. Incoming flows face the risk of being dropped or may necessitate the eviction of existing flows. The primary goal is to ensure that the flow table never becomes full. To this end, we periodically run a proactive eviction algorithm at intervals denoted by T. Within this algorithm, when table occupancy exceeds a predefined high threshold, the controller issues a stats request to the switch using OpenFlow API. Upon receiving the statistics, the algorithm calculates a heuristic named ActiveFlowMetric for each flow, based on the hit count from flow statistics and the packet_in time from the data table.

ActiveFlowMetric is calculated as follows:

$$\text{ActiveFlowMetric} = \frac{\text{total\_hits}}{t_{\text{current}} - t_{\text{packet\_in}}}$$

total_hits describes the times when a flow comes and matched with flow table. t_{current} is for current time and $t_{\text{packet\_in}}$ is the same parameter as in idle timeout allocation. The algorithm continues to remove flows from the proactive eviction table, which contains flows paired with their calculated heuristics until the occupancy drops below the low threshold. For each selected flow rule, a delete command is sent to the switch, effectively removing the flow.(lines 6, 7)

Both the idle timeout and proactive eviction algorithms offer tunable parame-

Algorithm 3 Proactive Eviction Algorithm Running Every T Seconds

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

ters, allowing adaptation to the specific characteristics of the network. This flexibility ensures that network performance is optimized while preventing table overflow and the associated negative consequences.

5. DESIGN

5.1. Information Flow

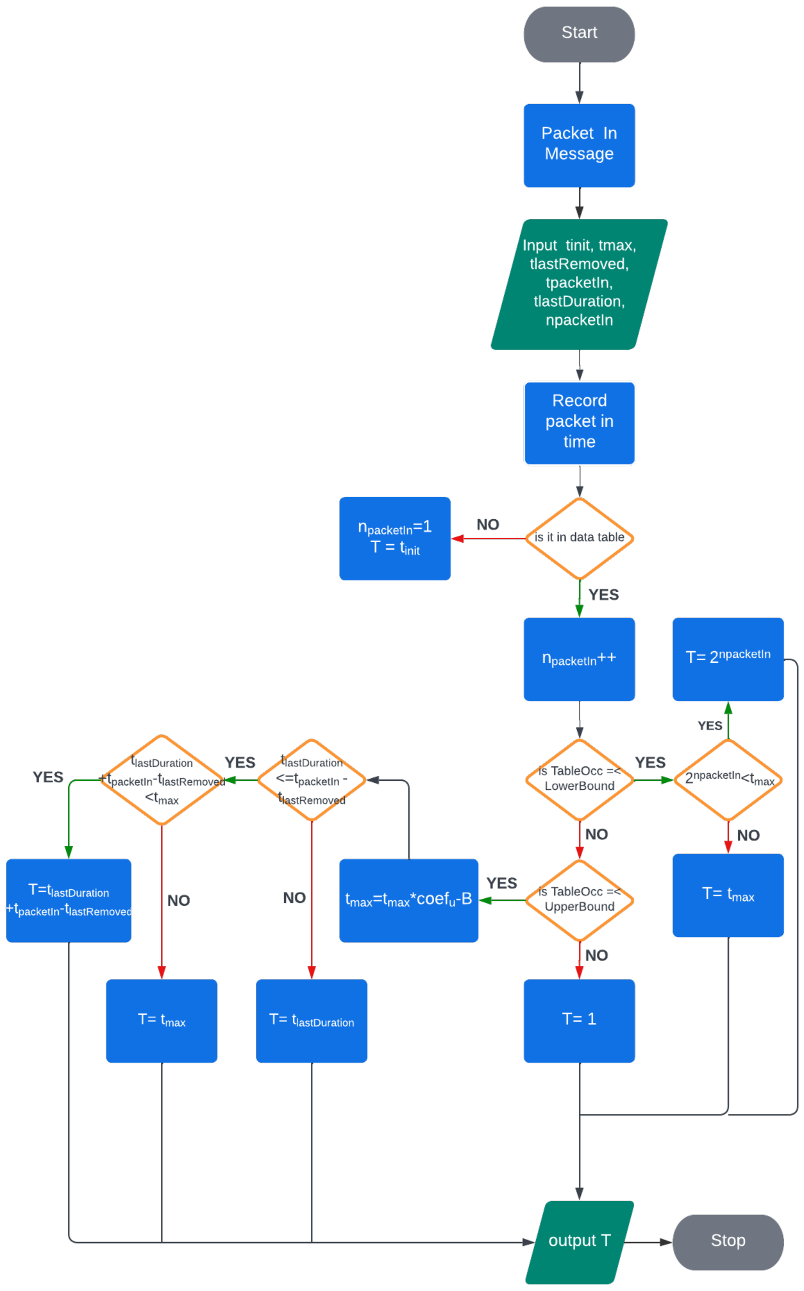
Start
Packet In Message
Input tinit, tmax, tlastRemoved, tpacketIn, tlastDuration, npacketIn
Record packet in time
is it in data table
NO
$n_{packetIn}$=1 T = t_{init}
YES
$n_{packetIn}$++
is TableOcc =< LowerBound
YES
$2^{npacketIn}$<t_{max}
YES
T= $2^{npacketIn}$
NO
T= t_{max}
NO
is TableOcc =< UpperBound
YES
t_{max}=t_{max}*$coef_u$-B
NO
T= 1
$t_{lastDuration}$ <=$t_{packetIn}$ - $t_{lastRemoved}$
YES
NO
T= $t_{lastDuration}$
$t_{lastDuration}$ +$t_{packetIn}$-$t_{lastRemoved}$ <t_{max}
YES
T=$t_{lastDuration}$ +$t_{packetIn}$-$t_{lastRemoved}$
NO
T= t_{max}
output T
Stop

Figure 5.1. Idle Timeout Allocation

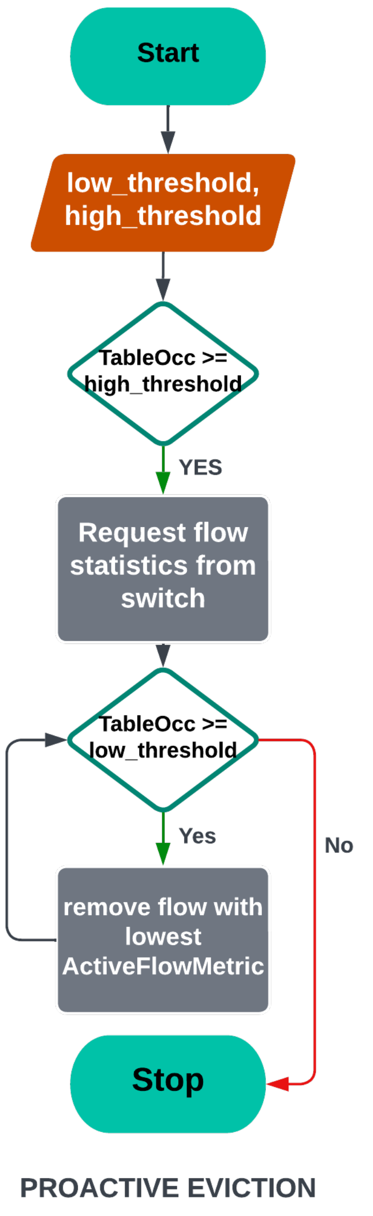
Start
low_threshold, high_threshold
TableOcc >= high_threshold
YES
Request flow statistics from switch
TableOcc >= low_threshold
Yes
No
remove flow with lowest ActiveFlowMetric
Stop
PROACTIVE EVICTION

Figure 5.2. Proactive Eviction

5.2. System Design

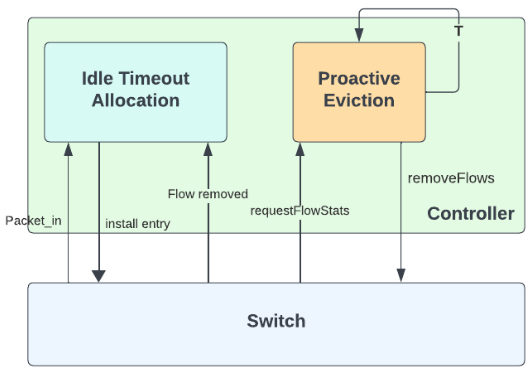

Figure 5.3. Algorithm Structure

6. REQUIREMENTS SPECIFICATION

6.1. Introduction

6.1.1. Project Scope

The project aims to develop an efficient flow table management system in SDN that combines proactive eviction with dynamic idle timeout mechanisms.

6.2. Overall Description

6.2.1. Product Perspective

This project enhances traditional SDN controllers by adding an algorithm for flow management.

6.2.2. Project Parts

6.2.2.1. Proactive Eviction. Removes less active flows from the table according to an algorithm.

6.2.2.2. Dynamic Idle Timeout Allocation. Assign idle timeouts based on flow characteristics and network conditions.

6.3. System Requirements

6.3.1. Proactive Eviction

Functional Requirements:

6.3.1.1. Requirement. System shall identify the flows for eviction.

6.3.1.2. Requirement. System shall have configurable thresholds for eviction mechanism.

6.3.1.3. Requirement. System shall detect the high table occupancy and evict the selected flows.

6.3.2. Dynamic Idle Timeout Allocation

Functional Requirements:

6.3.2.1. Requirement. The system shall set timeouts based on flow statistics dynamically for each flow.

6.3.2.2. Requirement. The system shall must not exceed the predefined maximum timeout value.

6.4. Non-Functional Requirements

6.4.1. Performance

The project shall be capable of processing flow table updates in real-time without significant latency.

6.4.2. Scalability

The project should be scalable to accommodate growing network demands.

6.4.3. Maintainability

The codebase should be well-documented and modular for easy maintenance.

7. IMPLEMENTATION AND TESTING

7.1. Implementation

We preferred the Python programming language for the implementation of the algorithm. We integrate the Ryu as the controller which is an open-source controller used in the field of SDN (Software Defined Networks). Additionally, we use the version 1.3 of the OpenFlow protocol and the software called Mininet to perform simulations of the algorithm. We are using OpenvSwitch which enables flow table management and packet transmission. We divide our project into 2 parts:

- Topology creation part where we are reading our dataset.
- Proposed algorithm implementation part that includes Idle timeout allocation and proactive eviction.

The GitHub link for our codebase can be found at this link.

7.1.1. Topology Creation

We utilized Python to construct our network topology. We have created 2 hosts. For simplicity, we created a single switch connected to the Ryu controller. We restricted our use to the first 700 seconds of the UNIV1 dataset, which comprises 2,023,744 packet counts. Network traffic from the dataset was replayed on both hosts using the 'tcpreplay' tool.

7.1.2. Algorithm Implementation

We implemented our algorithm into different functions for better coding practices. In the 'set idle timeout' function, we implemented our idle timeout allocation, which is called for each flow addition.

We use a separate thread for proactive eviction. In this thread, every two seconds, we check the table occupancy and perform the proactive eviction algorithm if the conditions are met.

We utilize different dictionaries to store the flow statistics necessary for different parts of our algorithm. The data table used in idle timeout allocation includes:

- last packet-in time,
- last removal time,
- packet count,
- last duration,
- the idle timeout

In the eviction data table used in the proactive eviction algorithm, we hold:

- packet-in time,
- hit count,
- active flow metric value

The hit count corresponds to the number of times a flow in the table is matched. The active flow metric is a heuristic value used to assign importance to each flow, as explained in the methodology part.

We also have a separate thread for calculating the average values of metrics such as CPU usage, memory usage, and table occupancy. Every five seconds, we request table statistics from the switch to monitor the miss rate.

All flows are maintained in the flow table set. Additionally, we identify each flow using a unique key formatted as $source - destination - inport$. We are using locks for each of the shared variables used in our algorithm such as table occupancy, data table, and eviction data table.

7.2. Testing

To evaluate our algorithm, we compared it with algorithms in the literature and a fixed timeout mechanism, using a part of the publicly available UNIV1 dataset. Our experiments were conducted with various flow table sizes, and we focused on several key metrics for comparison:

- Miss rate[2] ,
- Flow table occupancy rate,
- CPU usage,
- Memory usage,
- Packet in count
- Number of rejected flows[3]

We evaluated our algorithm's performance by looking at these metrics for varying flow table sizes, 200 and 300. We particularly compared our approach to the AFTM [1] and the first section of the TSMM [8][4] algorithm except for the handover handling part. Since AFTM [1] has been shown to be more effective than its competitors, such as Smart Time [16] and Intelligent Timeout Master [15], surpassing AFTM [1] would also indicate supremacy over them. Furthermore, we have integrated elements of TSMM [8] into the $2^{npacketin}$ strategy where we have improved our idle timeout allocation approach by adding a maximum idle timeout and using this method inside a threshold value for table occupancy. Additionally, we compare our algorithm with a fixed timeout strategy with an idle timeout of 3 seconds and a hard timeout of 5 seconds.

7.3. Deployment

In order to use the system, Python3, Ryu controller and Mininet must be installed. Before installation, you must configure this software correctly in accordance

[2]The percentage of flow requests that could not be matched, leading to packet-in messages.
[3]The number of flow requests denied due to space inefficiency in flow table.
[4]We will refer to it as the TSMM algorithm for simplicity.

with the operating system you prefer. For detailed instructions on Ryu Controller installation and usage, refer to the Ryu documentation. To download the Mininet VM, visit the Mininet download page.

7.3.1. Installing Ryu Controller

7.3.1.1. Using pip.

- Open a terminal or command prompt.
- Run: pip install ryu

7.3.1.2. Manual Installation.

- Clone the Ryu repository from GitHub: Ryu GitHub Repository
- Navigate to the Ryu directory.
- Run: pip install .

7.3.2. Installing Mininet

- Open a terminal.
- Run the Mininet installation script:

```
git clone git://github.com/faucetsdn/ryu
cd ryu
pip install .
```

The commands we are currently using for our study:

- `ryu-manager timeoutAllocation.py`
- `python3 our_topology.py`

8. RESULTS

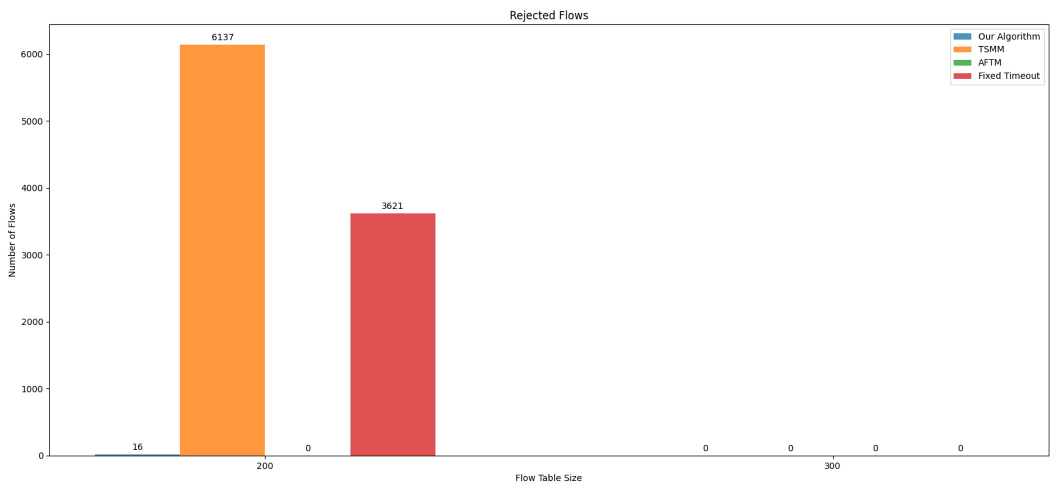

Figure 8.1. Rejected Flows

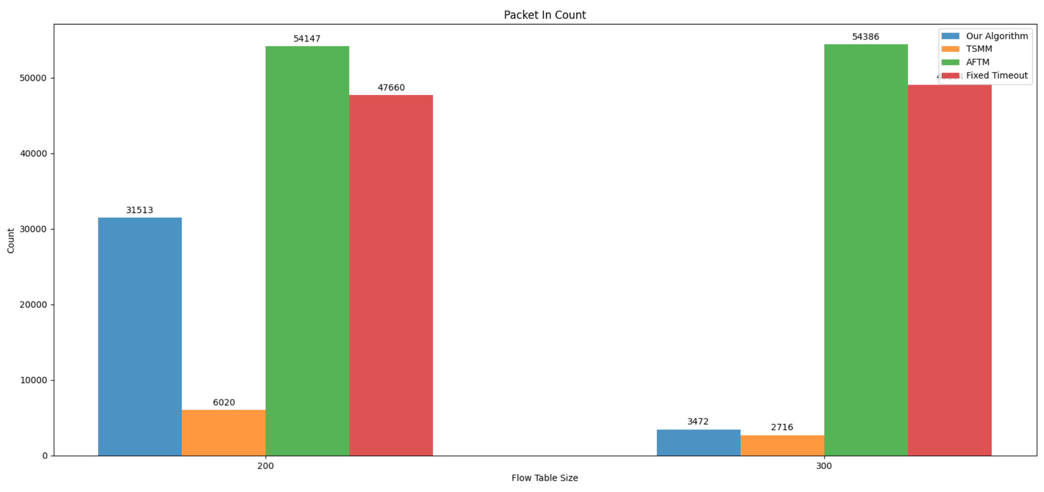

Figure 8.2. Total Packet-in Count

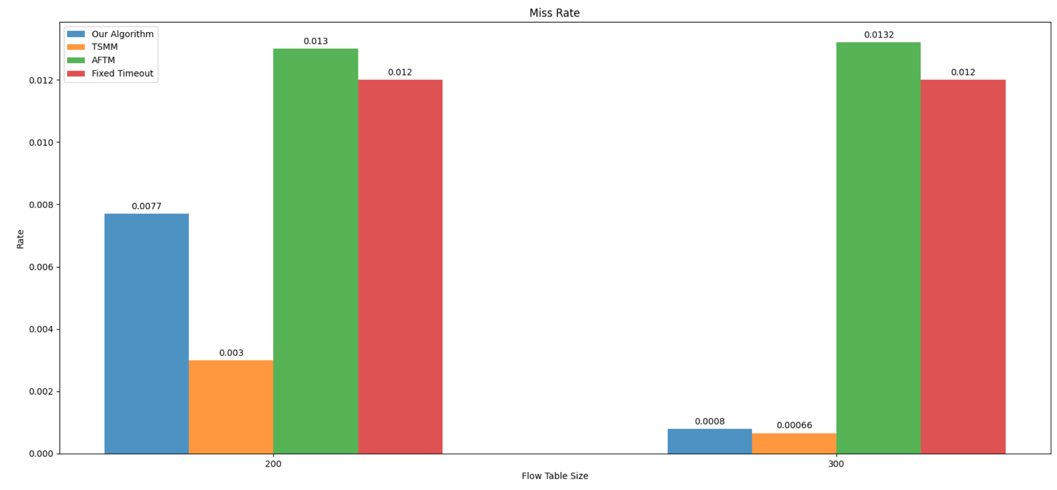

Figure 8.3. Miss Rate

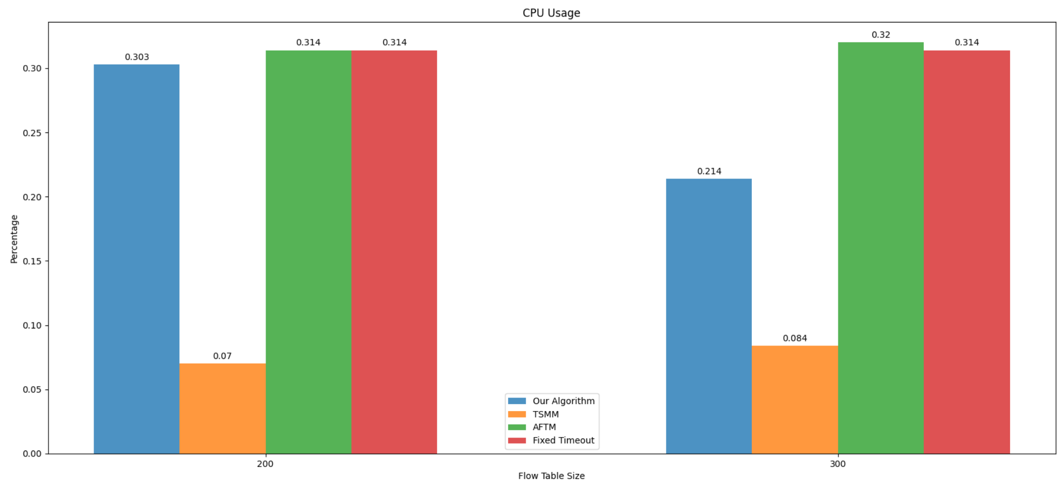

Figure 8.4. Average CPU Usage

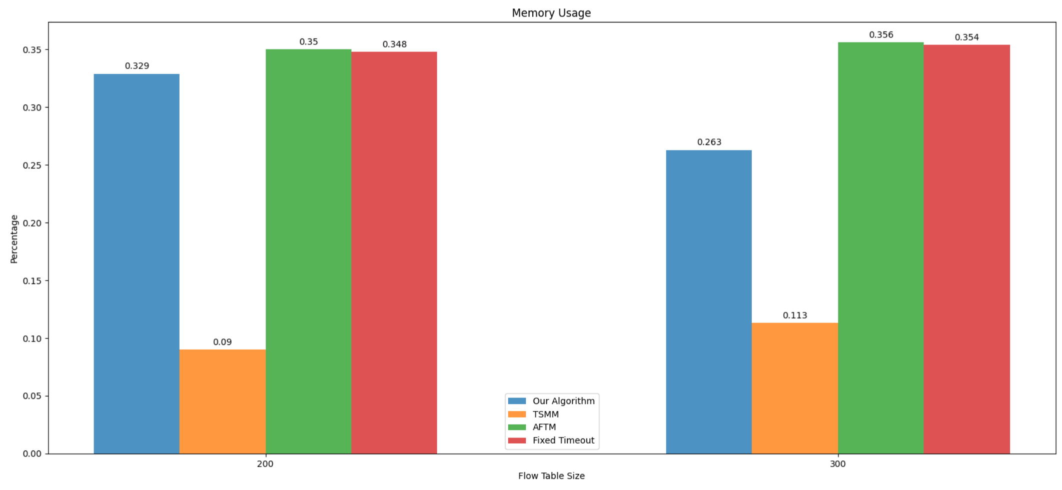

Figure 8.5. Average Memory Usage

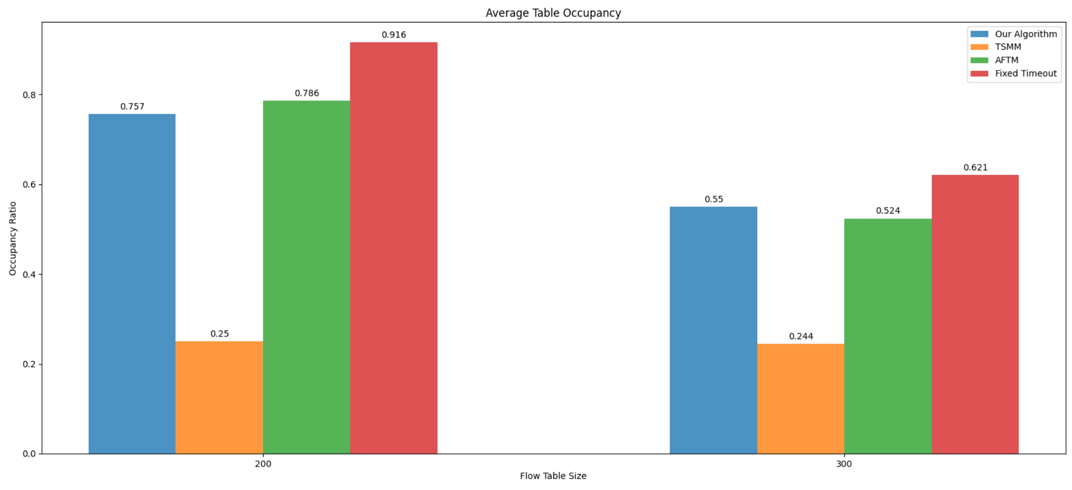

Figure 8.6. Average Table Occupancy

	Our Solution	TSMM	AFTM	Fixed Timeout
Total Packet Count	6020	54147	54147	47660
Rejected Flows	16	6137	0	3621
Avg. Table Occupancy	0.757	0.25	0.786	0.916
Avg. Memory Usage	0.329	0.09	0.350	0.348
Avg. CPU Usage	0.303	0.07	0.314	0.314
Miss rate	0.0077	0.003	0.013	0.012

Table 8.1. Comparison table for table size 200

	Our Solution	TSMM	AFTM	Fixed Timeout
Total Packet Count	3472	2716	54386	49074
Rejected Flows	0	0	0	0
Avg. Table Occupancy	0.550	0.244	0.524	0.621
Avg. Memory Usage	0.263	0.113	0.356	0.354
Avg. CPU Usage	0.214	0.084	0.320	0.314
Miss rate	0.0008	0.00066	0.0132	0.0120

Table 8.2. Comparison table for table size 300

The fixed timeout mechanism leads to a more occupied flow table, showing 0.916 and 0.621 occupancy for table sizes 200 and 300, respectively. This might seem a better utilization, however, without checks or control, it risks table overflow. Our solution has similar table occupancy, slightly better than AFTM [1]. Our results are 0.757 and 0.550 for sizes 200 and 300, while AFTM [1] shows 0.786 and 0.524 for these sizes.

In terms of average memory and CPU usage, TSMM [8] shows the best results, followed by our algorithm. AFTM [1] and fixed timeout have similar but worse outcomes. For the total packet count, TSMM [8] is the lowest, with our method next in line. The packet-in count in AFTM is worse than the packet count in the fixed timeout mechanism.

While TSMM [8] does well in packet counts, it has many rejected flows, possibly due to not using a maximum idle timeout. This means some flows stay too long on the table, leaving insufficient space for new ones.

For the miss rate, TSMM [8] scores 0.003 and 0.00066 for sizes 200 and 300, followed by our algorithm with 0.0077 and 0.0008. AFTM [1] has the worst miss rate at 0.013 for both sizes.

In conclusion, our solution could be considered a more suitable choice overall. While other solutions excel in certain areas, they perform poorly in others. For example, TSMM [8] has a good miss rate but many rejected flows. AFTM [1] has fewer rejected flows but a worse miss rate compared to others. Fixed timeout utilizes the table space more but risks an overflow. Our solution, although open to improvements, shows good results in these aspects.

9. CONCLUSION and FUTURE WORK

In Software-Defined Networking (SDN), efficiently managing the flow table is very important because of its limited capacity due to TCAM. In our project, we have developed a new solution that uses proactive eviction along with dynamic idle timeout allocation to better manage flow table capacity.

In our experiments, where we compared our algorithm with others, we found that our results were generally better than those achieved by the AFTM [1] algorithm concerning aspects mentioned in the testing section. This also means we outperform algorithms like Smart Time [16] and Intelligent Timeout Master [15] since AFTM [1] is already better than them. [5] In comparison with the TSMM [8] algorithm, we noticed that TSMM [8] performs slightly better than ours in miss rate, packet-in count, and average memory and CPU usage. However, we receive a smaller number of rejected flows. This shows us areas where we can improve our algorithm. Compared to the fixed timeout mechanism, we are achieving better results in almost all aspects mentioned in the results section.

It is important to note that these results might vary depending on the dataset used in our experiments. Thus, we plan to use different datasets and test various flow table sizes to fine-tune our algorithm, especially the thresholds for table occupancy. Since we use different sub-algorithms based on these thresholds, changes here could significantly affect how well our solution works.

Looking ahead, we are interested in exploring the use of artificial intelligence to decide which flows to evict. We were planning to adapt our solution for the satellite network, however, we postponed it to our future research. Thus, another future goal is to adapt our solution for satellite networks, taking into account the handover process.

Youtube video can be found on this link.

[5]This is evaluated in the AFTM [1] study.

REFERENCES

1. Y. Shen, 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; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, 2020, pp. 917-922, doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00122.

2. A. Panda, S. S. Samal, A. K. Turuk, A. Panda and V. C. Venkatesh, ”Dynamic Hard Timeout based Flow Table Management in Openflow enabled SDN,” 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), Vellore, India, 2019, pp. 1-6, doi: 10.1109/ViTECoN.2019.8899359.

3. D. Wu, L. Qiao and Q. Chen, ”Research and Implementation of LRU-Based Flow Table Management for Onboard Switch,” 2020 Prognostics and Health Management Conference (PHM-Besançon), Besancon, France, 2020, pp. 300-303, doi: 10.1109/PHM-Besancon49106.2020.00058.

4. H. Yang, G. F. Riley and D. M. Blough, ”STEREOS: Smart Table EntRy Eviction for OpenFlow Switches,” in IEEE Journal on Selected Areas in Communications, vol. 38, no. 2, pp. 377-388, Feb. 2020, doi: 10.1109/JSAC.2019.2959184.

5. B. Isyaku, M. B. Kamat, K. B. Abu Bakar, M. S. Mohd Zahid and F. A. Ghaleb, ”IHTA: Dynamic Idle-Hard Timeout Allocation Algorithm based OpenFlow Switch,” 2020 IEEE 10th Symposium on Computer Applications Industrial Electronics (ISCAIE), Malaysia, 2020, pp. 170-175, doi: 10.1109/ISCAIE47305.2020.9108803.

6. H. Yang and G. F. Riley, ”Machine Learning Based Flow Entry Eviction for Open-

Flow Switches," 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 2018, pp. 1-8, doi: 10.1109/ICCCN.2018.8487362.

7. Jan, S., Guo, Q., Jia, M., Khan, M.K. (2019). Intelligent Dynamic Timeout for Efficient Flow Table Management in Software Defined Satellite Network. In: Jia, M., Guo, Q., Meng, W. (eds) Wireless and Satellite Systems. WiSATS 2019. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 280. Springer, Cham.

8. Li, Taixin Huachun, Zhou Luo, Hongbin Quan, Wei Xu, Qi Li, Guanwen Li, Guanglei. (2017). Timeout Strategy-based Mobility Management for Software Defined Satellite Networks. 319-324. 10.1109/INFCOMW.2017.8116396.

9. T. Li, H. Zhou, H. Luo, I. You and Q. Xu, "SAT-FLOW: Multi-Strategy Flow Table Management for Software Defined Satellite Networks," in IEEE Access, vol. 5, pp. 14952-14965, 2017, doi: 10.1109/ACCESS.2017.2726114.

10. L. Zhang, 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), San Diego, CA, USA, 2015, pp. 1-6, doi: 10.1109/GLOCOM.2015.7417563.

11. Huang, G., Youn, H.Y. Proactive eviction of flow entry for SDN based on hidden Markov model. Front. Comput. Sci. 14, 144502 (2020).

12. X. Li and Y. Huang, "A Flow Table with Two-Stage Timeout Mechanism for SDN Switches," 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 2019, pp. 1804-1809, doi: 10.1109/HPCC/SmartCity/DSS.2019.00248.

13. J. Xu, L. Wang, C. Song and Z. Xu, "Proactive Mitigation to Table-Overflow in Software-Defined Networking," 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 2018, pp. 00719-00725, doi: 10.1109/ISCC.2018.8538670.

14. Abbasi, Mahdi Maleki, Shima Jeon, Gwanggil Khosravi, Mohamadreza Abdoli, Hatam. (2022). An intelligent method for reducing the overhead of analysing big data flows in Openflow switch. IET Communications. 1. 1-12. 10.1049/cmu2.12328.

15. H. Zhu, H. Fan, X. Luo, and Y. Jin, "Intelligent timeout master: Dynamic timeout for sdn-based data centers," in 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 734–737, IEEE, 2015.

16. A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in openflow networks," in Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, pp. 177–188, ACM, 2014.