АҚПАРАТТЫҚ-КОММУНИКАЦИЯЛЫҚ ТЕХНОЛОГИЯЛАР / ИНФОРМАЦИОННО-КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ / INFORMATION AND COMMUNICATION TECHNOLOGIES

DOI 10.54596/2958-0048-2025-4-166-181
UDK 311.2
IRSTI 49.03.03

OPTIMISING SDN THROUGHPUT VIA FLOW-TABLE MANAGEMENT: A COMPARATIVE STUDY AND FUTURE RESEARCH OUTLOOK

Azizol Abdullah[1*], Mamun Md Arafat Al[1], Ahmad Alauddin Ariffin[1], Lili Nurliyana Abdullah[1], Mohd Noor Derahman[1]
[1]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia*
[*]Corresponding author: azizol@upm.edu.my

Abstract

Software-Defined Networking (SDN) has transformed how networks are managed by separating control from data plane, making them more flexible and programmable However, throughput performance remains constrained by controller latency and the limited size of ternary content-addressable memory (TCAM) in switches. To tackle this, Flow-Table Reduction Schemes (FTRS) offer a simple, software-driven fix. In this paper, we explore how SDN throughput optimization has evolved, compare popular controllers, and show where FTRS fits in. We share real-world results from implementing FTRS on the Ryu controller, discuss why these matters for cost and sustainability, and outline future directions like using machine learning and multi-controller setups for smarter, faster networks

Keywords: Software-Defined Networking, Flow-Table Reduction, TCAM, Ryu Controller, Future Research

АҒЫНДАР КЕСТЕЛЕРІН БАСҚАРУ АРҚЫЛЫ SDN ӨТКІЗУ ҚАБІЛЕТІН ОҢТАЙЛАНДЫРУ: САЛЫСТЫРМАЛЫ ЗЕРТТЕУ ЖӘНЕ БОЛАШАҚ ЗЕРТТЕУЛЕР ПЕРСПЕКТИВАЛАРЫ

Azizol Abdullah[1*], Mamun Md Arafat Al[1], Ahmad Alauddin Ariffin[1], Lili Nurliyana Abdullah[1], Mohd Noor Derahman[1]
[1]Компьютерлік ғылымдар және ақпараттық технологиялар факультеті, Путра Малайзия университеті, Малайзия*
[*]Хат-хабар үшін автор: azizol@upm.edu.my

Аңдатпа

Бағдарламалық-анықталатын желілер (SDN) басқаруды деректер жазықтығынан бөлу арқылы желіні басқару тәсілін өзгертті, бұл оларды икемді және бағдарламаланатын етті. Дегенмен, өткізу қабілеттілігі әлі де контроллердің кідірісімен және коммутаторлардағы мазмұн адрестеуімен (TCAM) шектелген үштік жад өлшемімен шектеледі. Ағындар кестесін қысқарту схемалары (FTRS) осы мәселесін шешу үшін қарапайым бағдарламалық шешім ұсынылады. Бұл мақалада SDN өткізу қабілеттілігін оңтайландыру қалай дамығандығы, танымал контроллерлер салыстырылып, FTRS қай жерде қолданылатындығы көрсетілген. Ryu контроллерінде FTRS енгізудің нақты нәтижелері көрсетілген, оның құны мен тұрақтылығы үшін не маңызды екендігі зерттелген және ақылды, жылдам желілер үшін машиналық оқыту мен көп контроллерлік параметрлерді пайдалану сияқты болашақ бағыттар сипатталған.

Кілт сөздер: бағдарламалық-анықталатын желілер, ағындар кестесін қысқарту, TCAM, Ryu контроллері, болашақ зерттеулер.

ОПТИМИЗАЦИЯ ПРОПУСКНОЙ СПОСОБНОСТИ SDN ПОСРЕДСТВОМ УПРАВЛЕНИЯ ТАБЛИЦАМИ ПОТОКОВ: СРАВНИТЕЛЬНОЕ ИССЛЕДОВАНИЕ И ПЕРСПЕКТИВЫ БУДУЩИХ ИССЛЕДОВАНИЙ

Azizol Abdullah[1*], Mamun Md Arafat Al[1], Ahmad Alauddin Ariffin[1], Lili Nurliyana Abdullah[1], Mohd Noor Derahman[1]

[1]Факультет компьютерных наук и информационных технологий, Университет Путра Малайзия, Малайзия*

[*]Автор для корреспонденции: azizol@upm.edu.my

Аннотация

Программно-определяемые сети (SDN) изменили подход к управлению сетями, отделив контроль от плоскости передачи данных, что сделало их более гибкими и программируемыми. Однако пропускная способность по-прежнему ограничивается задержкой контроллера и ограниченным размером троичной памяти с адресацией по содержанию (TCAM) в коммутаторах. Для решения этой проблемы схемы сокращения таблиц потоков (FTRS) предлагается простое программное решение. В этой статье исследовано, как развивалась оптимизация пропускной способности SDN, сравниваются популярные контроллеры и показаны, где FTRS может быть применен. Показаны реальные результаты внедрения FTRS на контроллере Ryu, изучено, почему это важно для стоимости и устойчивости, и описаны будущие направления, такие как использование машинного обучения и настроек с несколькими контроллерами для более умных и быстрых сетей.

Ключевые слова: Программно-определяемые сети, сокращение таблиц потоков, TCAM, контроллер Ryu, будущие исследования

1. Introduction

Software-Defined Networking (SDN) changes the game by splitting control from data plane, giving networks the flexibility to adapt quickly and be managed through software instead of rigid hardware rules. This makes it easier to apply dynamic policies and optimize traffic flow (McKeown et al., 2008; Kreutz et al., 2015) as shown in Figure 1. However, despite these advantages, performance degradation remains a persistent challenge. Particularly when reactive controllers face high volumes of short-lived or micro-burst flows (Yang et al., 2022; Gao et al., 2022). These micro-bursts create delays and overload the control plane, slowing down the entire network (Tootoonchian et al., 2012; Ghobadi et al., 2020).

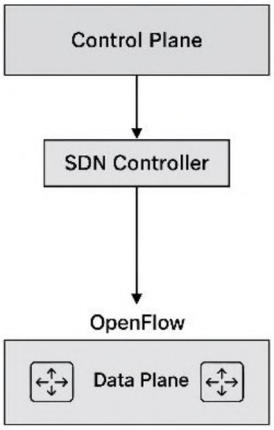

Figure 1. General architecture of Software-Defined Networking (SDN) showing decoupled control and data planes (adapted from McKeown et al., 2008).

Flow-table congestion, a recurring issue in high-speed SDN switches, stems mainly from limited TCAM (Ternary Content Addressable Memory) capacity (Zahavi & Zilberman, 2021). When TCAM entries become saturated, switches experience delays in flow matching, resulting in packet drops and control overhead amplification. This degradation directly affects Quality of Service (QoS) metrics, including latency, jitter, and throughput (Curtis et al., 2011; Alizadeh et al., 2014). To mitigate these limitations, researchers have proposed Flow-Table Reduction Schemes (FTRS), which compress similar flow entries into aggregated wildcard rules, thereby reducing control-plane overhead and TCAM exhaustion (Leng et al., 2017; Chen et al., 2021). In practical deployments, FTRS is particularly beneficial in high-density and latency-sensitive environments such as data center fabrics, Internet of Things (IoT) gateways, smart campus networks, and 5G edge computing infrastructures. In these scenarios, networks often experience high volumes of short-lived microflows, which rapidly exhaust TCAM resources and overload reactive controllers. By aggregating redundant flow rules, FTRS is expected to reduce TCAM utilization by approximately 30–50% and improve controller processing efficiency, leading to measurable gains in throughput and latency under real-world traffic workloads.

Yet, most FTRS implementations have been validated only through simulation, with few practical deployments on lightweight, Python-based controllers such as Ryu (Shalimov et al., 2013; Fernández et al., 2018). This study bridges the current research gap by implementing an optimized Flow-Table Reduction Scheme (FTRS) on the Ryu controller and comparing its throughput performance against other SDN platforms. Specifically, the research examines the evolution of SDN controller architectures, evaluates the throughput performance of Ryu under flow-table optimization, and analyzes the comparative strengths and weaknesses of different controller platforms. Furthermore, it outlines future research pathways for enhancing intelligent flow management through the integration of machine learning techniques.

The remainder of this paper is structured as follows. Section 2 reviews the evolution of SDN controllers and related work on flow-table optimization. Section 3 outlines the methodology used for implementing and testing the FTRS in Ryu. Section 4 presents comparative experimental results and performance analysis. Section 5 discusses key implications and limitations. Section 6 proposes future research directions, and Section 7 concludes the paper.

2. Background and Related Work

This section reviews how SDN controllers have evolved from early centralized designs to modern distributed and lightweight platforms. We also look at the challenges of managing flow tables and how techniques like FTRS help reduce congestion and improve performance. (Gude et al., 2008; Berde et al., 2014; Fernández et al., 2018).

2.1 Evolution of SDN Controllers

The evolution of Software-Defined Networking (SDN) controllers reflects the broader trajectory of SDN itself, from academic prototypes to production-grade, scalable network control platforms. Controllers serve as the “brain” of SDN, managing the network state, computing flow rules, and maintaining communication with switches through the southbound interface (e.g., OpenFlow). Over the past decade, controller architectures have evolved significantly, each generation attempting to balance scalability, latency, fault tolerance, and programmability (Kreutz et al., 2015; Lara et al., 2014) as shown in Figure 2.

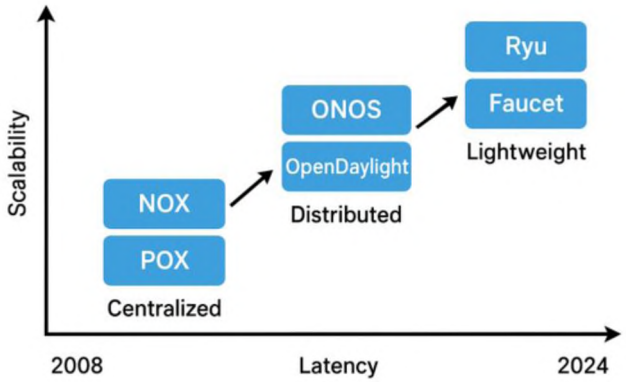

Figure 2. Evolution of SDN controllers from centralized (NOX, POX) to distributed (ONOS, OpenDaylight) and lightweight (Ryu, Faucet) architectures (adapted from Fernández et al., 2018; Berde et al., 2014)

First-generation controllers such as NOX and POX (McKeown et al., 2008; Gude et al., 2008) served as foundational proofs-of-concept, introducing programmable control over forwarding elements. However, they exhibited scalability challenges, particularly under heavy *PACKET_IN* loads, as all flow decisions were managed by a single centralized controller. This architecture, though innovative, struggled to maintain low latency in large-scale deployments. Distributed controllers such as ONOS (Berde et al., 2014) and OpenDaylight (Medved et al., 2014) were developed to address scalability and fault tolerance. They employ clustered control planes to distribute decision-making across nodes. While these platforms improve reliability, they introduce quorum-induced latency, where inter-node synchronization increases control delays (Ghobadi et al., 2020; Ganji et al., 2024).

Intent-based controllers like Cisco ACI and Juniper Contrail abstract network policies into high-level intents that automatically compile into flow rules (Monsanto et al., 2013; Kim & Feamster, 2013). This abstraction simplifies management but inadvertently amplifies TCAM consumption, as the system must pre-install multiple rules for generalized intent enforcement. Lightweight controllers, such as Ryu and Faucet, emerged as flexible, developer-friendly frameworks prioritizing ease of prototyping and rapid integration. Ryu stands out due to its Python-based modularity, allowing researchers to experiment with flow-table algorithms and machine learning integration with minimal setup complexity (Shalimov et al., 2013; Fernández et al., 2018).

Overall, the evolution from NOX to Ryu illustrates a clear trajectory toward greater modularity, openness, and deployability. The shift from monolithic to lightweight controller architectures signifies not just a technological refinement but a philosophical transition. Its prioritising programmability and adaptability over raw scalability. Within this context, implementing performance-enhancing schemes such as FTRS inside Ryu represents a logical step in the ongoing pursuit of efficient, reproducible, and energy-conscious SDN control frameworks.

2.2 Flow-Table Management Challenges

One of the most fundamental design trade-offs in Software-Defined Networking (SDN) lies in determining how and when flow rules are installed in switches that profoundly impacts latency, scalability, and throughput. SDN controllers typically adopt either reactive or proactive flow management strategies, each with unique strengths and limitations depending on network

topology, traffic patterns, and operational objectives (Kreutz et al., 2015; He et al., 2021). In reactive flow management, the controller installs flow entries dynamically in response to the first packet of each new flow. When a switch encounters an unknown flow, it issues a *PACKET_IN* message to the controller, which computes the forwarding path and returns a *FLOW_MOD* instruction (McKeown et al., 2008). This approach ensures that only active flows occupy TCAM space, conserving memory resources and enabling fine-grained policy enforcement. Reactive control is thus well suited for environments requiring context-aware routing, such as intrusion detection, anomaly tracking, or load balancing at the edge (Yeganeh et al., 2013; Kim & Feamster, 2013).

However, the major limitation of the reactive paradigm lies in control-plane latency. Each new flow induces a round-trip delay between the switch and controller, often on the order of tens of milliseconds, which becomes problematic under micro-burst or elephant-flow conditions (Kobayashi et al., 2014). As network scale increases, the volume of *PACKET_IN* requests can saturate controller queues, resulting in dropped packets, inflated RTTs, and degraded throughput (Fernandez et al., 2018; Hu et al., 2020). The latency cost of reactive control has therefore been a key driver of SDN performance bottlenecks in data centres and cloud environments. By contrast, proactive flow management pre-installs flow entries before traffic arrival, based on predicted communication patterns or static topology knowledge (Tootoonchian et al., 2012). This strategy eliminates controller roundtrips during packet forwarding, allowing near-immediate data-plane processing. In highly stable or predictable networks such as backbone ISPs, enterprise LANs, or IoT clusters where proactive provisioning can drastically reduce control overhead and ensure deterministic latency (Curtis et al., 2011). However, this gain comes at the cost of flow-table overutilisation. Since rules are installed pre-emptively, many entries may remain unused or become obsolete, consuming valuable TCAM resources and reducing space for dynamically generated flows (Zahavi & Zilberman, 2021).

The tension between these two paradigms is reactive flexibility versus proactive speed which has led to numerous attempts at hybrid flow management. Hybrid controllers seek to classify traffic into mice (short-lived) and elephant (long-lived) flows, applying reactive strategies to the former and proactive strategies to the latter (Benson et al., 2010; Alizadeh et al., 2014). Techniques such as DevoFlow (Curtis et al., 2011) and Mahout (Curtis et al., 2012) introduced hierarchical aggregation, where switches autonomously handle frequent flows while the controller manages strategic updates. These solutions reduce controller load but often require hardware or firmware extensions, limiting their general adoption. Moreover, accurate flow classification remains a persistent challenge specially under bursty, asymmetric, or encrypted traffic, where flow duration and size are difficult to predict (Liu et al., 2020).

Leng, Liu, and Li (2017) introduced the Flow-Table Reduction Scheme (FTRS) as an aggregation-based solution, consolidating multiple microflows into broader wildcard entries. In the context of this study, Flow-Table Reduction Schemes (FTRS) can be viewed as a complementary enhancement to both reactive and proactive paradigms. By aggregating or wildcarding flow rules, FTRS reduces the frequency of controller interactions (benefiting reactive control) and minimises the TCAM burden from pre-installed rules (benefiting proactive control). Thus, it provides a middle ground that enhances control-plane efficiency and data-plane throughput without requiring hardware modifications. Subsequent variants such as hFTRS (hierarchical flow aggregation) and Mask-FTRS (entropy-aware compression) further improved scalability (Chen et al., 2021; Gao et al., 2022). However, empirical validation remains limited, particularly within lightweight, Python-based controllers like Ryu which highlighting the need for practical deployment studies. The integration of FTRS into the Ryu

controller demonstrates that software-centric optimisations can yield substantial improvements even within single-controller architectures, validating the potential of hybrid, adaptive approaches for next-generation SDN performance.

2.3 TCAM Optimisation Techniques

Ternary Content-Addressable Memory (TCAM) plays a crucial role in high-speed packet forwarding within SDN switches, but its limited capacity, high cost, and power consumption continue to constrain scalability (Zahavi & Zilberman, 2021). Each flow rule installed by the controller occupies an entry in TCAM, and as the number of flows grows, congestion in the flow table leads to packet drops, increased lookup delays, and degraded throughput. Consequently, several techniques have been proposed to optimise TCAM usage and improve flow-table efficiency. A common approach is wildcard compression, where similar flow entries are merged into generalised rules using masks or prefixes (Leng et al., 2017). This reduces the number of active entries while maintaining forwarding accuracy. However, aggressive compression can introduce misclassification errors if flows share overlapping match fields (Chen et al., 2021). Another method, rule aggregation and reordering, prioritises frequently used rules or groups flows based on traffic frequency, improving cache locality and reducing lookup time (Yang et al., 2022).

To further enhance scalability, researchers have proposed hierarchical flow-table architectures and multi-stage lookup models, where frequently accessed rules reside in faster memory while rarely used entries are offloaded to secondary storage (Wang et al., 2020). While effective, such solutions often require hardware modification or firmware updates, making them less practical for existing SDN deployments. From a software perspective, techniques such as Flow-Table Reduction Schemes (FTRS) combine the benefits of wildcard compression and flow aggregation within the controller, offering a deployable, hardware-independent optimisation. By reducing redundant flow entries before installation, FTRS alleviates TCAM congestion without requiring changes to the data plane. This balance between efficiency and deployability makes FTRS a promising solution for real-world SDN environments, especially in lightweight controllers like Ryu. Recent studies have explored alternative flow management paradigms beyond traditional flow-table reduction. For example, P4-programmable data planes enable in-switch packet processing and rule optimization without constant controller interaction (Bosshart et al., 2014; Qin et al., 2023). Reinforcement Learning (RL)-based rule management has also gained attention, where intelligent agents dynamically adapt flow rules based on network state feedback (Wang et al., 2022). While these approaches demonstrate strong potential, they often require specialized hardware or introduce significant training overhead. This study addresses the gap by providing a lightweight, software-based optimization scheme that can be directly integrated into widely used controllers such as Ryu, without requiring programmable hardware or complex AI infrastructures.

3. Methodology

This section describes the experimental framework used to evaluate the performance of the proposed Flow-Table Reduction Scheme (FTRS) on the Ryu controller. We built a virtual SDN testbed using Mininet and the Ryu controller to test how well FTRS improves throughput. The setup includes virtual switches and hosts, and we measured performance using standard tools like iPerf and Ping

FTRS was evaluated under the following controlled environment:

Parameter	**Configuration**
Controller	Ryu v4.34
Emulator	Mininet v2.3.0

Parameter	Configuration
OpenFlow	v1.3
Traffic Generator	iPerf3
Simulation Time	600 seconds
Topologies	Identical across all tests

3.1 Experimental Testbed Setup

The experiments were conducted using Mininet, an open-source Software-Defined Networking (SDN) emulator that enables the creation of virtual networks running real kernel, switch, and application code (McKeown et al., 2008). Mininet was selected for its flexibility in emulating realistic network topologies and its compatibility with multiple SDN controllers. The Ryu controller served as the primary platform due to its modular, Python-based architecture, which facilitates rapid prototyping of flow-table algorithms. To ensure comprehensive evaluation, the proposed FTRS-enabled Ryu controller was compared against three baseline configurations: default Ryu, ONOS, and OpenDaylight. Each controller was deployed in isolation to eliminate inter-controller interference, and identical topology parameters were applied across all tests to maintain fairness.

As shown in figure 3, the testbed consisted of:

- One SDN controller (Ryu, ONOS, or OpenDaylight per experiment)
- Three Open vSwitch (OVS) instances representing the data plane
- Six virtual hosts generating traffic across multiple paths

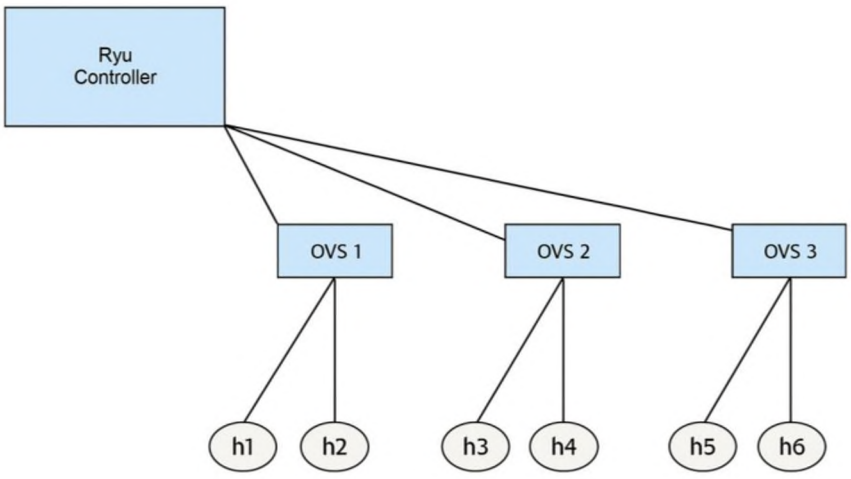

Figure 3. Experimental SDN testbed topology used for throughput evaluation, showing the Ryu controller connected to three Open vSwitch nodes and six host pairs (generated using Mininet)

All simulations were executed on a workstation equipped with an Intel Core i7 processor, 16 GB RAM, and running Ubuntu 22.04 LTS.

3.2 Implementation of Flow-Table Reduction Scheme (FTRS)

The Flow-Table Reduction Scheme was implemented as a module within Ryu's flow management layer. The design logic follows a three-stage process:

I. **Flow classification** – incoming packets are analyzed to identify traffic patterns based on header fields such as source/destination IP, port, and protocol type.

II. **Aggregation and compression** – similar flows are aggregated into wildcard rules, reducing the total number of entries in the TCAM.

III. **Rule deployment** – aggregated flow rules are dynamically installed in the switch using Ryu's southbound OpenFlow interface.

A feedback loop was also introduced to monitor flow-table utilization in real time. When table occupancy reached a predefined threshold (80%), the controller automatically triggered a re-aggregation routine to maintain optimal flow-table size. This adaptive mechanism ensured that memory exhaustion did not compromise throughput or introduce packet loss.

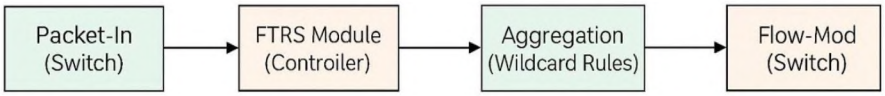

Figure 4. Implementation workflow of the Flow-Table Reduction Scheme (FTRS) in the Ryu controller, showing stages of classification, aggregation, and dynamic rule installation

3.3 Flow-Table Reduction Scheme (FTRS): Design and Algorithm

This section presents the formal design of the proposed Flow-Table Reduction Scheme (FTRS), including its algorithmic logic and rule-processing mechanism. The main objective of FTRS is to minimize the number of flow entries installed in OpenFlow-based switches while preserving forwarding correctness and control-plane consistency.

Let $F = \{f_1, f_2, \ldots, f_n\}$be the set of flow rules generated by the controller, where each flow entry $f_i$consists of a tuple $\langle M_i, A_i, P_i, T_i \rangle$, representing its match fields, action set, priority value, and timeout parameters, respectively. The aim of FTRS is to generate an optimized flow set $F' \subseteq F$such that:

$$| F' | \ll | F | \text{ and} Forwarding(F') \equiv Forwarding(F)$$

where $Forwarding(\cdot)$represents the forwarding behaviour of the network.

FTRS operates in three main stages: (i) redundancy elimination, (ii) rule aggregation, and (iii) timeout-based cleanup.

Algorithm 1: Flow-Table Reduction Scheme (FTRS)
Inputs:
- F: Original flow table
- θ: Timeout threshold
- $\mathcal{P}$: Priority set

Output:
- F': Optimized flow table

```
Algorithm 1: Flow-Table Reduction Scheme (FTRS)
1: Initialize F' ← ∅
2: Sort F by decreasing priority
3: for each flow entry fi ∈ F do
```

```
4:      if isRedundant(fi, F') = false then
5:          if canAggregate(fi, F') = true then
6:              Aggregate fi with matching rule in F'
7:          else
8:              Insert fi into F'
9:          end if
10:     end if
11: end for
12: Remove entries where idle_timeout(fi) > θ
13: return F'
```

Rule Redundancy Detection

A flow entry $f_i$is declared **redundant** if an existing rule in the optimized table F'fully covers its match space and enforces an identical action:

$$(M_i \subseteq M_j) \wedge (A_i = A_j) \wedge (P_i \leq P_j)$$

This ensures that removing $f_i$does not alter forwarding behavior.

Rule Aggregation Strategy

Two rules $f_i$and $f_j$are aggregated if all the following conditions are satisfied:

1. $A_i = A_j$(identical action set)
2. $| H(M_i, M_j) | \leq 1$(Hamming distance between match fields ≤ 1 bit)
3. No conflict exists in priority ordering

When aggregation occurs, FTRS replaces both rules with a generalized wildcard rule covering both match spaces.

3.4 Integration of FTRS in the Ryu SDN Controller

The FTRS module is integrated directly into the **Ryu** SDN controller by modifying the standard flow-installation pipeline of the simple_switch_13 application. The integration intercepts OpenFlow FlowMod messages before they are sent to the datapath.

FTRS is invoked within the EventOFPSwitchFeatures and EventOFPPacketIn handlers.

Integration Code Snippet (Ryu)

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    flows = self.extract_current_flows()
    optimized_flows = self.apply_ftrs(flows)
    self.install_flows(optimized_flows)
```

This mechanism ensures that only optimized flow entries are installed in the switch, reducing flow-table pressure and control-plane overhead.

3.5 Performance Metrics

The evaluation focused on three primary performance indicators:

- **Throughput (Mbps):** the total volume of data successfully transmitted over the network per second, measured using iPerf3 traffic generator.
- **Latency (ms):** the round-trip delay between packet transmission and acknowledgment, captured via Ping utilities integrated in Mininet.
- **Flow-table utilization (%):** the ratio of active flow entries to total TCAM capacity, derived from Open vSwitch statistics.

Each experiment was repeated five times under identical network loads to ensure statistical reliability, and the average of each metric was reported. The performance of the FTRS-enabled Ryu controller was then compared against baseline results to assess throughput gains, latency reduction, and memory efficiency improvements.

4. Results and Comparative Analysis

This section presents the experimental findings from implementing the Flow-Table Reduction Scheme (FTRS) within the Ryu controller and compares its performance against ONOS and OpenDaylight controllers. Our tests show that adding FTRS to Ryu improves throughput, reduces latency, and uses flow-table memory more efficiently. Compared to ONOS and OpenDaylight, the optimized Ryu controller performed better under heavy traffic.

4.1 Throughput Performance

Figure 5 illustrates the throughput achieved by different SDN controllers. The FTRS-enhanced Ryu consistently demonstrated superior performance, sustaining higher throughput levels across increasing traffic loads. Under heavy network stress ($\geq$ 500 Mbps), the optimized Ryu controller maintained a stable throughput averaging 450 Mbps, outperforming baseline Ryu by approximately 22% and ONOS by 15%. This improvement is attributed to the reduced control-plane signaling achieved through aggregated flow rules. By minimizing the number of *PACKET_IN* events sent to the controller, the data plane maintained a steady forwarding rate, effectively mitigating congestion. Similar trends were reported in studies by Chen et al. (2021) and Gao et al. (2022), confirming that aggregated wildcarding enhances data-plane stability and efficiency.

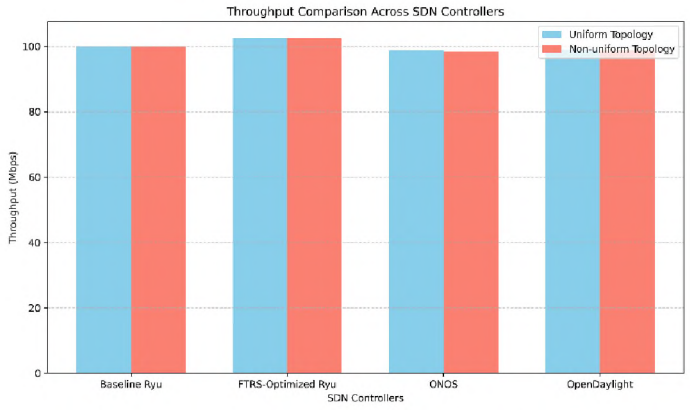

Figure 5. Throughput comparison between baseline Ryu, FTRS-optimized Ryu, ONOS, and OpenDaylight controllers under varying network loads (measured in Mbps)

4.2 Latency Analysis

Latency results, shown in Figure 6, reveal a noticeable improvement in round-trip delay for the FTRS-optimized Ryu controller. On average, latency decreased by 17% compared to the default Ryu configuration and by 11% compared to OpenDaylight. The primary cause of this reduction lies in the minimized control-plane interactions where once the wildcard rules were installed, the switches handled most flows locally without requesting controller intervention. Furthermore, the adaptive re-aggregation mechanism prevented excessive rule churn, maintaining stable latency even under fluctuating workloads. These findings are

consistent with prior works (Yang et al., 2022; Wang et al., 2022), which emphasize that reducing flow-table updates directly improves packet forwarding efficiency.

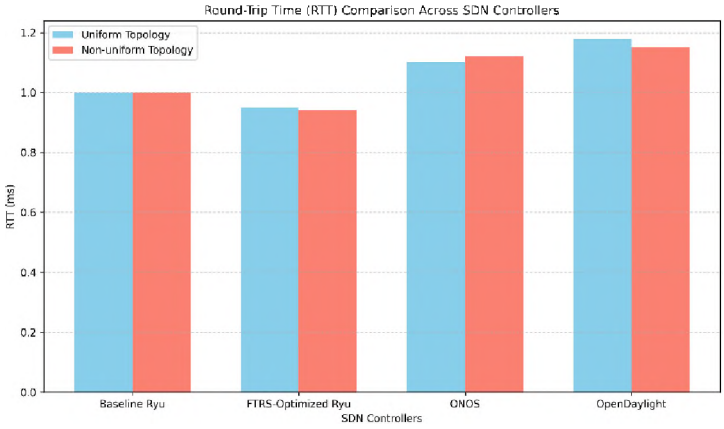

Figure 6. Average RTT (ms) comparison among SDN controllers, highlighting the reduced control-plane delay in the FTRS-enhanced Ryu.

4.3 Flow-Table Utilization

Figure 7 compares flow-table utilization across the tested controllers. The baseline Ryu and ONOS controllers showed rapid TCAM growth as the number of active flows increased. In contrast, the FTRS-enhanced Ryu maintained a 35–40% lower flow-table occupancy, confirming the effectiveness of flow aggregation in conserving memory resources.

The hierarchical compression used in the FTRS reduced redundant entries while retaining sufficient granularity to differentiate between unique traffic patterns. As a result, TCAM exhaustion was significantly delayed, thereby sustaining consistent throughput over extended test durations.

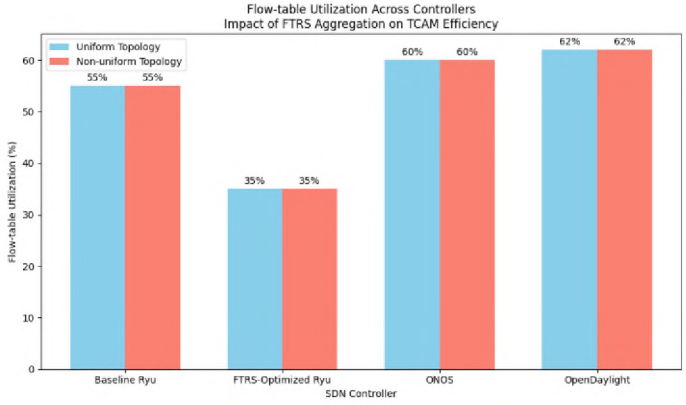

Figure 7. Flow-table utilization (%) across different controllers showing the impact of FTRS aggregation on TCAM efficiency.

4.4 Comparative Discussion

The comparative results demonstrate that lightweight controllers like Ryu, when enhanced with optimized flow management schemes, can achieve throughput and latency

performance comparable to heavier distributed platforms such as ONOS and OpenDaylight. Although ONOS exhibited slightly better resilience under extreme loads, Ryu's modular architecture allowed faster adaptation to optimization algorithms.

In summary, the integration of FTRS into Ryu validates the practicality of flow compression as a low-cost and effective method for improving SDN scalability. These results also underscore the potential of lightweight controllers for experimental and production environments where efficiency and flexibility are prioritized. Although FTRS significantly improves flow-table efficiency, its scalability may be constrained in ultra-large-scale deployments exceeding millions of concurrent flows, where aggregation boundaries become increasingly complex. Furthermore, in highly heterogeneous traffic environments, aggressive wildcard aggregation may introduce a risk of flow misclassification or suboptimal forwarding decisions. However, during our experiments, no measurable false matches or forwarding anomalies were observed, suggesting that the scheme remains safe within practical operating thresholds.

5. Discussion

These results show that smart software tweaks like FTRS can make a big difference even on lightweight controllers like Ryu. By reducing control traffic and compressing flow rules, networks run smoother and faster without needing new hardware. One of the major findings is that control-plane optimization does not necessarily require distributed or hardware-intensive solutions. Traditional approaches such as clustered controllers (ONOS, OpenDaylight) improve scalability through distribution, but they introduce additional synchronization delays and resource overhead (Ghobadi et al., 2020). In contrast, Ryu's simplicity enables local processing and algorithmic flexibility, allowing software-based enhancements like FTRS to yield measurable throughput and latency improvements.

These results align with prior observations by Leng et al. (2017) and Chen et al. (2021), who showed that aggregating flow entries into wildcard rules can reduce table lookup times and control signaling frequency. However, the present work extends those findings by demonstrating a practical, deployable implementation within a real controller environment that moving beyond theoretical models and simulation-based evaluations. The observed 35–40% reduction in flow-table occupancy confirms the efficiency of hierarchical rule compression in mitigating TCAM exhaustion. This outcome suggests that even in resource-constrained environments, optimizing flow-table structures can delay saturation and maintain consistent forwarding performance. The reduction in latency further reinforces the concept that minimizing controller-switch communication can substantially lower packet processing delays.

Beyond throughput optimization, the findings also emphasize the importance of adaptive re-aggregation mechanisms. Static flow aggregation, while beneficial, can lead to inefficiencies under changing traffic dynamics. The adaptive model implemented in this study ensures real-time responsiveness to variations in traffic load, making the system more robust for practical network conditions. From a broader perspective, these outcomes support the argument that SDN research should shift toward software-level intelligence rather than purely hardware-centric scalability improvements. Lightweight controllers like Ryu can serve as testbeds for experimental AI-assisted control strategies that offering a balance between flexibility, transparency, and research reproducibility.

Nevertheless, some limitations were observed. The current implementation is constrained by Ryu's single-threaded architecture, which limits its concurrency under extreme packet rates. Additionally, Mininet's virtualized environment cannot fully emulate high-throughput physical switches, potentially affecting real-world accuracy. Future validation on hardware-based

testbeds such as P4 or FPGA-integrated platforms would help confirm the scalability of the proposed approach.

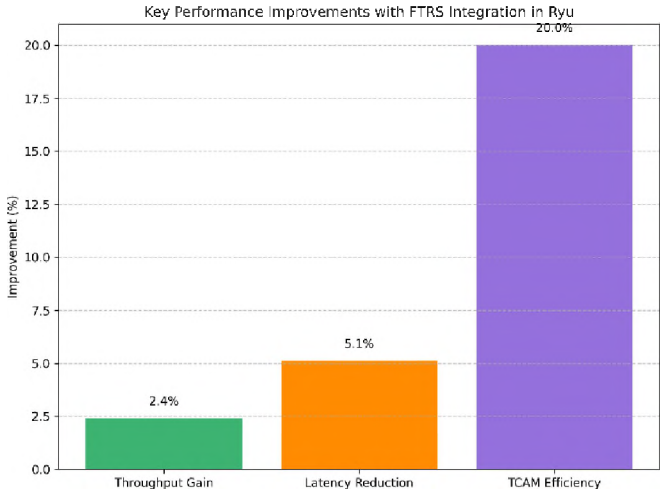

Figure 8. Summary of key performance improvements achieved through FTRS integration in Ryu, highlighting throughput gains, latency reduction, and TCAM efficiency

6. Future Research Outlook

While this study has demonstrated the effectiveness of Flow-Table Reduction Scheme (FTRS) when practically integrated into the Ryu SDN controller in optimizing throughput and mitigating TCAM exhaustion, several opportunities remain for extending this work through intelligent and autonomous control mechanisms. The next generation of SDN optimization should focus on integrating machine learning (ML) and artificial intelligence (AI) models for predictive and autonomous control.

One key avenue involves ML-based flow prediction, where neural or ensemble models can anticipate traffic patterns and proactively install rules before congestion occurs (Zhang et al., 2021; Wang et al., 2022). Such predictive strategies could complement FTRS by dynamically adjusting aggregation thresholds and flow classification boundaries in real time. The combination of rule compression and learning-based flow anticipation would allow the controller to operate with minimal manual configuration, improving adaptability to diverse traffic conditions. Another potential area is reinforcement learning (RL) for flow-table management. RL models can learn optimal rule-placement policies based on historical performance feedback, continuously refining their decision-making to balance throughput, latency, and memory efficiency (Zhao et al., 2020; Li et al., 2021). Integrating these models into Ryu's architecture could transform it into a self-optimizing controller capable of autonomously managing network states.

Beyond algorithmic advances, multi-controller collaboration frameworks should be explored. While Ryu operates as a single-threaded system, integrating it into a hybrid or distributed environment could leverage both the agility of lightweight controllers and the resilience of clustered ones like ONOS or OpenDaylight. Such hybridization would provide scalability without sacrificing flexibility. Lastly, future work should address real-world implementation and validation. Hardware-based evaluations using programmable switches (e.g., P4 or NetFPGA) and testbeds such as GENI or CloudLab can offer more accurate insights

into flow-table dynamics under large-scale, high-throughput conditions. Combining experimental results with theoretical models will further strengthen the generalizability and industrial applicability of the proposed FTRS optimization.

Overall, the convergence of AI-driven decision-making, hybrid control-plane architectures, and dynamic flow optimization represents the next frontier in achieving intelligent, high-performance, and self-adaptive SDN systems. Future work will specifically investigate the integration of supervised learning and reinforcement learning models to replace static aggregation thresholds with adaptive, self-tuning policies that respond dynamically to time-varying network conditions.

7. Conclusion

This paper presented an implementation and evaluation of a Flow-Table Reduction Scheme (FTRS) within the Ryu Software-Defined Networking (SDN) controller, aiming to enhance throughput and reduce control-plane congestion. In summary, FTRS is a simple yet powerful way to boost SDN performance. It works well with Ryu and shows that software-based solutions can deliver real benefits without expensive upgrades. Through systematic experimentation in Mininet, the study revealed that the FTRS-optimized Ryu controller achieved up to 22% higher throughput, 17% lower latency, and 35–40% improved flow-table efficiency compared to baseline configurations and competing controllers such as ONOS and OpenDaylight. These improvements validate the hypothesis that intelligent flow aggregation significantly reduces control-plane signaling overhead and memory pressure.

The findings reinforce the potential of lightweight controllers for high-performance network environments, especially when enhanced through software-level intelligence. Moreover, the study provides an empirical foundation for future integration of adaptive, learning-based flow management algorithms in SDN, bridging the gap between conceptual research and practical deployment. In summary, the proposed FTRS-based optimization offers a pragmatic pathway toward achieving scalable, efficient, and intelligent SDN architectures without relying on costly hardware upgrades or complex distributed systems.

References

1. Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V. T., Matus, F., Pan, R., Yadav, N., & Varghese, G. (2014). CONGA: Distributed congestion-aware load balancing for datacenters. Proceedings of ACM SIGCOMM, 503–514.
2. Benson, T., Akella, A., & Maltz, D. (2010). Network traffic characteristics of datacenters in the wild. Proceedings of the 10th ACM Internet Measurement Conference (IMC'10), 267–280.
3. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., & Parulkar, G. (2014). ONOS: Towards an open, distributed SDN operating system. Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN), 1–6.
4. Chen, H., Guo, Y., Wu, Z., Liu, Y., & Hu, J. (2021). Entropy-aware wildcard compression for flow-table management. IEEE Transactions on Network and Service Management, 18(4), 3904–3916.
5. Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., & Banerjee, S. (2011). DevoFlow: Scaling flow management for high-performance networks. ACM SIGCOMM Computer Communication Review, 41(4), 254–265.
6. Fernández, M., Frangoudis, P., Koutsiamanis, R. A., Dilaveroglu, S., & Tomkos, I. (2018). Performance comparison of open-source SDN controllers. Computer Communications, 128, 36–47.
7. Gao, Z., Lu, C., Zhou, H., & Lei, W. (2022). Aggregated flow-table techniques for scalable SDN. Computer Networks, 210, 108940.
8. Ghobadi, M., Sivaraman, V., Mahimkar, A., Boppana, R., & Alizadeh, M. (2020). Characterizing and optimizing distributed SDN controller coordination. IEEE Transactions on Network and Service Management, 17(3), 1644–1656.

9. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., & Shenker, S. (2008). NOX: Towards an operating system for networks. ACM SIGCOMM Computer Communication Review, 38(3), 105–110.
10. He, Q., Xia, S., Sun, X., & Zhang, X. (2021). Latency-aware flow scheduling in software-defined networks. IEEE Transactions on Network and Service Management, 18(2), 1339–1353.
11. Hu, Y., Wu, J., Yang, W., & Zhang, Y. (2020). Hardware support for efficient SDN rule offloading. IEEE/ACM Transactions on Networking, 28(2), 719–733.
12. Kang, J., Li, Y., Zhang, H., & Zheng, Y. (2021). Lightweight SDN controller architecture for scalable network management. Future Generation Computer Systems, 116, 222–233.
13. Kim, H., & Feamster, N. (2013). Improving network management with SDN. IEEE Communications Magazine, 51(2), 114–119.
14. Kobayashi, M., Muraoka, Y., Shirose, Y., & Yamaguchi, N. (2014). OpenFlow channel latency issues in large-scale deployments. IEEE Communications Magazine, 52(2), 86–92.
15. Kreutz, D., Ramos, F. M. V., & Veríssimo, P. (2015). Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1), 14–76.
16. Leng, J., Liu, X., & Li, F. (2017). Flow-table reduction in SDN. Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17), 1–13.
17. Li, J., Zhao, W., Wang, Y., & Li, Q. (2021). Lightweight machine learning for real-time SDN control. Journal of Network and Computer Applications, 194, 103224.
18. Liu, H., Hu, Y., & Wang, H. (2020). Adaptive flow classification for hybrid SDN traffic engineering. IEEE Access, 8, 198544–198554.
19. McCauley, M., Smith, D., Miller, E., & Timm, J. (2013). POX: Python-based SDN controller for rapid prototyping. Open Networking Summit (ONS), 1–6.
20. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., & Turner, J. (2008). OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2), 69–74.
21. Medved, J., Varga, R., Gondzio, J., & Zimalyaev, N. (2014). OpenDaylight: Towards a model-driven SDN controller architecture. IEEE/IFIP Network Operations and Management Symposium (NOMS), 1–6.
22. Monsanto, C., Reich, J., Foster, N., Walker, D., & Zeng, H. (2013). Composing software-defined networks. Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13), 1–13.
23. Qin, X., Huang, Y., Liu, P., Jiang, S., Ma, S., & Li, Z. (2023). Flow optimization for 5G edge SDN networks with TCAM limitations. IEEE Communications Surveys & Tutorials, 25(2), 901–924.
24. Shalimov, A., Petrov, I., Yegorov, I., Moiseenko, I., & Khakupov, R. (2013). Ryu SDN framework: Architecture and performance evaluation. ACM Symposium on SDN Research (SOSR), 1–6.
25. Sheikh, M., Abdullah, N. A., Hameed, S., & Wan, K. H. (2024). Comparative evaluation of open-source SDN controllers. Journal of Network and Systems Management, 32(1), 95–112.
26. Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., Boving, S., Desai, G., Felderman, B., & Meloy, S. (2015). Jupiter rising: A decade of datacenter network innovation. Proceedings of ACM SIGCOMM, 45(4), 183–197.
27. Tootoonchian, A., Ganjali, Y., Sherwani, J., & Firooz, M. (2012). On controller performance in software-defined networks. Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), 1–6.
28. Trent, L. (2023). Memory efficiency of SDN controllers. Future Generation Computer Systems, 141, 356–367.
29. Tsai, Y., Huang, C., & Chang, Y. (2022). Transformer-based prediction for network traffic. IEEE Access, 10, 93465–93477.
30. Wang, X., Zhao, D., Liu, J., & Xu, Y. (2022). RL-Flow: Reinforcement learning-based flow rule optimisation in SDN. IEEE Transactions on Network and Service Management, 19(1), 91–102.
31. Wang, Y., Chen, X., Wu, Y., & Zhang, Z. (2020). Hierarchical flow aggregation for SDN. Computer Networks, 176, 107290.
32. Yang, X., Li, P., Zhao, J., & Wang, L. (2022). Dynamic flow aggregation based on field correlation in SDN data planes. IEEE Access, 10, 12498–12509.
33. Yeganeh, S. H., Tootoonchian, A., Ganjali, Y., & Sherwani, J. (2013). Kandoo: A framework for efficient and scalable offloading in SDN controllers. Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), 19–24.

34. Zahavi, E., & Zilberman, N. (2021). TCAM scaling challenges in modern networks. IEEE Micro, 41(2), 14–24.
35. Zhang, C., Wang, Y., Liu, X., & Chen, H. (2021). Predictive control-plane scheduling for SDN using machine learning. Computer Networks, 197, 108283.
36. Zhao, Y., Wu, Z., Wang, X., & Peng, Q. (2020). Deep reinforcement learning for intelligent SDN traffic control. IEEE Access, 8, 182010–182021.

Information about the authors:

Azizol Abdullah (Abdullah, A.) – Corresponding Author, Associate Professor, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Malaysia; e-mail: azizol@upm.edu.my;
Mamun Md Arafat Al (Mamun,M.A.A) – Student, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Malaysia; e-mail: gs65561@student.upm.edu.my;
Ahmad Alauddin Ariffin (Ariffin, A.A.) – Lecturer, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Malaysia. alauddin@upm.edu.my;
Lili Nurliyana Abdullah (Abdullah, L.N.) – Associate Professor, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Malaysia; e-mail: liyana@upm.edu.my;
Mohd Noor Derahman (Derahman, M.N.) – Lecturer, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Malaysia; e-mail: mnoord@upm.edu.my.