JURNAL INFOTEL
VOL. 18, NO. 3, AUGUST 2026, PP. 411–437. DOI:10.20895/INFOTEL.V18I3.10799

RESEARCH ARTICLE

Characterizing a Flow Count-Based Three-Tier Timeout Policy for Flow-Table Management in Software-Defined Networking

Mukhamad Salman Nurdin[1], and Galura Muhammad Suranegara[2,*]

[1,2]Telecommunication Systems Study Program, Indonesian University of Education, Bandung, 40154, Indonesia

*Corresponding email: galurams@upi.edu

Received: March 12, 2026; Revised: September 02, 2026; Accepted: September 08, 2026.

Abstract: Software-Defined Networking (SDN) relies on finite switch flow tables, making flow-entry lifetime management relevant to table utilization and controller interaction. This study implements and empirically characterizes a simple, rule-based, three-tier timeout policy in POX, an open-source Python-based SDN controller. A controller-maintained flow-count indicator assigns recorded flow-installation decisions to LOW, MEDIUM, or HIGH, each with predefined idle- and hard-timeout values. The policy was examined in Mininet under nine many-to-one Transmission Control Protocol (TCP) scenarios with 50–500 total hosts, comprising one server and the remaining clients. Reported metrics include cumulative flow-installation events, timeout-category distributions, Packet-In and FlowRemoved counts, controller-derived peak concurrent-flow estimates, and installation events per host. Cumulative flow-installation activity was non-monotonic, reaching its highest observed value of 7,931 events at 350 hosts. HIGH decisions dominated at 50 hosts (65.7%) and reached its highest proportion of 93.7% at 350 hosts. Packet-In activity was also non-monotonic, reaching 10,599 events at 350 hosts, whereas FlowRemoved reached 700 events at 450 hosts. These findings characterize the tested configuration but do not establish optimal capacity or comparative superiority because comparative baselines, repeated trials, controller-resource measurements, independent switch-occupancy validation, and service-level performance measurements were not included.

Keywords: Software-Defined Networking, Flow-table management, Flow-count-based timeout, Three-tier timeout policy, POX controller

© Jurnal Infotel ISSN: 2085-3688; e-ISSN: 2460-0997

1 Introduction

Software-Defined Networking (SDN) separates the control plane from the data plane, enabling forwarding policies and network-control decisions to be managed through a programmable controller [1]. This architecture supports centralized and flexible network management. However, increasing the number of hosts, connections, and forwarding rules can increase the controller interaction and the demand for the flow-table. In reactive OpenFlow-based operations, packets without matching forwarding entries require controller involvement before new rules can be installed. Consequently, flow-entry management is relevant for both the use of finite flow-table resources and the amount of controller-switch interaction [2, 3].

The lifetime of an OpenFlow flow entry is commonly controlled by idle-timeout and hard-timeout parameters. The idle timeout removes an entry after a specified period without matching traffic, whereas the hard timeout limits its maximum lifetime regardless of packet activity [4]. Their selection involves a trade-off. Longer timeout values retain entries for longer periods, but may also preserve entries that are no longer required. Shorter timeout values release entries sooner, but subsequent traffic associated with an expired rule may generate additional table misses, Packet-In events, and flow reinstallation [5,6]. Therefore, the timeout configuration influences flow-entry persistence and controller interaction, while its effect on throughput, delay, packet loss, and service continuity requires additional service-level measurements.

Previous studies have addressed flow-table management through static and dynamic timeout assignment, traffic-aware mechanisms, proactive installation, flow eviction, and learning-based approaches [7, 8]. These approaches differ in the information used for decision making and in the complexity of their control logic. More elaborate mechanisms can incorporate traffic characteristics, flow-table conditions, or learned state, whereas simpler rule-based mechanisms can operate using fewer controller-side indicators [9]. Nevertheless, neither adaptivity nor algorithmic complexity alone establishes superior network performance, such conclusions require direct comparative evaluation under equivalent experimental conditions.

Within this design space, the present study investigates a simple, rule-based, three-tier timeout policy implemented in POX, an open-source Python-based SDN controller. The policy uses a controller-maintained flow_count indicator to assign each recorded flow-installation decision to LOW, MEDIUM, or HIGH, with each category associated with a predefined idle-timeout and hard-timeout pair. The mechanism is categorical rather than continuously optimized and does not employ traffic prediction or machine learning. The thresholds and timeout pairs used in the original implementation were manually selected rather than derived from formal optimization, sensitivity analysis, or measured hardware-capacity limits [10, 11]. They are therefore treated as heuristic configuration parameters of the evaluated policy.

The research gap addressed in this study is deliberately specific. Adaptive timeout management itself is not new, and neither the use of three categories nor the absence of machine learning is claimed as sufficient novelty. Instead, this work focuses on implementing and empirically characterizing a single-indicator, flow-count-driven categorical timeout policy in POX and jointly examining its timeout decisions and controller-level flow-management events across different host-count scenarios. The contribution is therefore positioned as the

implementation and characterization of a specific rule-based configuration rather than as evidence of state-of-the-art or comparative performance.

Based on this scope, the study addresses the following research question: How does the flow-count-driven three-tier timeout policy behave, in terms of recorded timeout-category decisions and controller-level flow-management events, as the total number of emulated hosts varies across the tested POX/Mininet scenarios?.

The policy was evaluated in Mininet using a single-switch, many-to-one Transmission Control Protocol (TCP) traffic configuration under nine reported host-count scenarios: 50, 100, 150, 250, 300, 350, 400, 450, and 500 total hosts. Host h_1 acted as the TCP server, while hosts h2 through hNacted as clients. The experimental host count N and the controller-maintained flow_count are distinct quantities: N defines the host-count scenario, whereas flow_count is the internal indicator used for timeout selection.

The reported observations include cumulative flow-installation events, timeout-category decisions, Packet-In events, FlowRemoved events, a controller-derived estimate of peak concurrent flows, and flow-installation events per host. These quantities characterize controller and flow-management behavior rather than direct end-to-end network performance. The original experiment did not include comparative timeout baselines, repeated trials for statistical aggregation, independent validation of flow_count against switch-table occupancy, controller-resource measurements or service-level metrics such as throughput, delay, packet loss, and connection success. Consequently, the present study is limited to an empirical characterization of the tested configuration and does not claim comparative superiority, optimal capacity, improved controller stability, controlled saturation, or graceful degradation.

Accordingly, the objectives of this study are to implement and document the flow-count-driven three-tier timeout policy, characterize how its timeout decisions and recorded controller-level events vary across the tested host-count scenarios, and identify the highest observed values and non-monotonic patterns of the reported metrics without interpreting them as evidence of an optimal operating point. The main contributions of this study are:

1. A documented rule-based three-tier timeout policy that uses a controller-maintained flow-count indicator to select predefined LOW, MEDIUM, and HIGH timeout configurations.
2. An implementation and experimental characterization of the policy in a POX/Mininet environment across nine reported many-to-one TCP host-count scenarios.
3. A joint descriptive analysis of timeout-category decisions, cumulative flow-installation events, Packet-In events, FlowRemoved events, controller-derived peak concurrent flows, and flow-installation events per host.
4. A clear separation between directly reported experimental observations and conclusions that would require comparative baselines, repeated trials, independent switch-occupancy validation, controller-resource measurements, or service-level performance metrics.

2 Related Works

Flow-table management in Software-Defined Networking (SDN) has been investigated through several complementary strategies, including timeout assignment, proactive flow

installation, rule caching, flow eviction, and learning-based control [12]. These approaches address related resource-management problems, but differ in the information used for decision making and in the stage at which flow-table behavior is controlled. Because the present study modifies the flow-entry lifetime through a controller-maintained flow-count indicator, this section focuses primarily on timeout management approaches while discussing complementary techniques only to clarify the position of the investigated policy [13].

2.1 Flow-Entry Lifetime and Flow-Table Management

In OpenFlow-based SDN, forwarding rules are maintained as flow entries in switch flow tables. Flow-entry lifetime is commonly controlled using idle-timeout and hard-timeout parameters. The idle period removes an entry after a specified period without matching traffic, whereas the hard period limits the total lifetime of an entry regardless of the continued activity [7, 13].

The timeout configuration involves a trade-off. Longer timeout values can reduce repeated installation for persistent traffic, but may retain entries that are no longer required. Shorter timeout values release entries sooner, but may result in additional controller interaction when traffic continues after an entry has expired. The practical result therefore depends on the characteristics of the traffic, the behavior of the controller, the resources of the switch-table, and the performance metric being considered [14, 15].

Dynamic timeout mechanisms have been proposed as alternatives to applying one fixed timeout configuration to all entries. For example [16], introduced a dynamic adaptive timeout approach for SDN switches. Subsequent studies have incorporated additional traffic characteristics, flow-table information, or learning mechanisms into timeout decisions [17].

These studies establish that adaptive timeout management itself is not a new concept. Accordingly, the present work does not claim novelty from dynamically changing timeout values or from the use of multiple timeout categories [18].

2.2 Adaptive Timeout Mechanisms

Adaptive timeout mechanisms differ primarily in the information used to determine the lifetime of an entry. Isyaku et al. proposed the Adaptive and Hybrid Idle-Hard Timeout Allocation (AH-IHTA) mechanism, which considers traffic patterns and flow-table utilization when determining timeout values and combines timeout allocation with flow eviction [18]. Their evaluation reports reductions in flow-setup requests and flow eviction compared with the configurations evaluated in that study. HQTimer uses a different strategy by combining hybrid timeout management with Q-learning. The mechanism assigns timeout values according to traffic dynamics and data-plane performance, and its evaluation considers table-hit rate and flow-table overflow [17, 19].

Harris and Khorsandroo proposed Delayed Dynamic Timeout (DDT) [20], which uses reinforcement learning for dynamic flow-timeout assignment. The reported implementation uses Ryu, Mininet, Open vSwitch, and OpenFlow, and the method relies on network-state information and learning logic rather than a single predefined threshold rule [6,18,21].

These approaches illustrate a range of adaptive designs. Some mechanisms use several traffic or table variables, whereas learning-based approaches additionally require a state representation and adaptation logic. The policy studied in this paper instead uses only the controller-maintained $flow_count$ indicator and two predefined thresholds to select among

three timeout configurations [22]. This distinction describes the information and decision structure of the mechanisms. It does not demonstrate that the present policy has lower computational cost or better network performance, because computational overhead and comparative performance were not measured in the original experiment.

2.3 Complementary Flow-Table Management Techniques

Timeout assignment is only one mechanism for managing finite flow-table resources. Other techniques change when rules are installed, how rules are represented, or which existing rules are removed. Cai et al. proposed Dynamic Adjustment for Proactive Flow Installation Mechanism (DAPFIM) for SDN-based Internet of Things environments [18]. DAPFIM detects traffic transmission periods and adjusts proactive rule installation. Its evaluation includes flow-entry duration, hit-related measures, flow-entry efficiency, and message activity. This method therefore controls when rules are installed rather than using the same categorical timeout-selection mechanism investigated in the present work.

Rule caching provides another approach. Adaptive wildcard rule-cache management represents related forwarding conditions using wildcard entries and therefore addresses flow-table usage through rule representation rather than directly through timeout assignment [23–25].

Flow eviction determines which existing entries should be removed when table resources become constrained. Flow-table overflow-prevention systems such as FTOP address this broader problem through monitoring, detection, and selective management of entries [26]. FTOP is therefore relevant to the flow-table-management context but is not treated as a direct experimental baseline for the timeout policy studied here. The published FTOP work specifically addresses flow-table overflow prevention rather than categorical timeout assignment [27].

These distinctions are important because timeout assignment, proactive installation, rule caching, and eviction represent different interventions. Their reported results should not be interpreted as directly comparable unless they are implemented and evaluated under equivalent experimental conditions.

2.4 Comparison with Representative Previous Studies

Table 1 summarizes representative approaches using the dimensions relevant to the reviewer-requested comparison. The comparison is methodological rather than experimental, because none of these approaches was re-implemented under the same topology, traffic configuration, controller, and measurement procedure as in the present study.

Table 1: Comparison of representative timeout and flow-table-management approaches

Study	Controller / environment	Mechanism	Adaptation indicator	Topology / scale	Traffic	Evaluation focus	Decision processing requirement	Limitation / scope
[16]	Not specified here beyond the verified publication record	Dynamic adaptive timeout	Dynamic network/flow condition	Not reproduced in the present study	Not reproduced in the present study	Dynamic timeout management	Rule-based adaptation; formal complexity not reported here	Relevant early adaptive-timeout work, but not an experimental baseline in this study
[19]	SDN/OpenFlow experimental environment	Adaptive idle-hard timeout with eviction	Traffic pattern and flow-table utilization	Experimental SDN setting	Flow-setup requests and flow eviction	Multiple traffic/table observations; formal complexity not reported	Uses additional traffic/table information and includes eviction	Traffic-based evaluation
[7]	SDN experimental evaluation	Hybrid timeout with Q-learning	Traffic dynamics and data-plane performance	Real and synthetic workloads	Real and synthetic workloads	Table-hit rate and overflow	Q-learning-based adaptation	Learning-based timeout mechanism rather than fixed categorical thresholds
[20]	Ryu, Mininet, Open vSwitch	Reinforcement-learning dynamic timeout	Network-state information and learned policy	Reported emulated SDN environment	Generated network traffic	Dynamic timeout behavior and flow-rule activity	Reinforcement learning processing	Requires learned state and additional control logic

Continued on next page

Table 1 – continued from previous page

Study	Controller / environment	Mechanism	Adaptation indicator	Topology / scale	Traffic	Evaluation focus	Decision processing requirement	Limitation / scope
[18]	SDN-IoT simulation	Dynamic proactive flow installation	Detected transmission period and timing adjustment	Six reported simulation stages	Multiple traffic-pattern conditions	Flow-entry duration, hit-related measures, efficiency, messages	Transmission-period detection and adjustment	Changes installation timing rather than applying the same timeout-selection rule
[27]	SDN flow-table-management environment	Overflow prevention	Flow-table/traffic conditions	Overflow-oriented evaluation	Overflow-related traffic conditions	Overflow prevention and mitigation	Multiple detection/prevention components	Addresses overflow protection rather than timeout assignment
Present study	POX 0.7.0 and Mininet in one virtual environment	Three-tier categorical timeout	Controller-maintained `flow_count` only	1 controller, 1 switch, 50–500 total hosts; 1 server and $N-1$ clients	Many-to-one TCP using iperf	Timeout decisions, cumulative installation events, Packet-In, FlowRemoved, controller-derived peak concurrent flows, events per host	Two threshold comparisons; no formal computational benchmark	No comparative baseline, repeated-trial statistics, resource/service metrics, sensitivity analysis, or independent switch-occupancy validation

As summarized in Table 1, representative approaches differ substantially in both the information required for decision making and the problem being addressed. AH-IHTA incorporates traffic and flow-table information, HQTimer and DDT use learning-based adaptation, DAPFIM modifies proactive installation behavior, and FTOP addresses flow-table overflow protection. In contrast, the present policy uses one controller-maintained indicator to select among three predefined timeout pairs [7, 20, 27].

This difference should not be interpreted as evidence that the present mechanism is computationally superior. The experiment did not measure computational complexity, CPU utilization, or memory overhead and therefore supports only a comparison of mechanism structure and required decision information.

2.5 Research Gap and Position of the Present Study

The comparison in Table 1 shows that adaptive timeout and flow-table management already encompass a range of rule-based, traffic-aware, proactive, and learning-based mechanisms. Therefore, this study does not claim novelty from adaptive timeout management itself, from the existence of three timeout categories, or from avoiding machine learning.

The research gap addressed here is narrower. The study examines a categorical timeout policy in which a single controller-maintained flow_count indicator selects one of three predefined timeout configurations. The experiment characterizes how the recorded timeout decisions and controller-level flow-management events of this specific configuration vary across the many-to-one host-count scenarios tested [28].

The thresholds of 100 and 500 and the corresponding timeout pairs were selected heuristically in the original implementation. They were not obtained through formal optimization or sensitivity analysis and are therefore treated as configuration parameters rather than general recommendations.

The available measurement scope also differs from several previous studies. The present dataset contains timeout categories, cumulative flow-installation events, Packet-In events, FlowRemoved events, a controller-derived estimate of peak concurrent flows, and flow-installation events per host. It does not contain comparative timeout baselines, repeated-trial statistics, controller-resource measurements, service-level metrics, or independently validated switch-table occupancy.

Accordingly, the contribution of this work is the implementation and empirical characterization of one flow-count-driven categorical timeout configuration in POX. The study does not establish that this configuration outperforms fixed-timeout, traffic-aware, learning-based, or other flow-table-management mechanisms.

3 Research Method

This section describes the experimental design, implementation, traffic configuration, data collection, and measurement scope used to characterize the flow-count-driven three-tier timeout policy. The methodology reports only procedures and parameters supported by the surviving experimental documentation. Parameters or measurements that cannot be verified from the available records are explicitly identified rather than retrospectively reconstructed.

3.1 Experimental Design and Scope

The experiment characterizes the behavior of a predefined three-tier timeout configuration while varying the total number of emulated hosts. The independent experimental variable is the total host count, denoted by N, with nine reported scenarios:

$$N \in \{50, 100, 150, 250, 300, 350, 400, 450, 500\} \tag{1}$$

The three-tier thresholds, timeout pairs, controller implementation, single-switch topology, multiple-to-one traffic pattern, and approximate test duration were maintained as documented across the reported scenarios. The recorded observations consist of cumulative flow-installation events, Packet-In events, FlowRemoved events, timeout-category decisions, a controller-derived estimate of peak concurrent flows, and a descriptive flow-installation-events-per-host ratio. Their methodological roles and interpretation limits are summarized in Table 4.

The final reported dataset contains an observation for each host-count scenario. Repeated trials of the same configuration were not available for statistical aggregation. Therefore, means, standard deviations, confidence intervals, and statistical significance tests are not reported.

No fixed-timeout, default-POX, two-tier, or other adaptive timeout policy was evaluated under identical experimental conditions. Consequently, the study does not test a comparative hypothesis of superiority and does not define a comparative success criterion such as improved scalability or controller stability. The experiment is therefore treated as an empirical characterization of a single-configuration.

3.2 Emulation Environment and System Architecture

The experiments were conducted using Ubuntu 20.04 running in an Oracle VirtualBox virtual-machine environment. The original experimental description records an x86-64 multi-core computing platform with at least 8 GB of memory. However, the exact processor model, physical-memory configuration, virtual CPU allocation, and virtual-machine memory allocation are not preserved for independent verification.

POX version 0.7.0 was used as the Software-Defined Networking controller. The investigated timeout policy was implemented in the custom $adaptive_learning.py$ module within the POX workflow. Mininet version 2.3 was used to construct the emulated hosts and switch. Python 3.10 was used in the documented experimental and analysis workflow together with NumPy, Pandas, and Matplotlib.

A preserved execution trace shows that POX 0.7.0 issued a compatibility warning because Python 3.10 was outside the versions listed formally by that release. However, the same trace shows that POX started and that the Mininet switch established a connection with the controller. The possible effect of this version mismatch was not independently evaluated.

The controller, Mininet topology, virtual switch, emulated hosts, and traffic-generation processes operate within the same virtual-machine environment. Therefore, the behavior observed at larger host counts cannot be exclusively attributed to the timeout policy because controller processing, virtualization, emulator behavior, and traffic-generation effects were not measured separately. The documented topology consists of one POX controller,

one switch s1, and N hosts directly connected to the switch. Host h_1 acts as the TCP server, whereas hosts h_2 through h_N act as clients. Thus,

$$N_{\text{clients}} = N - 1, \tag{2}$$

the preserved 50-host execution trace, for example, records one iperf server on h_1, 49 clients, and an approximately 25s test interval. Table 2 summarizes the experimental environment and distinguishes between configuration information supported by the surviving records and low-level parameters that are no longer available for verification.

Table 2: Experimental environment and preserved configuration information

Component	Recorded configuration	Verification status / limitation
Operating system	Ubuntu 20.04	Preserved in experiment description
Virtualization	Oracle VirtualBox	Preserved
Controller	POX 0.7.0	Preserved
Custom module	$adaptive_learning.py$	Preserved
Network emulator	Mininet 2.3	Preserved in documented workflow
Programming language	Python 3.10	Preserved; compatibility warning recorded
Analysis tools	NumPy, Pandas, Matplotlib	Peserved in documented workflow
Computing platform	x86-64, multi-core	Exact processor model unavailable
Memory	At least 8GB in original description	Exact physical/VM allocation unavailable
Open vSwitch version	Not preserved	Not reconstructed
OpenFlow version	Not preserved with sufficient certainty	Not reconstructed
Link bandwidth and delay	Not preserved	Not reconstructed
Queue configuration	Not preserved	Not reconstructed
CPU/RAM monitoring	Not performed	No resource-performance analysis

As shown in Table 2, missing low-level parameters are reported as unavailable rather than inferred from software defaults or reconstructed from memory.

3.3 Flow-Count-Driven Three-Tier Timeout Policy

The investigated mechanism is a categorical rule-based timeout policy. Its decision input is the controller-maintained variable flow_count. The mechanism does not predict the flow lifetime, inspect the payloads of the packets, perform traffic forecasting, or train a machine-learning model. In this paper, adaptive refers specifically to the selection of one of three predefined timeout configurations according to the controller-maintained flow-count indicator. It does not refer to continuous optimization or learning parameter adjustment.

3.3.1 Controller-Maintained Flow-Count Indicator and Load Classification

The original implementation maintains flow_count as an internal controller indicator intended to represent entries considered active by the controller. It is used as input to the timeout-selection rule. The original experiment did not independently synchronize the flow_count with direct periodic measurements of switch-table occupancy. Accordingly, flow_count is treated throughout this paper as a controller-maintained indicator rather than as an independently validated measurement of physical switch-table occupancy. The documented classification rule is:

$$L(c) = \begin{cases} \text{LOW}, & c < 100 \\ \text{MEDIUM}, & 100 \leq c \leq 500 \\ \text{HIGH}, & c > 500 \end{cases} \tag{3}$$

where c denotes the flow-count indicator maintained by the controller used for timeout selection. The thresholds of 100 and 500 were manually selected in the original implementation to form three categorical decision regions. The surviving experimental records do not indicate that these thresholds were derived from formal optimization, sensitivity analysis, measured TCAM capacity, or systematic parameter tuning. They are therefore treated as heuristic configuration parameters, not as optimal or generally recommended thresholds. The timeout values corresponding to the three categories are summarized in Table 2.

3.3.2 Timeout Assignment

Each timeout category is associated with one predefined idle-timeout and hard-timeout pair. The complete configuration documented for the original experiment is summarized in Table 3.

Table 3: Three-tier timeout configuration used in the original experiment

Category	Flow-count condition	Idle timeout (s)	Hard timeout (s)
LOW	(c < 100)	10	30
MEDIUM	($100 \leq c \leq 500$)	7	20
HIGH	(c > 500)	5	15

As shown in Table 3, the configured timeout values become shorter from LOW to MEDIUM and HIGH. LOW uses an idle timeout of 10s and a hard timeout of 30s, MEDIUM uses 7s and 20s, and HIGH uses 5s and 15s.

These timeout pairs were manually selected during the original implementation. No alternative timeout combinations were evaluated in the final reported experiment. Therefore, the study does not claim that the values in Table 3 are optimal, robust to parameter changes, or superior to other possible configurations. The documented relationship is the following.

$$c \rightarrow L(c) \rightarrow (\tau_{\text{idle}}, \tau_{\text{hard}}) \rightarrow \text{flow installation} \tag{4}$$

A preserved execution trace provides direct evidence that a category-specific timeout assignment occurred during execution. For example, events logged with Count=101,

Count=102 and subsequent values were classified as MEDIUM and assigned the 7s/20s timeout pair.

3.3.3 Documented Event Logic

The documented controller logic maintains the internal flow-count indicator in response to flow-installation and FlowRemoved events. For a recorded flow-installation event, the documented intended counter update is

$$ct + 1 = ct + 1 \tag{5}$$

whereas a processed FlowRemoved event is documented to decrement the counter:

$$ct + 1 = ct - 1 \tag{6}$$

These equations represent the documented intended behavior of the original implementation. They are not presented as a new line-by-line verification of the unavailable source code.

The surviving documentation supports timeout selection at a flow-installation decision. In making such a decision, the controller reads the maintained indicator c, evaluates the category defined in Eq.(3), and applies the corresponding timeout pair in Table 3. The available records do not document continuous recalculation or retrospective modification of timeout values for entries that have already been installed.

The timeout-selection procedure supported by the available documentation is illustrated in Figure 1. The selected category and associated controller-event information are recorded after the timeout pair is selected. A preserved execution trace confirms this categorical behavior; for example, events logged with Count=101 and subsequent values were classified as MEDIUM and assigned the 7s/20s timeout pair.

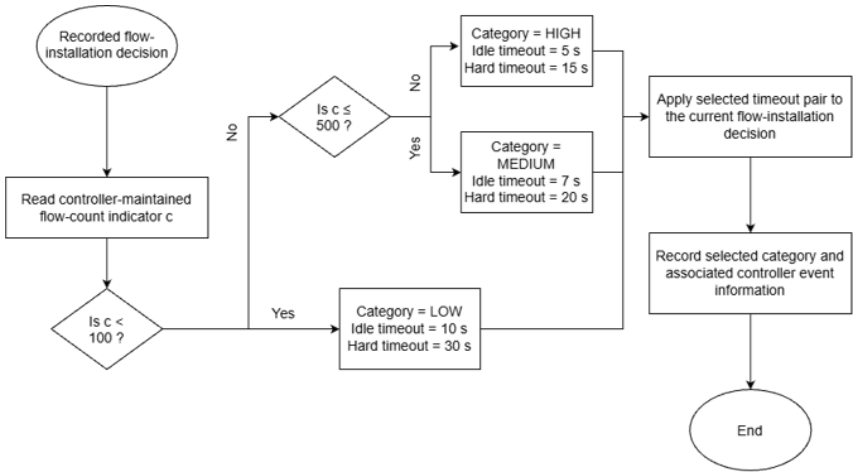

Figure 1: Documented timeout-selection rule of the flow-count-driven three-tier policy.

Figure 1 intentionally represents only the timeout-selection logic supported by the surviving experimental documentation. The original experiment did not independently rec-

oncile the controller-maintained counter with periodic switch-table measurements. Consequently, possible effects from duplicate installation events, failed installations, delayed FlowRemoved notifications, or other controller-switch mismatches cannot be quantified from the surviving evidence.

Likewise, the available records do not provide sufficient evidence to establish the exact within-handler ordering between category evaluation and counter update or to confirm the use of an explicit hysteresis or smoothing mechanism around the thresholds. Therefore, these implementation details are not retrospectively reconstructed.

Consequently, the term adaptive in this study refers to categorical timeout selection at a flow-installation decision based on the current flow-count indicator maintained by the controller. It does not imply continuous optimization, predictive adaptation, or learned control.

3.3.4 Timeout-Category Counts and Percentages

For a scenario that contains T recorded installation decisions, the number assigned to category i is

$$C_i = \sum_{t=1}^{T} 1(L_t = i), \tag{7}$$

where $i \in \text{LOW}, \text{MEDIUM}, \text{HIGH}$, and 1(.) is an indicator function. The corresponding percentage is:

$$P_i = \frac{C_i}{C_{\text{LOW}} + C_{\text{MEDIUM}} + C_{\text{HIGH}}} \times 100\% \tag{8}$$

These percentages represent the distribution of recorded flow-installation decisions assigned to each category. They do not represent the percentage of wall-clock time during which the network remained in LOW, MEDIUM, or HIGH.

For example, a HIGH percentage represents the proportion of recorded installation decisions assigned to the HIGH timeout pair defined in Table 3. It does not independently demonstrate higher throughput, better scalability, controller stability, or service quality. A second distinction is essential:

$$N \neq c, \tag{9}$$

the total host count N defines the experimental scenario, while C is the controller-maintained indicator used for timeout selection. Therefore, a 50-host scenario can contain a majority of HIGH decisions when the controller-maintained indicator exceeds the HIGH threshold during a large proportion of recorded installation decisions.

3.4 Network Topology and Traffic Configuration

The experiment used a single-switch star topology in Mininet. All N hosts were connected directly to switch s1. Host h_1 acted as the TCP server, and hosts h_2 through h_N acted as clients, producing the many-to-one communication pattern.

$$\{h_2, h_3, \ldots, h_N\} \rightarrow h_1 \tag{10}$$

Traffic was generated using iperf with the Transmission Control Protocol (TCP). The preserved 50 host execution trace shows the creation of 50 hosts, one switch, one iperf server on h_1, 49 clients, and an approximately 25s test interval. The many-to-one arrangement is reported as the traffic configuration actually used in the experiment. The study does not claim that this configuration reproduces a specific production data-center, Internet of Things, edge-computing, or telecommunications workload.

The exact iperf command line, the traffic rate per-client, the TCP connection rate, the link bandwidth, the propagation delay, and the queue configuration are not preserved in the surviving experimental documentation. No numerical values for these parameters are introduced retrospectively.

3.5 Host-Count Scenarios and Experimental Procedure

The experiment used the nine host-count scenarios defined in Eq. (1). Each value represents the total number of emulated hosts, including the server. Thus, the number of clients follows Eq. (2). No 200-host scenario is contained in the reported dataset. The surviving documentation does not provide sufficient evidence to reconstruct a scientific rationale for its omission; therefore, no retrospective explanation is introduced.

For each reported scenario, the documented workflow consisted of environment cleanup, POX start with the custom timeout module, Mininet topology creation, switch-controller connection, TCP traffic initialization, controller-event logging during an approximately 25-second interval, topology termination, and cleanup before the subsequent scenario.

An iperf server was started on h_1, while the remaining $N - 1$ hosts acted as clients. Controller activity was recorded in scenario-specific files identified in the original workflow as exp_Nhosts.txt. The documented logs were subsequently used for summary-data extraction. The research history preserves files for the reported scenarios, including 50, 100, 150, 250, 300, 350, 400, 450, and 500 hosts.

The cleanup was intended to reduce the residual Mininet state between scenarios. It is not interpreted as proof that the scenarios were completely independent or free from all effects on the shared-platform. The final reported dataset contains one observation per host-count scenario. Development or demonstration executions preserved elsewhere in the research history are not treated as repeated trials of the final dataset. Consequently, run-to-run statistical variability cannot be estimated.

3.6 Data Extraction and Evaluation Metrics

The documented analysis workflow processed scenario-specific controller logs in a summary dataset identified as scalability_results.csv. The reported and derived quantities are defined as follows. Hosts denote the total number of emulated hosts N, including server h_1. The Total_Flows denotes the cumulative number of flow-installation events recorded within a scenario. It is an event count, rather than the number of simultaneously resident flow entries. Therefore,

$$\text{Total_Flows} \neq \text{instantaneous switch occupancy} \tag{11}$$

Packet_In denotes the total number of Packet-In events recorded by the controller. Characterizes the controller-switch interaction, but does not independently measure controller

overload, processing latency, or service degradation. FlowRemoved denotes the total number of FlowRemoved events recorded during a scenario. Characterizes recorded flow-removal activity and is not interpreted as evidence of improved flow-table management.

Timeout-category counts and percentages represent the recorded installation decisions assigned to LOW, MEDIUM, and HIGH according to Eqs. (7) and (8). Peak concurrent flows are a controller-derived estimate based on the maximum value obtained from controller-maintained flow-count tracking. Because this quantity was not independently validated against direct switch-table measurements, it is not treated as a physical flow-table or TCAM occupancy.

The original analysis also introduced a derived quantity termed flow efficiency. To avoid implying a standardized performance metric, the revised manuscript defines it explicitly as flow-installation events per host:

$$E_N = \frac{\text{Total_Flows}_N}{N} \tag{12}$$

E_N is only a descriptive ratio. It does not represent throughput efficiency, energy efficiency, controller-resource efficiency, packet-delivery performance, or Quality of Service.

3.7 Study Variables and Interpretation Boundaries

Table 4 summarizes the experimental variable, fixed configuration parameters, recorded observations, and quantities that were not measured.

As summarized in Table 4, the total host count is the experimental variable intentionally varied between the reported scenarios, whereas the timeout policy configuration and documented traffic structure are treated as control conditions. LOW/MEDIUM/HIGH selections are internal policy outputs, while event counts and derived quantities are descriptive observations used to characterize the tested configuration.

In the original experiment, no quantitative acceptance criterion for scalability, stability, saturation, or optimal capacity was defined. Therefore, the completion of a scenario or the absence of a fatal controller crash is treated only as an operational observation and not as a proof of controller stability.

Likewise, the host-count scenario with the largest value of an individual metric is described only as having the highest observed value of that metric under the tested configuration. It is not interpreted as an overall optimum.

The methodological interpretation is additionally constrained by the shared virtual-machine environment, unavailable low-level configuration parameters identified in **??**, absence of comparative baselines and repeated trials, lack of sensitivity analysis, and absence of controller-resource, service-level, and independently validated switch-occupancy measurements.

Accordingly, the method supports empirical characterization of the recorded behavior of the tested configuration. It does not provide a basis for claims of comparative superiority, statistically demonstrated scalability improvement, controller stability, optimal capacity, controlled saturation, or graceful degradation.

Table 4: Experimental variables, observations, and interpretation limits

Methodological role	Variable / condition	Treatment in this study
Independent variable	Total host count N	Varied across nine reported scenarios
Control condition	Controller and policy implementation	POX 0.7.0 with the documented three-tier policy.
Control condition	Thresholds	100 and 500; fixed and heuristic.
Control condition	Timeout pairs	10/30, 7/20, and 5/15 s; fixed and heuristic.
Control condition	Topology	One controller, one switch, Nhosts.
Control condition	Traffic pattern	Many-to-one TCP using iperf.
Control condition	Approximate traffic interval	Preserved in documented workflow.
Computing platform	Approximate traffic interval	Approximately 25 s.
Internal policy output	LOW/MEDIUM/HIGH decision	Category selected for a recorded installation decision.
Internal policy output	Category percentage	Proportion of recorded installation decisions.
Dependent observation	Total_Flows	Cumulative recorded flow-installation events.
Dependent observation	Peak concurrent flows	Controller-counter-derived estimate.
Dependent observation	E_N	Flow-installation events per host.
Comparative hypothesis	None tested	No competing timeout policy was evaluated.
Comparative success criterion	Not defined	The study does not test superiority, improved scalability, or stability.
Not measured	CPU/RAM utilization	No controller-resource conclusion.
Not measured	Throughput	No throughput conclusion.
Not measured	Flow-setup/end-to-end delay	No latency conclusion.
Not measured	Packet loss	No packet-loss conclusion.
Not measured	Connection success	No service-reliability conclusion.
Not independently validated	Physical switch occupancy	flow_count is not treated as direct occupancy.

4 Results and Discussion

This section presents the recorded behavior of the flow-count-driven three-tier timeout policy across the nine host-count scenarios defined in section 3. The analysis focuses on cumulative flow-installation events, timeout-category decisions, Packet-In events, FlowRe-

moved events, a controller-derived estimate of peak concurrent flows, and flow-installation events per host.

The final reported dataset contains an observation for each host-count scenario. Accordingly, the results are interpreted descriptively and are not used to establish statistical significance or comparative superiority over timeout policies that were not evaluated in the original experiment. In addition, the controller-maintained flow_count and the controller-derived peak concurrent-flow estimate are not interpreted as independently validated measurements of physical switch-table occupancy.

4.1 Numerical Basis

To maintain consistency among the numerical results, figures, Abstract, and subsequent discussion, Table 5 provide the common numerical reference used throughout this section.

Table 5: Reported numerical observations across the reported host-count scenarios

Hosts	Cumulative flow-installation events	HIGH decisions (%)	MEDIUM decisions (%)	Packet-In events	FlowRemoved events	Installation events per host
50	1,692	65.7	28.4	3,280	100	33.84
100	3,591	86.1	11.1	5,584	200	35.91
150	4,159	88.0	9.6	5,970	200	27.73
250	6,626	92.5	6.0	9,241	500	26.50
300	4,290	88.3	9.3	6,347	500	14.30
350	7,931	93.7	5.0	10,599	600	22.66
400	3,899	87.2	10.3	5,634	600	9.75
450	4,542	89.0	8.8	6,633	700	10.09
500	1,958	74.0	20.9	3,970	600	3.92

The LOW category corresponds to the remaining proportion after HIGH and MEDIUM decisions. Because separate LOW percentages are not required for the main interpretation and are not reported individually in Table 5, additional LOW values are not reconstructed for analysis.

The values in Table 5 are used consistently in subsection 4.2 to subsection 4.6. No additional numerical values are introduced where the underlying numerical records are not available.

4.2 Cumulative Flow-Installation Activity

As summarized in Table 5 and illustrated in Figure 2, cumulative flow-installation activity varies non-monotonically across the nine tested host-count scenarios. $Total_Flows$ represents the cumulative number of recorded flow-installation events within each scenario and does not represent the number of entries simultaneously resident in the switch.

The cumulative count increases from 1,692 events at 50 hosts to 3,591 at 100 hosts, 4,159 at 150 hosts, and 6,626 at 250 hosts. It then decreases to 4,290 at 300 hosts before increasing to 7,931 at 350 hosts. The 350-host scenario therefore contains the highest observed

cumulative flow-installation count among the tested scenarios. The pattern remains non-monotonic at larger host counts. The count decreases to 3,899 at 400 hosts, increases to 4,542 at 450 hosts, and decreases again to 1,958 at 500 hosts. Therefore, the complete sequence cannot be described as a continuous increase followed by a continuous decline.

The 7,931 events observed at 350 hosts should be interpreted only as the largest cumulative installation-event count in the reported dataset. It does not establish physical flow-table capacity, maximum supported host count, or an optimal operating point. The experiment did not define a throughput, delay, connection-success, resource-utilization, or switch-occupancy criterion from which such an optimum could be determined.

The available observations also do not identify a single cause for the non-monotonic pattern. Flow installation and removal, controller processing, traffic generation, Mininet execution, and shared virtual-machine resources may all contribute to the observed values, but these factors were not measured independently. Therefore, the decline observed above 350 hosts cannot be attributed uniquely to the timeout policy.

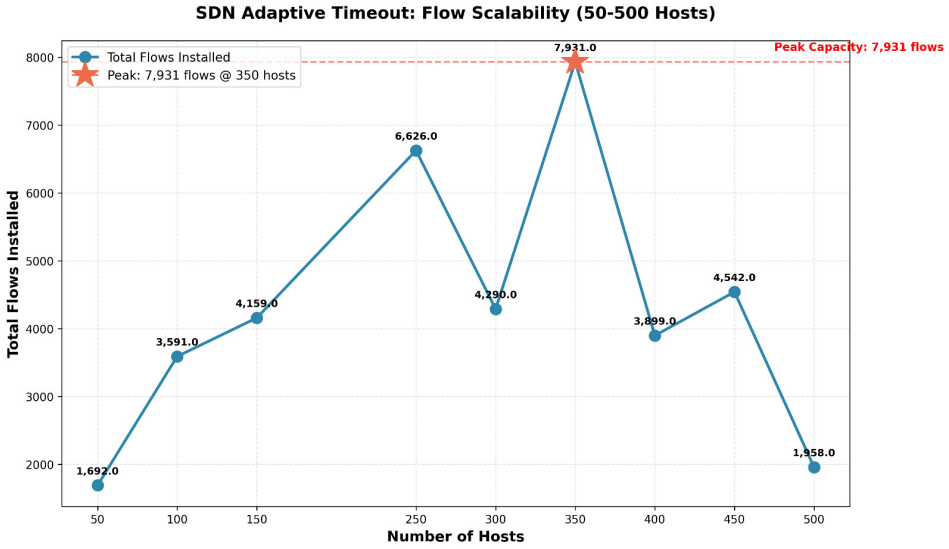

Figure 2: Cumulative recorded flow-installation events across the tested host-count scenarios.

4.3 Timeout-Decision Distribution

Figure 3 presents the distribution of recorded flow-installation decisions assigned to the LOW, MEDIUM, and HIGH timeout configurations defined in Table 3. The percentages follow Eq. (8).

HIGH is already the majority category at the smallest tested host count. At 50 hosts, HIGH represents 65.7% of recorded timeout decisions, whereas MEDIUM represents 28.4%. At 100 hosts, the HIGH proportion increases to 86.1%.

HIGH remains the majority category in every reported scenario: 88.0% at 150 hosts, 92.5% at 250 hosts, 88.3% at 300 hosts, 93.7% at 350 hosts, 87.2% at 400 hosts, 89.0% at 450

hosts, and 74.0% at 500 hosts. Its highest observed proportion is therefore 93.7% at 350 hosts.

MEDIUM follows a different pattern. It decreases from 28.4% at 50 hosts to 5.0% at 350 hosts, then increases to 10.3% at 400 hosts, decreases to 8.8% at 450 hosts, and increases to 20.9% at 500 hosts. Thus, neither HIGH nor MEDIUM varies monotonically with host count.

These values do not support a gradual interpretation in which HIGH becomes dominant only as the host count approaches 250–300 hosts. The retained figure does not support that earlier narrative. The distinction established in section 3 is therefore essential:

$$N \neq c$$

The total host count N defines the experimental scenario, while c is the controller-maintained flow-count indicator used for timeout selection. Consequently, a 50-host scenario can contain a majority of HIGH decisions when the internal indicator exceeds the HIGH threshold during a large proportion of recorded installation decisions. The percentages in Figure 3 represent proportions of recorded installation decisions, not percentages of experimental time spent in LOW, MEDIUM, or HIGH. Similarly, the dominance of HIGH does not independently demonstrate improved throughput, lower latency, better scalability, or controller stability.

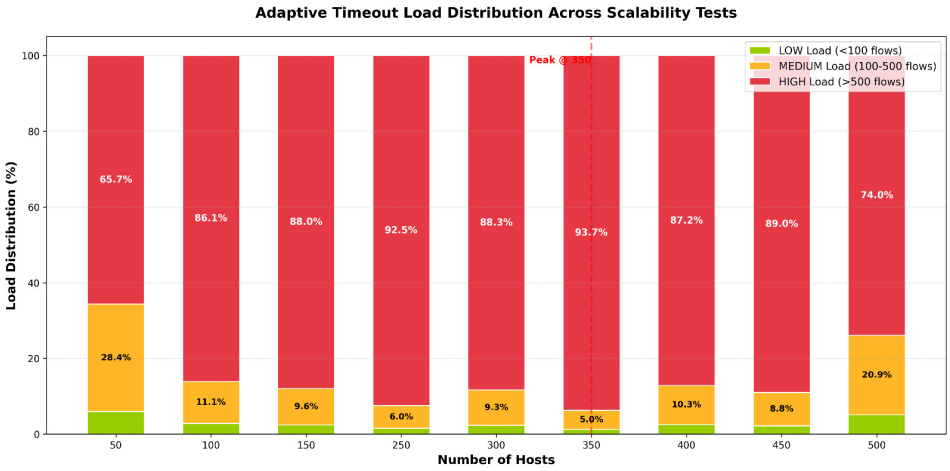

Figure 3: Distribution of recorded LOW, MEDIUM, and HIGH timeout decisions across the tested host-count scenarios.

4.4 Flow Removed and Packet-In Activity

Figure 4 presents the Packet-In and $FlowRemoved$ event counts summarized in Table 5. These two controller-level observations exhibit different patterns across the tested host-count scenarios.

Packet-In activity is non-monotonic. The recorded count increases from 3,280 events at 50 hosts to 5,584 at 100 hosts, 5,970 at 150 hosts, and 9,241 at 250 hosts. It then decreases to

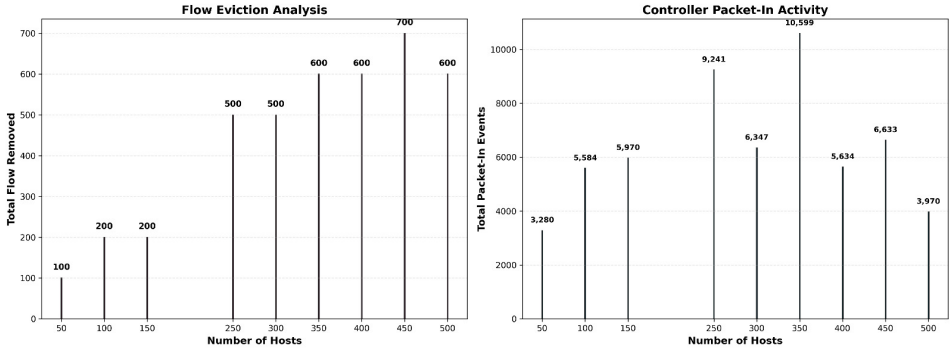

Figure 4: Recorded FlowRemoved and Packet-In events across the tested host-count scenarios.

6,347 at 300 hosts before reaching its highest reported value of 10,599 events at 350 hosts. At the larger host-count scenarios, the count decreases to 5,634 at 400 hosts, increases to 6,633 at 450 hosts, and decreases again to 3,970 at 500 hosts. Therefore, the reported Packet-In activity does not support a continuously increasing relationship with host count.

FlowRemoved activity follows a different pattern. The recorded count increases from 100 events at 50 hosts to 200 at both 100 and 150 hosts, 500 at both 250 and 300 hosts, and 600 at both 350 and 400 hosts. It reaches its highest observed value of 700 events at 450 hosts before decreasing to 600 at 500 hosts. Thus, the highest Packet-In and FlowRemoved counts occur at different host-count scenarios.

This difference indicates that the two event types should not be interpreted as having a simple one-to-one relationship. Packet-In events represent occasions in which packets require controller involvement within the reactive forwarding process, whereas FlowRemoved events represent removal notifications processed by the controller. Their aggregate counts describe controller-switch interaction and flow-removal activity, but do not independently measure controller overload, flow-setup delay, successful connection completion, throughput, or service degradation.

The timeout configuration may influence these event patterns because entries assigned shorter lifetimes can expire earlier and may subsequently require the rule reinstallation when related traffic continues. However, the present experiment does not isolate this effect from other factors such as traffic behavior, controller processing, Mininet execution, or shared virtual-machine resources. Therefore, the observed Packet-In and FlowRemoved counts are interpreted descriptively and are not used as evidence that the three-tier policy prevents flow-table overflow, improves scalability, or maintains controller stability.

4.5 Cross-Metric Descriptive Behavior

Figure 5 provides a combined descriptive view of cumulative flow-installation events, HIGH timeout-decision proportion, controller-derived peak concurrent flows, and flow-installation events per host. The combined observations show that different metrics reach their highest values in different scenarios. Cumulative flow-installation events reach their highest reported value at 350 hosts:

$$\text{Total_Flows}_{350} = 7,931 \tag{13}$$

The HIGH timeout-decision proportion also reaches its highest value at 350 hosts:

$$P_{\text{HIGH},350} = 93{,}7\% \tag{14}$$

Packet-In activity, shown separately in Figure 4, likewise reaches its highest reported value at 350 hosts:

$$\text{PacketIn}_{350} = 10,599 \tag{15}$$

In contrast, FlowRemoved reaches its highest reported value at 450 hosts.

$$\text{FlowRemoved}_{450} = 700 \tag{16}$$

The flow-installation-events-per-host ratio defined in Eq. (12) provides another distinct pattern. Its highest reported value occurs at 100 hosts:

$$E_{100} = 35{,}91 \tag{17}$$

The ratio is 33.84 at 50 hosts, 27.73 at 150 hosts, 26.50 at 250 hosts, 14.30 at 300 hosts, 22.66 at 350 hosts, 9.75 at 400 hosts, 10.09 at 450 hosts and 3.92 at 500 hosts. This quantity is therefore also non-monotonic.

The original analysis referred to E_N as flow efficiency. In the revised paper, it is interpreted only as flow-installation events per host. A higher value does not independently indicate better throughput, lower delay, greater resource efficiency, or improved quality of Service.

The controller-derived peak concurrent-flow curve in Figure 5 also exhibits a non-monotonic pattern and visually reaches its largest plotted value in the 350-host scenario. However, the exact numerical series for this metric is not reported in the retained numerical summary used for Table 5. Therefore, exact values are not visually estimated from the graph or retrospectively reconstructed.

Furthermore, this metric is derived from the controller-maintained flow-count tracking and was not independently validated against instantaneous switch-table occupancy. It is therefore not treated as physical TCAM capacity or exact switch occupancy.

Taken together, the cross-metric results do not identify one overall optimum. The coincidence of several highest observed metrics at 350 hosts is an empirical observation of the tested dataset, not evidence of an optimal capacity or a validated saturation boundary.

4.6 Interpretation Relative to the Research Question

The results answer the descriptive research question established in section 1 by showing how timeout-category decisions and controller-level event observations vary across the tested host-count scenarios. First, the timeout-category distribution does not show a gradual LOW-to-MEDIUM-to-HIGH transition as the host count increases. HIGH is already the majority category with 50 hosts and remains dominant in all nine scenarios. This confirms that the host count and the internal flow-count indicator must not be interpreted as interchangeable.



Figure 5: Cross-metric descriptive summary of cumulative flow-installation events, HIGH-decision proportion, controller-derived peak concurrent flows, and flow-installation events per host.

Second, the controller-level observations vary non-monotonically. Cumulative flow-installation events and Packet-In activity reach their highest reported values at 350 hosts, FlowRemoved reaches its maximum at 450 hosts, and flow-installation events per host reach their maximum at 100 hosts.

Third, reductions in several metrics at larger host counts cannot be interpreted as graceful degradation, controlled saturation, or successful overload handling. The available measurements cannot distinguish whether these reductions are predominantly related to timeout-policy behavior, controller processing, unsuccessful or altered traffic activity, Mininet execution, virtualization constraints, or interactions among these factors.

Therefore, the strongest conclusion supported by the available evidence is narrower: the implemented three-tier policy produces observable categorical timeout decisions and controller-level event patterns under the tested POX/Mininet configuration, and those observations vary non-monotonically across the nine reported host-count scenarios.

This characterization does not establish comparative performance improvement because no alternative timeout policy was evaluated under identical conditions. The observed behavior can also be qualitatively positioned in relation to the timeout mechanisms discussed in section 2. AH-IHTA uses traffic characteristics and utilization of the flow-table, whereas HQTimer and DDT incorporate learning-based information into the selection of the timeout. In contrast, the present policy selects one of three predefined timeout pairs using only the controller-maintained flow-count indicator. Because these studies differ in controller platform, topology, traffic generation, parameterization, and evalua-

tion metrics, their reported performance values are not compared quantitatively with the present results. In the tested policy, HIGH decisions correspond to the shortest configured timeout pair of 5 s/15 s and constitute the majority of recorded installation decisions in every scenario. Shorter configured lifetimes can create more opportunities for entry expiration and subsequent rule reinstallation when related traffic continues; however, the present data do not isolate this mechanism from other causes of Packet-In and FlowRemoved variation. Therefore, recorded event counts cannot determine whether the shorter HIGH timeout improves or degrades service-level performance.

4.7 Limitations of the Present Results

The interpretation of the results is constrained by the experimental design established in section 3. No fixed-timeout, default-POX, two-tier, or alternative adaptive timeout policy was evaluated under identical conditions. Therefore, the results cannot quantify comparative superiority.

The final reported dataset contains one observation per host-count scenario, so run-to-run variability, confidence intervals, and statistical significance cannot be estimated. Controller CPU and memory utilization, flow-setup latency, throughput, end-to-end delay, packet loss, connection success, and direct switch-table occupancy were not measured. Consequently, the available metrics cannot establish service-level performance or controller stability.

The POX controller, Mininet emulator, virtual switch, emulated hosts, and traffic-generation processes shared the same virtual-machine environment. The effects caused by the timeout policy cannot therefore be quantitatively separated from possible controller, emulator, or virtualization constraints.

Finally, thresholds of 100 and 500 and timeout pairs of 10/30 s, 7/20 s, and 5/15 s were selected heuristically and were not subjected to sensitivity analysis. The results therefore characterize this specific parameter configuration rather than establishing generally optimal timeout settings.

These limitations do not invalidate the recorded observations. These factors establish the parameters that serve as a reference for interpreting these observations. This distinction directly addresses the need to ensure that internal policy metrics are not treated as independent evidence of scalability, stability, saturation, or optimality.

5 Conclusion

This study implemented and empirically characterized a three-tier rule-based timeout policy for flow-table management in a POX/Mininet Software-Defined Networking environment. The policy uses a controller-maintained flow_count indicator to select one of three predefined timeout configurations: LOW, MEDIUM, or HIGH. The thresholds and timeout pairs were manually selected in the original implementation and are therefore treated as heuristic configuration parameters rather than optimized or generally applicable values.

The experiment considered nine many-to-one TCP scenarios containing 50, 100, 150, 250, 300, 350, 400, 450, and 500 total hosts. The reported results show that the policy behavior is non-monotonic across these scenarios. HIGH was already the majority timeout category in 50 hosts, accounting for 65.7% of recorded installation decisions, and remained

the majority in every tested scenario, reaching its highest observed proportion of 93.7% in 350 hosts. The cumulative flow-installation activity also reached its highest reported value at 350 hosts with 7,931 events, while Packet-In reached 10,599 events at the same host count. In contrast, FlowRemoved reached its highest reported value of 700 events at 450 hosts, and the flow-installation-events-per-host ratio reached its highest value of 35.91 at 100 hosts. These differing maxima reinforce that the reported metrics do not identify a single overall optimum.

The results therefore answer the descriptive research question by showing that the three-tier policy produces measurable timeout-category decisions and controller-level flow-management event patterns as the tested host count changes. However, the observations do not establish physical flow-table capacity, an optimal operating point, improved scalability, controller stability, flow-table overflow prevention, controlled saturation, or graceful degradation. The controller-maintained flow_count and the derived peak concurrent-flow measure were not independently validated against instantaneous switch-table occupancy, and the available event counts do not directly measure service-level network performance.

The interpretation is further limited by the original experimental design. No fixed-timeout, default-POX, two-tier, or alternative adaptive policy was evaluated under identical conditions, and the final reported dataset contains one observation per host-count scenario. Controller CPU and memory utilization, flow-setup latency, throughput, end-to-end delay, packet loss, connection success, and direct switch-table occupancy were not collected. In addition, the controller, Mininet emulator, virtual switch, hosts, and traffic generation operated within the same virtual-machine environment. Consequently, the present contribution should be understood as an empirical characterization of one specific flow-count-driven timeout configuration rather than a comparative demonstration of performance superiority.

Future work should evaluate the policy against fixed-timeout and alternative adaptive mechanisms under identical experimental conditions, use repeated trials to quantify run-to-run variability, and include controller-resource and service-level measurements. Sensitivity analysis of the thresholds and timeout pairs is also required to determine how strongly the observed behavior depends on the selected configuration. Evaluation on multi-switch topologies and, where feasible, physical OpenFlow-capable switches would further clarify the extent to which the observed behavior generalizes beyond the present POX/Mininet environment.

Acknowledgments

Mukhamad Salman Nurdin contributed to conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing, visualization, and revision. Galura Muhammad Suranegara contributed to methodology, validation, formal analysis, investigation, supervision, writing, and project administration. Both authors read and agreed to the published version of the manuscript.

Declaration

Conflict of Interest

The authors declare no competing financial interests or personal relationships that could have influenced this work.

Generative AI

Generative AI tools assisted brainstorming and writing style only. All technical work was done independently by the authors, who reviewed the output and take full responsibility.

Authors Contributions

Mukhamad Salman Nurdin: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing, Visualization, and revision in response to reviewers. Galura Muhammad Suranegara (Corresponding Author): Methodology, Validation, Formal analysis, Investigation, Supervision, Writing, Project administration, and scholarly guidance. Both authors read and approved the final manuscript.

References

[1] B. B. Deneke, A. M. Beyene, and E. A. Haile, "Improving software defined network controllers in a multi-vendor environment," *Heliyon*, vol. 10, no. 4, 2024.

[2] V. Tyagi and S. Singh, "A partial load distribution aware cluster migration approach for stable control plane in software-defined wireless sensor networks," *Applied Soft Computing*, p. 114023, 2025.

[3] J. Zhang, X. Huang, J. Li, K. Xue, Q. Sun, and J. Lu, "A dynamic flow table management method based on real-time traffic monitoring," in *2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR)*, pp. 212–217, IEEE, 2022.

[4] L. Djennadi, G. Diaz, K. Boussetta, and C. Cerin, "Exploring sdn architectures for iot over low-power and lossy networks (llns): A survey," *Computer Networks*, p. 111494, 2025.

[5] G. Huang and H. Y. Youn, "Proactive eviction of flow entry for sdn based on hidden markov model," *Frontiers of Computer Science*, vol. 14, no. 4, p. 144502, 2020.

[6] D. Tang, C. Gao, W. Liang, J. Zhang, and K. Li, "Ftmaster: A detection and mitigation system of low-rate flow table overflow attacks via sdn," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 5073–5084, 2023.

[7] Q. Li, 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.

[8] S. Bera, S. Misra, and N. Saha, "Traffic-aware dynamic controller assignment in sdn," *IEEE Transactions on Communications*, vol. 68, no. 7, pp. 4375–4382, 2020.

[9] Y. Zeng, B. Ye, B. Tang, S. Lu, F. Xu, S. Guo, and Z. Qu, "Mobility-aware proactive flow setup in software-defined mobile edge networks," *IEEE Transactions on Communications*, vol. 71, no. 3, pp. 1549–1563, 2023.

[10] S. Bera, S. Misra, and N. Saha, "Traffic-aware dynamic controller assignment in sdn," *IEEE Transactions on Communications*, vol. 68, no. 7, pp. 4375–4382, 2020.

[11] M. Hamdan, B. Mohammed, U. Humayun, A. Abdelaziz, S. Khan, M. A. Ali, M. Imran, and M. N. Marsono, "Flow-aware elephant flow detection for software-defined networks," *IEEE Access*, vol. 8, pp. 72585–72597, 2020.

[12] G. Huang and H. Y. Youn, "Proactive eviction of flow entry for sdn based on hidden markov model," *Frontiers of Computer Science*, vol. 14, no. 4, p. 144502, 2020.

[13] H. Yang, G. F. Riley, and D. M. Blough, "Stereos: Smart table entry eviction for openflow switches," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 377–388, 2019.

[14] W. Li, C. Zeng, J. Hu, and K. Chen, "Flowsail: Fine-grained and practical flow control for datacenter networks," *IEEE/ACM Transactions on Networking*, vol. 32, no. 5, pp. 3916–3928, 2024.

[15] N. Kröger, H. Harkous, F. Mehmeti, and W. Kellerer, "Performance modeling and analysis of p4 programmable devices with general service times," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 4543–4562, 2024.

[16] Y. Liu, B. Tang, D. Yuan, J. Ran, and H. Hu, "A dynamic adaptive timeout approach for sdn switch," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 2577–2582, IEEE, 2016.

[17] Z. Li, Y. Hu, T. Hu, and P. Wei, "Dynamic sdn controller association mechanism based on flow characteristics," *IEEE Access*, vol. 7, pp. 92661–92671, 2019.

[18] Y.-Z. Cai, Y.-T. Wang, and M.-H. Tsai, "Dynamic adjustment for proactive flow installation mechanism in sdn-based iot," *Computer Networks*, vol. 194, p. 108167, 2021.

[19] Y.-Z. Cai, Y.-T. Wang, and M.-H. Tsai, "Dynamic adjustment for proactive flow installation mechanism in sdn-based iot," *Computer Networks*, vol. 194, p. 108167, 2021.

[20] N. Harris Jr and S. Khorsandroo, "Ddt: a reinforcement learning approach to dynamic flow timeout assignment in software defined networks," *Journal of Network and Systems Management*, vol. 32, no. 2, p. 35, 2024.

[21] S. S. G. Shiny and K. Murugan, "Tsdn-wise: Automatic threshold-based low control-flow communication protocol for sdwsn," *IEEE Sensors Journal*, vol. 21, no. 17, pp. 19560–19569, 2021.

[22] Y. Li, N. Ren, X. Wang, Y. Chen, X. Zhang, L. Guo, and G. Xie, "Tuplechain: Fast flow table lookup with efficient on-line updates and scalability," *IEEE Transactions on Computers*, vol. 75, no. 2, pp. 527–540, 2025.

[23] B. Yan, Y. Xu, and H. J. Chao, "Adaptive wildcard rule cache management for software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 962–975, 2018.

[24] S. H. Rastegar, A. Abbasfar, and V. Shah-Mansouri, "Rule caching in sdn-enabled base stations supporting massive iot devices with bursty traffic," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8917–8931, 2020.

[25] R. Li, Y. Pang, J. Zhao, and X. Wang, "A tale of two (flow) tables: Demystifying rule caching in openflow switches," in *Proceedings of the 48th International Conference on Parallel Processing*, pp. 1–10, 2019.

[26] D. Tang, Z. Zheng, C. Yin, B. Xiong, Z. Qin, and Q. Yang, "Ftodefender: An efficient flow table overflow attacks defending system in sdn," *Expert Systems with Applications*, vol. 237, p. 121460, 2024.

[27] D. Tang, Z. Zheng, K. Li, C. Yin, W. Liang, and J. Zhang, "Ftop: An efficient flow table overflow preventing system for switches in sdn," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 2524–2536, 2023.

[28] L. M. A. Fariño, J. F. A. Pizarro, M. J. Infante, A. R. T. Reyes, and B. M. M. Morán, "Sdn redes definidas por software usando mininet," *Revista científica y tecnológica UPSE*, vol. 9, no. 1, pp. 48–56, 2022.