

Traffic-adaptive Field Ordering for Improving Tree-rule Firewall Performance

Thawatchai Chomsiri ¹ and Suwichai Phunsa ^{2,*}

¹ Department of Information Technology, Research Center of Information Technology for the Future,
Faculty of Informatics, Mahasarakham University, Mahasarakham, Thailand

² Department of Creative Media, Digital Contents for Development Research Unit, Faculty of Informatics,
Mahasarakham University, Mahasarakham, Thailand

Email: thawatchai.c@msu.ac.th (T.C.); suwichai.p@msu.ac.th (S.P.)

*Corresponding author

Abstract—Firewall performance depends heavily on the order in which packet fields are examined during rule matching. The Tree-Rule Firewall (TRF) organizes rules as a hierarchical tree whose levels correspond to packet fields, but existing designs fix the field-to-level assignment statically at compile time, which is suboptimal under non-stationary traffic. This paper proposes Automatic Field-Level Positioning (AFLP), a lightweight mechanism that monitors per-field elimination rates online and repositions field levels so that the most discriminating field always occupies the root. Three trigger strategies are evaluated: Periodic, Threshold-based Exponential Moving Average (EMA), and On-demand. Under a 90% skewed two-phase traffic model (10–200 rules, up to 100,000 packets, 30 repetitions per scenario), AFLP-Periodic reduces average comparisons per packet by up to 68.77% and total processing time by up to 51.60% relative to a static TRF, while AFLP-On-demand achieves 44.77% and 29.19% with a single repositioning event. Analysis of the Threshold-based strategy reveals a sensitivity trade-off between thrashing and under-triggering; an extended 378-configuration sweep locates a narrow workable regime at percent-level thresholds ($\delta \leq 0.02$), motivating future work on adaptive trigger design. These results establish AFLP as a fully online mechanism that discovers a near-optimal field order—with an average optimality gap of 6.57% (maximum 8.72%) relative to the brute-force optimum—without offline search or prior traffic knowledge. This study is a proof of concept based entirely on software simulation with synthetic rule sets and traffic; the reported gains cannot be assumed to transfer directly to production environments without validation on real traffic traces.

Keywords—tree-rule firewall, field-level positioning, traffic frequency, elimination rate, firewall optimization, network security, processing speed, Exponential Moving Average (EMA) threshold

I. INTRODUCTION

Network firewalls are foundational components of modern network security, enforcing access-control

policies by inspecting packet header fields against a predefined rule set and determining whether each packet should be admitted or discarded [1]. The Listed-Rule Firewall (LRF), which evaluates rules sequentially using a first-match policy, has historically dominated practical deployments due to its conceptual simplicity and ease of implementation [2]. However, LRF is known to exhibit several well-characterized structural deficiencies that become increasingly severe as networks scale. Chomsiri *et al.* [3] identified five fundamental limitations of the listed-rule representation: shadowed rules, redundant rules, conflicting rules, $O(n)$ per-packet matching complexity, and difficulty of management. These limitations were independently corroborated through relational-algebra analysis [4, 5], establishing a comprehensive theoretical critique of the listed-rule model.

To address these structural deficiencies, Chomsiri *et al.* [3] proposed the Tree-Rule Firewall (TRF), in which firewall rules are organized as a rooted tree. Each level of the tree corresponds to one packet field—Source IP at Level 1, Destination IP at Level 2, Protocol at Level 3, and Destination Port at Level 4 in the standard configuration—and packet matching proceeds by traversing the tree from root to leaf, examining one packet field per level. In its original sorted-range design, this reduces matching complexity from $O(n)$ to $O(d \cdot \log n)$, where d is the number of fields (tree depth), while simultaneously eliminating all three types of structural anomaly by construction. Subsequent works extended TRF to cloud computing environments [6], large enterprise networks [7], and high-speed data transmission through a hybrid stateful architecture [8, 9].

Despite these advances, a fundamental assumption present in all prior TRF designs remains unexamined: the assignment of packet fields to tree levels—referred to in this paper as the field order—is determined at compile time and does not change during operation. This assumption is problematic in practice because network traffic is non-stationary. The field that provides the greatest discriminatory power at one point in time—that is, the field that most effectively partitions the packet space at its assigned level—may differ substantially from

the optimal field at another. During a period of concentrated web-server access, for example, Destination IP may resolve the vast majority of packets at the root level with a single comparison. During a volumetric attack from a single source, however, Source IP becomes the dominant discriminator. The default field order specified in the standard TRF configuration may perform far from optimal for the actual traffic and rule-set structure encountered in deployment, and identifying a good order manually requires either exhaustive offline evaluation of all $d!$ orderings or representative traffic traces collected in advance.

This paper proposes Automatic Field-Level Positioning (AFLP), a mechanism that continuously monitors live traffic statistics and automatically adjusts the field-level assignment of the tree-rule firewall in response to detected changes in traffic patterns. AFLP introduces the elimination rate as a theoretically motivated frequency metric: for each packet field F assigned to level k , the elimination rate measures the fraction of packets for which field F uniquely resolved the packet's onward matching path at level k , indicating that F was the sole discriminating field at that point in the tree. Fields with high elimination rates are the most effective discriminators and should be assigned to higher tree levels to minimize the expected number of comparisons per packet. When AFLP detects a shift in the elimination-rate distribution, it invokes a repositioning procedure that sorts fields by their current elimination rates and rebuilds the tree with the new field order.

Three trigger strategies are designed to accommodate different network environments and operational requirements. The Periodic strategy repositions at fixed packet-count intervals, providing simple and predictable overhead. The Threshold-based strategy uses an Exponential Moving Average (EMA) of the elimination rate to smooth packet-level noise before applying a change-detection threshold, addressing the thrashing problem observed in prior formulations. The On-demand strategy allows administrators to initiate repositioning explicitly at known traffic-change events.

In brief, TRF matches a packet by walking a single root-to-leaf path: each tree level examines one packet field, and child branches that cannot match are eliminated from further consideration. How much of the candidate set each field eliminates—its elimination rate—varies with the traffic mix: a field that discriminates strongly under one workload may become uninformative when the dominant traffic shifts. Because the field-to-level assignment is fixed at build time in existing designs, no single assignment can remain best as conditions change. Section III formalizes this observation and presents AFLP, which measures elimination rates online and repositions fields so that the strongest discriminator occupies the root.

The contributions of this paper are as follows:

- (1) We formally identify and characterize the performance limitation of static field-level ordering in the Tree-Rule Firewall under non-stationary

traffic, providing a theoretical basis for frequency-driven field repositioning.

- (2) We propose the AFLP algorithm with the elimination-rate metric, which automatically repositions field levels based on real-time traffic statistics without requiring machine learning, rule modification, or hardware changes.
- (3) We design three trigger strategies—Periodic, Threshold-based (EMA), and On-demand—with formal complexity analysis, and identify the fundamental sensitivity trade-off of the Threshold-based strategy between thrashing and under-triggering.
- (4) We evaluate AFLP through comprehensive simulation experiments under a controlled two-phase skewed traffic model, reporting both average comparisons per packet and wall-clock processing time with 30 independent repetitions per scenario, and providing per-rule-set analysis across four rule-set sizes and two traffic volumes.

The remainder of this paper is organized as follows. Section II reviews related work. Section III presents the proposed AFLP mechanism, together with the TRF preliminaries on which it builds and the three trigger strategies. Section IV describes the simulation design. Section V reports the experimental results; Section VI discusses their interpretation, limitations, and practical implications; and Section VII concludes and outlines future work.

II. RELATED WORK

A. Listed-rule Firewall: Limitations and Analysis

The conventional packet-filtering firewall operates by comparing each incoming packet against an ordered sequence of rules using a first-match policy, resulting in $O(n)$ worst-case time complexity per packet [2]. This linear cost has motivated substantial research on rule-set optimization and anomaly detection. Wool [2] conducted a large-scale empirical study of firewall configuration errors, finding that anomalous rules—including redundant, shadowed, and conflicting rules—were prevalent in real enterprise deployments. Complementary tools for firewall analysis include the Lumeta firewall analyzer [10], the firewall policy advisor of Al-Shaer and Hamed [11], and the FIREMAN toolkit [12]. Gouda and Liu [13] addressed firewall design correctness through a formal framework covering consistency, completeness, and compactness. Guttman [14] examined local enforcement of global firewall policies in distributed settings, and Liu and Gouda [15] proposed diverse firewall design to improve fault tolerance. Al-Shaer and Hamed [16] proposed a formal taxonomy of firewall policy anomalies and developed automated tools for their detection and resolution, establishing a widely cited framework for correctness analysis.

Chomsiri and Pornavalai [4] approached LRF analysis through relational algebra, representing each rule as a tuple and using set-theoretic operations to enumerate all pairwise rule relationships. Pornavalai and Chomsiri [5] extended this analysis to distributed firewall

configurations spanning multiple enforcement points. These two works established the formal theoretical foundation that motivated the design of TRF as a structurally anomaly-free alternative to LRF.

B. Tree-rule Firewall

Chomsiri *et al.* [3] proposed TRF as a structural solution to LRF limitations. In TRF, each packet field occupies one level of a tree, and packet matching traverses the tree in $O(d)$ comparisons. The tree structure inherently prevents shadowing, redundancy, and conflict, since each path from root to leaf is unique and non-overlapping. He *et al.* [6] extended TRF to cloud computing, demonstrating its effectiveness for policy enforcement at virtual machine boundaries and its throughput advantage over iptables-based LRF. Chomsiri *et al.* [7] scaled TRF to large enterprise networks by introducing structural optimizations that reduce tree size and lookup latency for large rule sets. The Hybrid Tree-Rule Firewall [8] incorporated stateful connection tracking via hash functions, allowing established-flow packets to bypass tree matching entirely, and achieved significant throughput improvements over both LRF and standard TRF. The doctoral thesis [9] provides the comprehensive theoretical framework, including formal anomaly-freedom proofs and complexity analysis, underlying all TRF variants.

C. Traffic-frequency-based Optimization

Frequency-driven optimization of firewall rule sets has been studied primarily within the LRF model. Acharya *et al.* [17] proposed organizing rules by match probability to reduce average matching latency. Bellovin and Cheswick [18] discussed traffic-aware rule ordering as an operational heuristic. Ben-Yehuda *et al.* [19] developed algorithms for LRF rule reordering based on empirically observed match probabilities. Maci and Pirlo [20] proposed a two-stage rule-reordering algorithm that respects dependency constraints between rules, enabling safe reordering even in the presence of rule interactions. Hamed and Al-Shaer [21] proposed dynamic rule-ordering optimization for high-speed firewall filtering. Building on this, Hamed, El-Atawy, and Al-Shaer [22] proposed a dynamic optimization technique that uses statistical search trees and early rejection of unwanted flows to exploit traffic characteristics and minimize the average packet-matching time. Most directly related to our work, Trabelsi, Zhang, and Zeidan [23] proposed Dynamic Rule and Rule-Field Optimization (DR-RFO), which reorders both rules (for early acceptance) and rule-fields (for early rejection) within the LRF model based on packet-matching histograms. Critically, however, DR-RFO operates on the listed-rule structure, where field-level reordering refers to permuting condition columns in a flat table, not to restructuring a hierarchical tree. The structural semantics and algorithmic consequences of field-level repositioning in TRF—where a change in field order requires rebuilding the entire tree—are fundamentally different and have not been studied.

D. Packet Classification Algorithms

Decision-tree classifiers such as HiCuts [24], HyperCuts [25], and hierarchical intelligent cuttings [26] use rule-distribution statistics to shape multi-dimensional lookup trees, while range-matching structures for layer-four switching [27], high-speed policy-based forwarding [28], and Ternary Content-Addressable Memory (TCAM) hardware [29] target line-rate lookup. In Software-Defined Networking (SDN) settings, OpenFlow-based virtual firewalls [30], policy-verification frameworks [31], and flow-driven TCAM caching [32] manage rules at the network level. All of these optimize worst-case lookup over a static rule set; AFLP instead targets average-case matching cost under a non-stationary traffic distribution.

E. Change Detection and Adaptive Thresholds

AFLP's Threshold-based trigger draws on statistical changepoint detection: Basseville and Nikiforov [33] give the foundational treatment of abrupt-change detection, and Adams and MacKay [34] introduced Bayesian online changepoint detection for streaming data. Smoothing time-series statistics with exponential moving averages before applying threshold tests is standard practice in network measurement; our variant applies this principle to the elimination-rate series. The resulting trade-off between thrashing and under-triggering is analyzed in Sections V.C and VI.B.

F. Machine Learning for Network Security

Beyond software classifiers, P4-programmable pipelines [35] expose match-table structure to reconfiguration in hardware. Machine learning has been applied across this space: multi-class classification of firewall logs [36], decision-tree log analysis for performance improvement [37], learned-index packet classification [38], Artificial Neural Network (ANN)-based packet classification with synthetic minority over-sampling [39], and machine-learning-driven attack detection more broadly [40], alongside switch-level hardening practice against Man-in-the-Middle (MITM) attacks [41]. AFLP deliberately avoids learning models: simple frequency counters and one sorting step add $O(1)$ per-packet overhead, remain fully interpretable, require no training data, and yield provably correct TRF trees at every repositioning—properties suited to security-critical deployment.

G. Research Gap

Table I summarizes the key distinctions between AFLP and representative prior works. To the best of our knowledge, no prior work has addressed the problem of dynamic field-level positioning in a tree-structured firewall. AFLP is the first mechanism to exploit real-time field-frequency statistics to automatically optimize the structural field-order assignment within the TRF framework, without modifying the rules, requiring machine learning, or using additional hardware.

TABLE I. COMPARISON OF AFLP WITH REPRESENTATIVE PRIOR WORKS

Approach	Architecture	Freq.-driven	Dynamic update	No ML
Chomsiri <i>et al.</i> [3]	TRF (static)	No	No	Yes
He <i>et al.</i> [6]	TRF (cloud)	No	No	Yes
Chomsiri <i>et al.</i> [7, 8]	TRF (hybrid)	No	No	Yes
Acharya <i>et al.</i> [17]	LRF	Yes (rule)	Partial	Yes
Trabelsi <i>et al.</i> [23]	LRF (DR-RFO)	Yes (rule+field)	Yes	Yes
HiCuts/HyperCuts [24, 25]	Decision tree	Yes (static)	No	Yes
Rashelbach <i>et al.</i> [38]	Learned index	Yes (ML)	Yes	No
AFLP (this work)	TRF (dynamic)	Yes (field-level)	Yes	Yes

III. PROPOSED METHOD: AFLP

A. Preliminaries and Problem Formalization

1) Structure and notation

A Tree-Rule Firewall T is a rooted directed tree in which each internal node at depth $k \in \{1, 2, 3, 4\}$ is labeled with a value of the k -th packet field F_k , and each leaf node is labeled with a firewall action $a \in \{\text{Accept}, \text{Deny}\}$. A root-to-leaf path $\pi = \langle v_1, v_2, v_3, v_4, a \rangle$ encodes one firewall rule, where v_i is the field value matched at level i . The wildcard symbol (*) at any node matches any field value.

In the standard TRF formulation [3], four packet fields are assigned to four tree levels as shown in Table II. The tree is constructed by inserting rules one by one: for each rule, the engine traverses the tree from the root, creating new internal nodes as needed, and attaches the action label to the leaf. This construction guarantees that every root-to-leaf path is structurally unique, eliminating shadowing and redundancy by construction.

TABLE II. STANDARD FIELD-TO-LEVEL ASSIGNMENT IN TRF [3]

Level	Packet Field
1 (Root)	Source IP
2	Destination IP
3	Protocol
4	Destination Port

2) Packet matching and complexity

To match a packet $P = (\text{src}, \text{dst}, \text{proto}, \text{port})$ against T , the matching engine begins at the root and, at each level k , iterates over the child nodes of the current node. For each child labeled with value v , it tests whether v matches the corresponding field of P (exact match or wildcard). The first complete root-to-leaf path on which all d comparisons succeed determines the action. If no such path exists, the default-deny rule applies. In the present implementation, each node's children are stored in an unordered associative container keyed by field value, and matching scans all children linearly; because the wildcard (default-deny) branch is always a co-matching candidate alongside any specific-value match, more than one child may match at a given level, requiring exploration of multiple candidate sub-paths. The number of comparisons at level k is therefore bounded by the number of distinct values of field F_k across the rule set—which at the root approaches the rule-set size n —and total per-packet matching cost is $\Theta(n)$ rather than $O(d)$. This differs from

the original Tree-Rule Firewall design [3], in which node children are sorted into non-overlapping ranges, enabling $O(\log k)$ lookup per level without exploring multiple branches—i.e., $O(d \log n)$ total matching cost, versus $\Theta(n)$ for the linear-scan engine simulated here; adopting that structure in the simulation is identified as future work and would be expected to both reduce matching cost and sharpen the elimination-rate signal used by AFLP (Section III.C). We emphasize that AFLP does not change these asymptotic bounds: matching remains $\Theta(n)$ in the simulated engine and tree rebuilding remains $O(n)$ (Section III.F). What AFLP optimizes is the constant factor—the expected number of comparisons per packet—by selecting the field order that maximizes early elimination; every improvement reported in Section V is a reduction of this constant factor, not of the asymptotic growth rate.

3) Anomaly freedom

A fundamental property of TRF is that, by construction, it admits no shadowed, redundant, or conflicting rules. Shadowing is impossible because every path is structurally unique. Redundancy is impossible because inserting a semantically equivalent rule would overwrite the existing leaf rather than creating a new path. Conflict is impossible because every packet P follows a unique path and receives exactly one action. These properties were formally proved in [3] and extended to the stateful and hybrid cases in [7–9].

4) Static-order limitation and the elimination rate

All prior TRF designs fix the field order at compile time. The per-packet matching cost depends critically on this order: a field placed at Level 1 that effectively partitions the packet space reduces the number of nodes visited in all subsequent levels. This observation motivates the following definition.

Definition 1 (Elimination Rate). For a field F assigned to level k in TRF tree T , given a stream of N packets, the elimination rate of F is defined as:

$$\text{elim_rate}(F) = \text{elim}(F) / \max(\text{visit}(F), 1)$$

where $\text{visit}(F)$ is the number of packets that reached level k during matching, and $\text{elim}(F)$ is the number of those packets for which field F had exactly one matching child node at level k —meaning the field uniquely resolved the packet's onward path at that level. A field with $\text{elim_rate}(F) \approx 1$ resolves nearly every packet unambiguously and is a strong candidate for promotion

toward Level 1 (Root), subject to the branching structure of the rule set. Note that $elim_rate$ is a conditional statistic: it is estimated only from packets that reach the field's current level, so values measured at different levels are not directly comparable samples of a position-independent quantity (see Section V.I). A field with $elim_rate(F) \approx 0$ provides little discriminatory power at its current level and should be demoted.

Observation 1. Because network traffic is non-stationary, $elim_rate(F)$ for a given field F may change substantially over time. A static field order optimized for traffic conditions at time t_1 may be suboptimal at t_2 , although the magnitude of this effect depends on the rule-set structure; when the rule set itself is stable, a single well-chosen order may remain effective across traffic regimes (see Section V.H). AFLP detects and responds to such changes automatically. Fig. 1 illustrates this shift in elimination rate across the two traffic phases, providing a visual demonstration of why the appropriate field order depends on the prevailing traffic and rule-set structure—the dependence that AFLP discovers automatically.

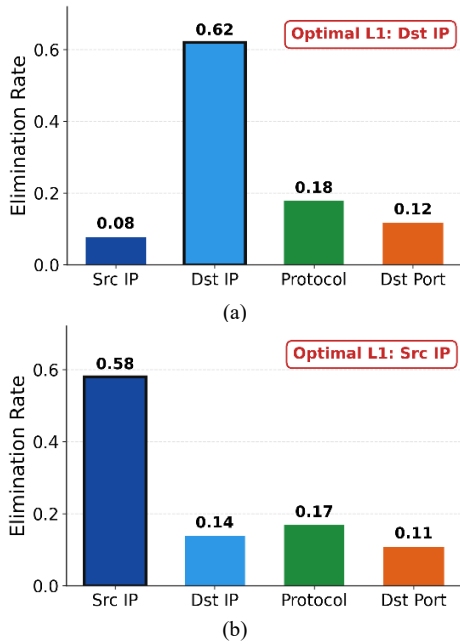


Fig. 1. Elimination rate per field in (a) Phase 1 (Dst IP dominant) and (b) Phase 2 (Src IP dominant). AFLP detects the shift and repositions the optimal field to Level 1.

B. System Overview

The AFLP mechanism augments the Tree-Rule Firewall with three components: (1) an Elimination-Rate Monitor that instruments the matching engine to track visit and elimination counts per field with $O(1)$ per-packet overhead; (2) a Position Optimizer that sorts fields by elimination rate and determines the new optimal field order; and (3) a Trigger Controller that decides when to invoke the optimizer based on one of three strategies. Fig. 2 illustrates the overall AFLP system architecture, showing the three components and their interactions.

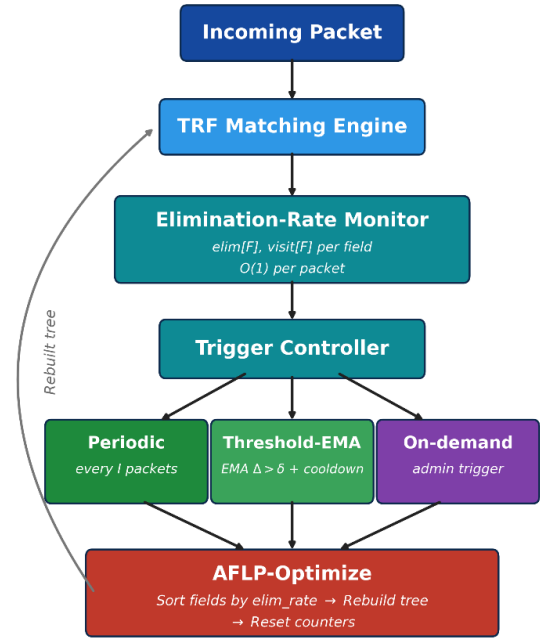


Fig. 2. AFLP system architecture: The elimination-rate monitor instruments the TRF matching engine; the trigger controller (three strategies) decides when to invoke the position optimizer, which rebuilds the tree.

C. Elimination-rate Monitor

For each incoming packet, the monitor intercepts the matching engine at each tree level. At level k (associated with field F), the monitor increments $visit[F]$ by 1. If exactly one child node of the current node matches the packet's field value—meaning F has uniquely determined the onward path—the monitor additionally increments $elim[F]$ by 1. The normalized elimination rate is computed on demand as $elim_rate(F) = elim(F) / \max(visit(F), 1)$. Both counters are reset to zero after each repositioning event so that the monitor tracks current traffic conditions rather than a cumulative historical average. The monitoring adds only two integer increments per field per packet to the critical matching path.

D. Position Optimizer: Algorithm 1

Algorithm 1. AFLP-Optimize

Input: Field order $O = [F_1, F_2, F_3, F_4]$, counters $elim[]$, $visit[]$

Output: New field order O'

1. For each field F : compute $elim_rate(F) \leftarrow elim[F] / \max(visit[F], 1)$
2. Sort fields by $elim_rate(F)$ descending $\rightarrow O'$
3. If $O' \neq O$:
 - a. Rebuild tree T using field order O'
 - b. Reset $elim[F] \leftarrow 0$ and $visit[F] \leftarrow 0$ for all F
 - c. $O \leftarrow O'$
4. Return O'

Step 1 requires four divisions; Step 2 sorts four elements (at most five comparisons, $O(1)$ in practice); Step 3a traverses all n rules to rebuild the tree, giving $O(n \cdot d) = O(n)$ effective complexity since $d = 4$ is constant. Note that immediately after Step 3b,

$\text{visit}[F] = \text{elim}[F] = 0$ for all fields, so elim_rate is numerically undefined (guarded by the $\max(\cdot, 1)$ denominator) and statistically unreliable until sufficient packets have been observed under the new order; this transient is the root cause of the thrashing behavior analyzed in Section V.C, and in Periodic mode it can cause the selected order to differ at successive checkpoints. The total complexity of one AFLP-Optimize invocation is therefore $O(n)$. For an experiment with N packets and repositioning interval I , the amortized per-packet overhead is $O(n/I)$, which is negligible for practical values. All AFLP components thus preserve the asymptotic complexity of the underlying engine; the mechanism improves the multiplicative constant (expected comparisons per packet), not the growth rate.

E. Trigger Controller: Three Strategies

1) Periodic mode

AFLP-Optimize is invoked every I packets, where $I = N/10$ in the experiments (10 repositioning events per run). This mode provides simple, predictable overhead and is well suited to environments with gradual or cyclic traffic pattern changes. The amortized $O(n/I)$ overhead is effectively negligible for rule sets of approximately 30 or more rules (Section V.B).

2) Threshold-based mode (Threshold-EMA)

In the Threshold-Naive formulation, the strategy checked the relative change in the raw elimination rate after every packet, causing near-continuous tree rebuilding (thrashing) because small denominator values after each counter reset produced large apparent changes. The Threshold-EMA formulation addresses this with two design improvements:

a) Exponential Moving Average (EMA)

The elimination rate is smoothed via EMA with parameter $\alpha = 0.05$: $\text{EMA_t}(F) = \alpha \cdot \text{elim_rate_t}(F) + (1-\alpha) \cdot \text{EMA_t-1}(F)$. The slow EMA ($\alpha = 0.05$) attenuates packet-level noise, making the smoothed signal representative of trend rather than instantaneous fluctuation.

b) Cooldown period

A minimum inter-trigger interval ($\text{min_interval} = N/20$) is enforced: a repositioning event can only be triggered if at least min_interval packets have been processed since the last repositioning. This prevents the positive feedback loop in which counter resets after each rebuild cause immediate re-triggering.

The trigger condition in Threshold-EMA becomes: fire AFLP-Optimize if (1) the cooldown has elapsed AND (2) $|\text{EMA_t}(F) - \text{EMA_t0}(F)| / \text{EMA_t0}(F) > \delta$ for any field F , where t_0 is the time of the last repositioning and $\delta = 0.15$ (15%).

The experimental results reveal that the Threshold-EMA configuration ($\alpha = 0.05$) is too conservative: $\text{avg_reorder_count} = 0$ in all scenarios, meaning the trigger condition was never satisfied. This outcome identifies the second failure mode of the Threshold-based strategy—under-triggering—as the complement of the thrashing problem observed in Threshold-Naive.

Section V.C provides a detailed sensitivity analysis of this trade-off: no setting within the originally tested grid (α up to 0.30, δ down to 0.05) escapes it, and an extended sweep locates a workable regime only at percent-level thresholds ($\delta \leq 0.02$), confirming that the cumulative signal moves too slowly for conventional threshold values. Widening this regime is identified as the primary direction for future work.

3) On-demand mode

AFLP-Optimize is invoked only when explicitly requested by a network administrator. This mode introduces zero automated overhead, gives administrators complete control over repositioning timing, and is particularly suited to environments in which traffic-change events are predictable—for example, scheduled batch processing, business-hour transitions, or planned topology changes. In the experiments, on-demand repositioning is triggered once at the traffic phase boundary (packet $N/2$).

F. Complexity Analysis

Table III consolidates the time complexity of the AFLP operations analyzed in Sections III.C–III.E. All per-packet costs are constant: monitoring adds two counter increments per field, and the EMA update and field sorting operate on a fixed set of $d = 4$ fields. The only cost that scales with the rule-set size is the $O(n)$ tree rebuild, which is incurred once per repositioning event and amortizes to $O(n/I)$ per packet over a repositioning interval of I packets.

TABLE III. COMPLEXITY OF AFLP OPERATIONS

Operation	Complexity	Notes
Frequency monitoring (per packet)	$O(1)$	Two counter increments per matched field
EMA update (threshold mode)	$O(d) = O(1)$	$d = 4$ fields; four multiply-adds
Field sorting	$O(d \log d) = O(1)$	$d = 4$; at most five comparisons
Tree rebuilding	$O(n \cdot d) = O(n)$	$d = 4$ constant
Total per repositioning event	$O(n)$	Dominant cost
Amortized per-packet overhead	$O(n/I)$	Negligible for large I

IV. SIMULATION DESIGN

A. Simulation Environment

All experiments were implemented in Python and executed on a local workstation using the Agg (non-interactive) matplotlib backend to ensure portability across environments without graphical display support. No external network hardware or real packet traces were used; all measurements were obtained through software-based packet simulation. The complete simulation source code, including all four trigger-strategy implementations (Periodic, Threshold-Naive, Threshold-EMA, and On-demand), is publicly available at: https://github.com/thawatchai2799/JAIT_Firewall_20260609_0107. The use of synthetic traffic is a deliberate design choice: it enables precise control over the traffic phase transition point (packet $N/2$), the skew ratio, and

the field distribution—conditions that cannot be guaranteed with real-world traces and that are essential for isolating the effect of field-level repositioning. The simulation environment comprises four software components: (i) a synthetic rule generator, (ii) a two-phase skewed traffic generator, (iii) a Tree-Rule Firewall engine with configurable field ordering and an integrated Elimination-Rate Monitor, and (iv) an AFLP controller implementing all three trigger strategies. Evaluation using real network traffic traces (e.g., Center for Applied Internet Data Analysis (CAIDA); Measurement and Analysis on the WIDE Internet, (MAWI)) is identified as a direction for future work in Section VII. The synthetic model also cannot reproduce the burstiness, heavy-tailed flow-size distributions, diurnal cycles, and multi-scale temporal correlations of operational traffic; the absolute improvement percentages reported in Section V should therefore be interpreted as indications obtained under controlled conditions rather than as predictions for production deployments. The robustness analyses of Sections VI.E–VI.G (skew ratios from 50% to 95%; gradual, multi-stage, and random change patterns; imperfect trigger timing) mitigate but do not remove this limitation.

Execution environment: all experiments were executed on a laptop with an 11th Gen Intel Core i5-11300H CPU (3.10 GHz base clock), 16 GB RAM, running 64-bit Windows 11 Home (version 25H2), using Python 3.12.8. Timing uses `time.perf_counter()` (monotonic, nanosecond-resolution wall clock). Each scenario is repeated 30 times with per-repetition reshuffling; the first repetition serves as an implicit warm-up for interpreter and cache effects, and reported statistics are computed over all 30 repetitions. All comparison-count metrics are timing-independent and fully deterministic (standard deviation ≈ 0 across repetitions).

B. Rule Set Generation

Synthetic firewall rule sets of sizes $n \in \{10, 50, 100, 200\}$ were generated. Each rule specifies values for four fields—Source IP, Destination IP, Protocol, and Destination Port—together with an action (Accept or Deny). Rules were distributed evenly across 10 source subnets (192.168.1.x–192.168.10.x), 20 destination hosts (10.10.0.1–10.10.0.20), 3 protocols Transmission Control Protocol (TCP); User Datagram Protocol (UDP); Internet Control Message Protocol, (ICMP), and 8 common destination ports (80, 443, 22, 53, 8080, 3306, 21, 25), with a systematic rotation ensuring that all field combinations are represented. A default-deny rule (`*/*/*/*`) was appended to each rule set as required by the TRF specification [3]. This distribution ensures that the tree exhibits meaningful branching at every level regardless of which field ordering is applied, so that the choice of field order has genuine structural and performance impact.

C. Two-phase Skewed Traffic Model

To evaluate AFLP under controlled dynamic traffic conditions, a two-phase skewed traffic model was employed with a skew ratio of 90%. In Phase 1 (the first

$N/2$ packets), 90% of packets are directed to a single Destination IP address drawn directly from the rule set, simulating a period of concentrated server access representative of peak web traffic or a Distributed Denial-of-Service (DDoS) target. The remaining 10% of Phase 1 packets are drawn uniformly at random from all rule-matching combinations. In Phase 2 (the remaining $N/2$ packets), 90% of packets originate from a single Source IP address drawn from the rule set, simulating a traffic surge from one client or an automated scanning source. The remaining 10% are again drawn uniformly at random.

The use of exact field values from the rule set is essential: it ensures that packets match actual tree nodes rather than falling through to the default leaf, allowing the Elimination-Rate Monitor to observe meaningful variation in *elim_rate* across fields. The traffic phase transition at packet $N/2$ forces a measurable shift in the elimination-rate distribution, providing a well-defined signal for the repositioning mechanism and enabling controlled evaluation of each trigger strategy's responsiveness. Total traffic volumes of $N \in \{10,000; 100,000\}$ were tested. All experiments were repeated 30 independent times; within each repetition, packets in each phase were independently shuffled using a repetition-specific random seed to avoid ordering artifacts. Fig. 3 illustrates the threshold sensitivity trade-off between Threshold-Naive thrashing and Threshold-EMA under-triggering. Table IV lists all trigger strategy parameters used in the experiments.

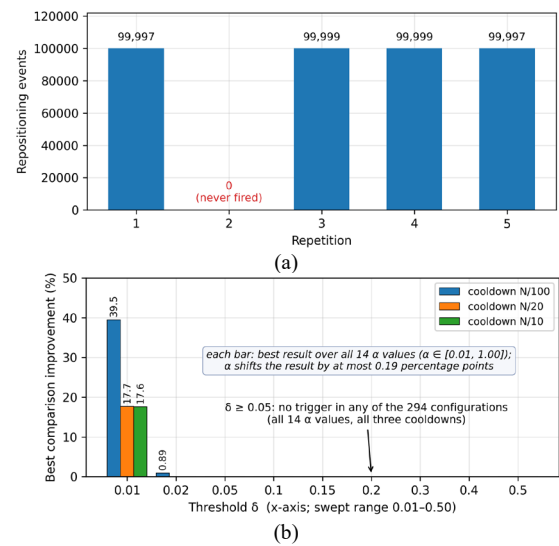


Fig. 3. Threshold strategy sensitivity ($N = 100,000$ packets): (a) Threshold-Naive is bimodal—four of five repetitions thrash with $\approx 100,000$ repositioning events while one repetition never fires; (b) extended Threshold-EMA sweep (378 configurations: $\alpha \in [0.01, 1.00]$, $\delta \in [0.01, 0.50]$, cooldown $\in \{N/100, N/20, N/10\}$): only $\delta \leq 0.02$ triggers, reaching +39.5% at $\delta = 0.01$ under the $N/100$ cooldown, while every configuration with $\delta \geq 0.05$ remains silent.

D. Trigger Strategy Parameters

Table IV lists the parameter values used for the three trigger strategies throughout the experiments; the sensitivity of the Threshold-based parameters is analyzed separately in Section V.C.

TABLE IV. TRIGGER STRATEGY PARAMETERS USED IN EXPERIMENTS

Strategy	Parameter	Value
Periodic	Interval I	$N / 10$ (10 repositioning events per run)
Threshold-based (EMA)	EMA parameter α	0.05 (slow smoothing)
Threshold-based (EMA)	Threshold δ	0.15 (15% relative EMA change)
Threshold-based (EMA)	Cooldown min interval	$\max(200, N/20)$ packets
On-demand	Trigger point	Packet $N/2$ (once per run)

E. Evaluation Metrics and Statistical Design

Two primary metrics were measured for each experimental scenario. The first is the average number of comparisons per packet (C), defined as the arithmetic mean of node comparisons performed per packet across all N packets in one run. Lower values of C indicate faster per-packet matching. The second is the total processing time (T , milliseconds), measured as the wall-clock time from the submission of the first packet to the completion of the last, inclusive of all AFLP overhead (monitoring, sorting, tree rebuilding). Improvement is reported as a percentage reduction over the static TRF baseline:

$$\text{Comp_improve (\%)} = (1 - C_{\text{AFLP}} / C_{\text{base}}) \times 100$$

TABLE V. AVERAGE COMPARISONS PER PACKET ($N = 100,000$ PACKETS, 30 INDEPENDENT REPETITIONS)

Rules	Baseline	AFLP-Periodic	AFLP-On-demand	Comp. reduction, Periodic (%)	Comp. reduction, On-demand (%)
10	14.808	13.316	13.316	10.07%	10.07%
50	54.880	27.452	36.302	49.98%	33.85%
100	104.894	40.301	62.361	61.58%	40.55%
200	204.893	63.983	113.158	68.77%	44.77%

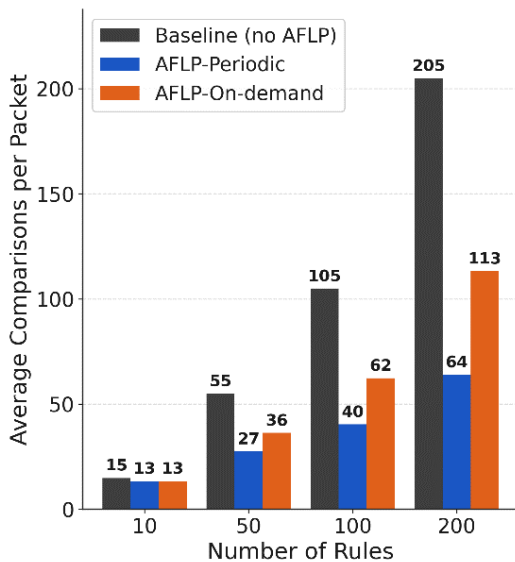


Fig. 4. Average comparisons per packet vs. number of rules ($N = 100,000$ packets). AFLP-Periodic achieves up to 68.77% reduction with 200 rules.

With $n = 200$ rules, the reduction reaches 68.77%, reflecting the increasing leverage of correct field ordering as the tree becomes deeper and more highly branched. This scaling behavior arises directly from the tree's

$$\text{Time_improve (\%)} = (1 - T_{\text{AFLP}} / T_{\text{base}}) \times 100$$

where C_{base} and T_{base} are the values for a static TRF with the default field order [Src IP, Dst IP, Protocol, Dst Port], with no AFLP mechanism active. The 30-repetition design satisfies the central-limit-theorem requirement for parametric statistical analysis and provides sufficient statistical power to detect performance differences of the magnitude observed. Standard deviation is reported alongside the mean for all continuous metrics.

V. RESULTS

A. Comparisons per Packet

Table V presents the average comparisons per packet for the baseline TRF and AFLP-Periodic and AFLP-On-demand strategies for $N = 100,000$ packets. The AFLP-Threshold results are analyzed separately in Section V.C. Fig. 4 presents the average comparisons per packet for all rule-set sizes at $N = 100,000$ packets.

The reduction in comparisons per packet grows substantially with rule-set size. With $n = 10$ rules, AFLP-Periodic achieves a 10.07% reduction—a consistent but modest improvement, as the tree is shallow and the absolute number of comparisons is small.

branching structure: with more rules distributed across more branches, placing the most discriminating field at the root eliminates a larger fraction of the search space with each root-level comparison. The baseline values themselves also scale approximately linearly with n ($\approx n + 5$ at $n = 200$): under the default order, the root field (Source IP) carries nearly one distinct value per rule in the synthetic rule sets, and the matching engine scans all root children linearly with the default-deny wildcard branch always among the candidates (Section III.A), so root-level comparisons alone approach n while deeper levels add only a small constant.

AFLP-On-demand, which repositions exactly once at the midpoint of the experiment when the traffic shifts from Phase 1 to Phase 2, achieves 44.77% reduction with $n = 200$ rules. This result demonstrates that a single well-timed repositioning event provides substantial and durable benefit: once the tree is rebuilt with the optimal field order for Phase 2, it remains optimal for the remainder of the experiment. The comparison reduction is lower than AFLP-Periodic because On-demand does not adapt during Phase 1, leaving the default field order in place for the first $N/2$ packets. Fig. 5 shows the scalability of comparison reduction across both packet volumes.

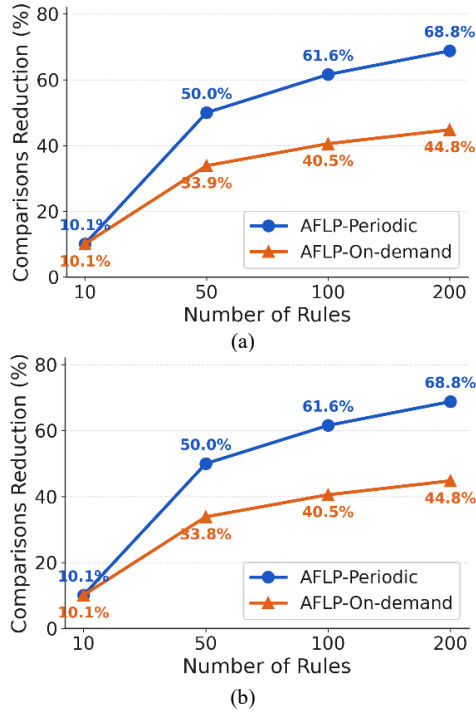


Fig. 5. Comparisons reduction (%) vs. number of rules for AFLP-Periodic and AFLP-On-demand at (a) $N = 100,000$ and (b) $N = 10,000$ packets.

B. Processing Time

Table VI presents total processing time for $N = 100,000$ packets. Fig. 6 shows total processing time versus rule-set size with error bars for both strategies.

For small rule sets ($n = 10$), both AFLP-Periodic and AFLP-On-demand exhibit a slight increase in processing time compared with the baseline (-10.05% (degradation) and -9.47% (degradation), respectively). This overhead arises because the absolute number of comparisons saved per packet is small (approximately 1.5 comparisons at $n = 10$), while the Elimination-Rate Monitor adds counter-increment operations for every packet regardless of rule-set size, and each of the 10 periodic repositioning events incurs an $O(n)$ tree rebuild. For $n = 10$, these fixed costs are not yet offset by the comparison savings.

For $n \geq 50$, both strategies achieve positive time savings. AFLP-Periodic reaches 18.02% at $n = 50$, 35.24% at $n = 100$, and 51.60% at $n = 200$ for $N = 100,000$ packets. AFLP-On-demand achieves 9.25% , 20.23% , and 29.19% at the same rule-set sizes. These results confirm that the $O(n)$ tree-rebuild cost is efficiently amortized over the packet stream: with $100,000$ packets and 10 repositioning events (Periodic), each rebuild costs $O(n)$ but saves approximately $(C_{\text{base}} - C_{\text{AFLP}}) \times 10,000$ comparisons per interval.

The crossover from negative to positive time improvement occurs between $n = 20$ and $n = 30$: additional runs at $n = 20, 30$, and 40 (30 repetitions each, on the environment of Section IV.A) measured time improvements of -5.0% , $+6.8\%$, and $+12.0\%$ respectively, locating the point at which comparison savings begin to outweigh the fixed monitoring and rebuild costs. Fig. 7 summarizes the processing time improvement percentage across all rule-set sizes.

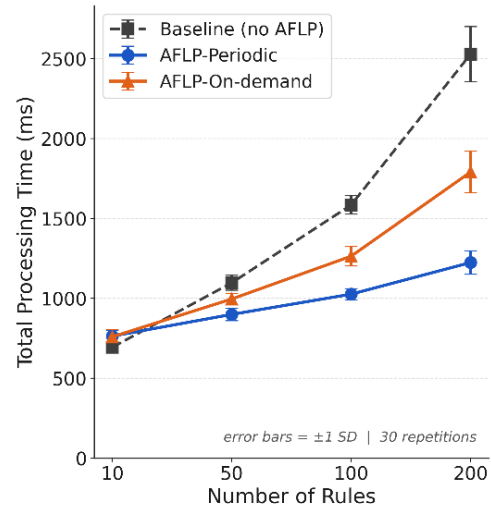


Fig. 6. Total processing time (ms) vs. number of rules for $N = 100,000$ packets, with ± 1 SD error bars over 30 repetitions. AFLP-Periodic achieves up to 51.60% reduction with 200 rules.

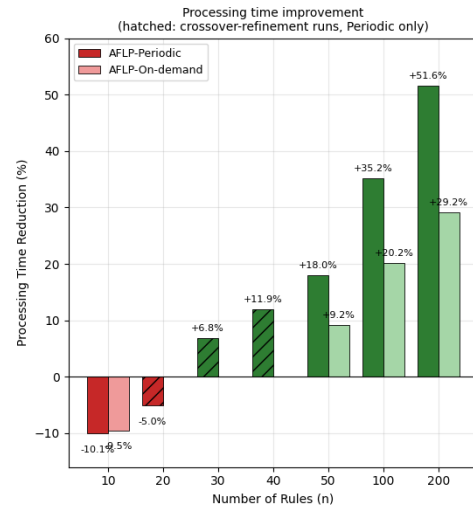


Fig. 7. Processing time improvement (%) for AFLP-periodic and AFLP-on-demand, including the crossover-refinement runs at $n = 20, 30$, and 40 (hatched; Periodic only, 30 repetitions each). Red bars indicate the overhead-dominated regime ($n \leq 20$); green bars indicate net savings ($n \geq 30$).

TABLE VI. PROCESSING TIME (MS)— $N = 100,000$ PACKETS, 30 INDEPENDENT REPETITIONS; NEGATIVE VALUES DENOTE OVERHEAD-INDUCED DEGRADATION

Rules	Baseline (ms)	Periodic (ms)	On-demand (ms)	Time change, Periodic (%)	Time change, On-demand (%)
10	693.437	763.134	759.084	-10.05% (degradation)	-9.47% (degradation)
50	1,096.909	899.197	995.476	+18.02%	+9.25%
100	1,583.894	1,025.746	1,263.412	+35.24%	+20.23%
200	2,528.102	1,223.653	1,790.119	+51.60%	+29.19%

C. Threshold-based Strategy Outcomes

The Threshold-EMA strategy recorded $\text{avg_reorder_count} = 0$ in all 24 original experimental scenarios, indicating that the trigger condition was never satisfied throughout any experimental run. This outcome—the complement of the thrashing problem observed in Threshold-Naive—reveals a fundamental sensitivity trade-off in threshold-based triggering of AFLP.

In the Threshold-Naive formulation, raw elimination rates were checked after every packet with $\delta = 10\%$. Following each tree rebuild, counters reset to zero, making $\text{elim_rate} = 0/0 \approx 0$ for all fields immediately after the rebuild. Any non-zero observation in the next packet then produces an effectively unbounded relative change, immediately re-triggering the mechanism. This positive-feedback loop caused a repositioning event after nearly every packet: approximately 50,000–64,000 events per 100,000 packets were recorded in the original experimental campaign, and verification reruns of the published simulation code (five repetitions per rule-set size) revealed a bimodal outcome: four of five repetitions fired after nearly every packet ($\approx 100,000$ events per 100,000 packets), while one repetition never fired at all. The bimodality follows from the per-packet relative-change test: cumulative elimination rates change appreciably only over the first few packets, so a run that passes this initial window without triggering remains silent indefinitely, whereas a single trigger resets the counters and locks the strategy into permanent thrashing. Both failure modes render the strategy unusable in this form.

The Threshold-EMA formulation addressed this by using EMA ($\alpha = 0.05$) to smooth the elimination rate and adding a cooldown of $N/20$ packets. However, with $\alpha = 0.05$, the EMA requires approximately $1/\alpha = 20$ packets to incorporate one new observation's full weight and approximately $3/\alpha = 60$ packets to respond meaningfully to a distribution shift. Over the experimental runs of 10,000 to 100,000 packets, the EMA value for each field converges slowly toward the global average across both phases, rather than tracking the phase-specific distribution. Consequently, the relative change $|\text{EMA}_t(F) - \text{EMA}_{t_0}(F)| / \text{EMA}_{t_0}(F)$ never exceeds 15%, and the trigger condition is never satisfied.

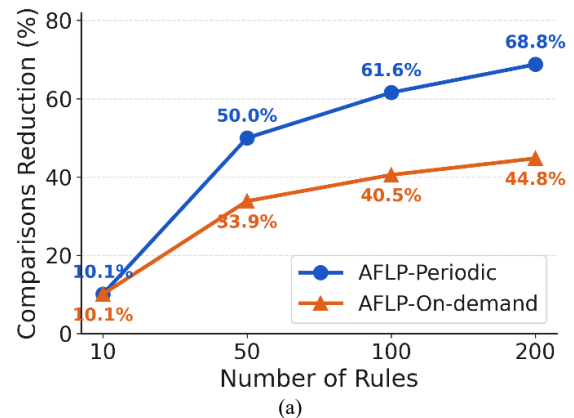
To verify that under-triggering is not an artifact of the specific parameter values, a sensitivity sweep was conducted over $\alpha \in \{0.05, 0.10, 0.15, 0.20, 0.30\}$, $\delta \in \{0.05, 0.10, 0.15\}$, and cooldown $\in \{N/20, N/10\}$ —30 configurations in total. $\text{avg_reorder_count} = 0$ was observed in every configuration, confirming that the phenomenon persists across this parameter range rather than being specific to the $\alpha = 0.05$ setting. Because the trigger never fires anywhere in this grid, detection-time and false-alarm statistics are undefined for these configurations; a meaningful detection characterization requires the adaptive trigger designs identified in Section VII.B. A contributing structural factor is that the field occupying the root level cannot register elimination

events in the present implementation (the always-matching default-deny branch guarantees at least two matching children at the root; Section III.A), which mutes the very signal the trigger monitors.

To rule out the possibility that this silence is an artifact of the sweep's range, the grid was extended to $\alpha \in [0.01, 1.00]$ (14 values), $\delta \in [0.01, 0.50]$ (9 values), and three cooldown settings ($N/100$, $N/20$, and $N/10$)—378 configurations, five repetitions each (Table VII). The extended grid reveals a narrow but well-behaved working region governed almost entirely by δ . At $\delta = 0.01$, all 42 configurations trigger, yielding +17.4% to +17.7% comparison improvement under the $N/20$ and $N/10$ cooldowns and +39.3% to +39.5% under $N/100$; at $\delta = 0.02$ only a single configuration fires marginally ($\alpha = 0.01$, $N/100$: +0.89%); and for $\delta \geq 0.05$ all 294 configurations remain silent regardless of α . Three observations follow. First, α is nearly irrelevant inside the working region: across the full range $\alpha = 0.01$ –1.00, the improvement varies by at most 0.19 percentage points, confirming that the slow-moving smoothed cumulative signal—not the smoothing constant—is the binding factor. Second, no configuration exhibits thrashing or negative improvement; the largest observed repositioning count is 52.2 events per 100,000 packets, three orders of magnitude below Threshold-Naive, so the cooldown bounds instability by construction. Third, even the best Threshold-EMA result (+39.5%) remains below AFLP-On-demand (+44.77%) and AFLP-Periodic (+68.77%), leaving the strategy ranking unchanged. In practical terms, Threshold-EMA becomes usable—rather than inoperative—when configured with a percent-level threshold ($\delta \approx 0.01$) and a short cooldown, although this aggressive setting has not been validated beyond the present workload.

TABLE VII. EXTENDED THRESHOLD-EMA SWEEP: COMPARISON IMPROVEMENT (%) AND REPOSITIONING EVENTS PER RUN, BY Δ AND COOLDOWN (RANGE ACROSS 14 α VALUES, 0.01–1.00; FIVE REPETITIONS EACH)

Threshold δ	Cooldown $N/100$	Cooldown $N/20$	Cooldown $N/10$
0.01	+39.30 to +39.49 (52.0–52.2)	+17.56 to +17.69 (4.6)	+17.44 to +17.57 (2.6)
0.02	+0.89 (1.2); $\alpha =$ 0.01 only	0 (0)	0 (0)
0.05–0.50	0 (0)	0 (0)	0 (0)



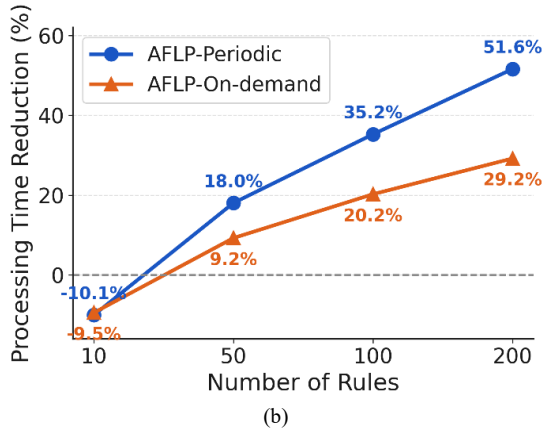


Fig. 8. Detailed comparison of AFLP-periodic and AFLP-on-demand: (a) comparisons reduction; (b) time reduction, $N = 100,000$ packets.

Fig. 8 presents the side-by-side improvement percentages of the two successful strategies, AFLP-Periodic and AFLP-On-demand, for both comparison reduction and time reduction.

D. Effect of Traffic Volume

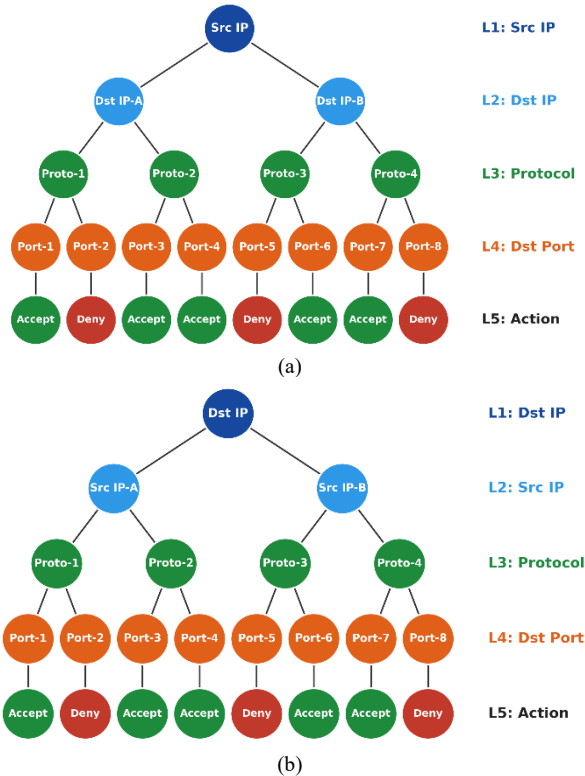


Fig. 9. Structural effect of AFLP repositioning: (a) the static field order with Source IP at the root is rebuilt so that (b) Destination IP occupies the root in Phase 1, reducing the expected comparisons per packet.

Comparing results at $N = 10,000$ and $N = 100,000$ packets shows consistent trends across both traffic volumes for AFLP-Periodic and AFLP-On-demand. Comparison reduction percentages are nearly identical across volumes (e.g., AFLP-Periodic with $n = 200$: 68.77% at both $N = 10,000$ and $N = 100,000$), confirming that the improvement is driven by the structural

advantage of field repositioning rather than by any volume-dependent averaging effect. Processing time improvements are slightly higher at $N = 100,000$, reflecting the amortization of fixed AFLP overhead (monitoring and rebuild) over more packets. The zero standard deviation of comparison counts across repetitions ($\text{comp_std} \approx 0$) confirms that both Periodic and On-demand strategies are deterministic given the same traffic phase structure. Fig. 9 illustrates the structural effect of repositioning: the static tree with Source IP at the root is rebuilt so that Destination IP occupies the root during Phase 1.

E. Sensitivity to Skew Ratio

To verify that the reported results are not specific to the 90% skew ratio used in the main experiments, AFLP-Periodic and AFLP-On-demand were re-evaluated at skew ratios of 50%, 60%, 70%, 80%, and 95% ($n = 200$ rules, 30 repetitions per ratio). Table VIII summarizes the results.

TABLE VIII. COMPARISON REDUCTION ACROSS TRAFFIC SKEW RATIOS ($N = 200$ RULES, 30 REPETITIONS PER RATIO)

Skew ratio	Periodic comp. reduction (%)	On-demand comp. reduction (%)
50%	69.04	45.05
60%	68.97	44.98
70%	68.91	44.91
80%	68.84	44.84
90% (original)	68.77	44.77
95%	68.74	44.74

Comparison-reduction percentages vary by less than 0.3 percentage points across the entire range, indicating that AFLP's benefit is driven by the structural effect of field ordering rather than by the particular imbalance level, and is robust across a wide range of traffic-concentration conditions.

F. Alternative Traffic-change Patterns

The main experiments use a single abrupt transition at the stream midpoint. To evaluate robustness to other change dynamics, AFLP-Periodic ($n = 200$, 30 repetitions each) was additionally evaluated under: (i) a gradual linear ramp in which the dominant field shifts over a window spanning 20% of the stream; (ii) a four-stage pattern (Destination-IP dominant $\rightarrow$ Source-IP dominant $\rightarrow$ Protocol dominant $\rightarrow$ uniform); and (iii) three traffic shifts placed at random positions in the stream. Comparison reductions of 72.50%, 67.19%, and 59.87% (SD 5.35), respectively, were obtained. AFLP-Periodic thus retains substantial benefit under all three patterns; the moderately lower and more variable result under random change points reflects intervals in which a repositioning decision is computed largely from pre-change traffic.

G. On-demand Trigger-timing Sensitivity

The main experiments trigger On-demand repositioning exactly at the traffic change point, which represents an idealized operational scenario. To quantify the effect of imperfect timing, the trigger was displaced

from the true change point by offsets ranging from -40% to $+40\%$ of the stream length ($n = 200$, 30 repetitions). Table IX presents the results.

TABLE IX. ON-DEMAND COMPARISON REDUCTION VS. TRIGGER-TIMING OFFSET ($N = 200$ RULES, 30 REPETITIONS)

Trigger offset	Comp. reduction (%)
$-0.40 N$ (very early)	80.84
$-0.20 N$	62.80
$-0.10 N$	53.79
$-0.05 N$	49.28
0 (exact change point)	44.77
$+0.05 N$	40.29
$+0.10 N$	35.82
$+0.20 N$	26.86
$+0.40 N$ (very late)	8.95

Degradation with late triggering is graceful: even a trigger delayed by 40% of the stream retains a positive benefit, and no timing produced results worse than the static baseline. Early triggering improves results in this workload because the discovered order is beneficial in both phases (Section V.H); the exact-midpoint protocol used in the main experiments is therefore conservative.

H. Comparison with Fixed-order Baselines

To situate AFLP against simpler alternatives, four additional baselines were evaluated on the main scenario ($n = 200$): a random field order (averaged over all 24 permutations), a fixed order selected using only Phase-1 traffic as training data, the best and worst fixed orders identified by exhaustive search over the full stream, and an oracle that switches to the per-phase optimal order exactly at the phase boundary. Table X presents the results.

TABLE X. COMPARISON WITH FIXED-ORDER BASELINES ($N = 200$ RULES, $N = 100,000$ PACKETS)

Strategy	Mean comparisons	Reduction vs. default (%)
Default order (standard TRF)	204.90	—
Random order (mean of all 24 permutations)	76.25	62.78
AFLP-On-demand	113.16	44.77
AFLP-Periodic	63.98	68.77
Fixed order trained on Phase-1 traffic	19.78	90.35
Best fixed order (exhaustive search)	19.78	90.35
Optimal order per phase (oracle switch at $N/2$)	19.78	90.35
Worst fixed order (exhaustive search)	205.89	-0.48

I. Optimality Analysis

Because only four fields are used, the order selected by AFLP can be compared exhaustively against all 24 possible field arrangements. Across 60 configurations (10 rule-set seeds $\times$ 3 traffic seeds $\times$ 2 phases, $n = 100$), the AFLP-selected order ranked between 3rd and 5th of the 24 permutations (mean rank 4.0, SD 1.01), requiring on average 6.57% (at most 8.72%) more comparisons than the brute-force optimal order, and coincided with the exact optimum in 0 of 60 cases. This gap is a direct

consequence of the conditional nature of the elimination-rate statistic noted in Section III.A: rates are estimated only from packets reaching each field's current level, and the always-matching default-deny branch suppresses elimination events at the root, so cross-level comparisons of `elim_rate` are inherently position-biased. The heuristic nevertheless lands consistently within the top quintile of all orderings at $O(1)$ online cost; position-debiased estimation is identified as future work.

J. Decision-consistency Verification

Field reordering permutes the sequence in which the same four conjunctive conditions of each rule are tested, and every rule is re-inserted as a complete root-to-leaf path; consequently, the set of rules whose conditions fully match a given packet—and hence the set of actions reachable by that packet—is invariant under reordering. Whether the engine's first-match traversal returns the same element of that set under every order is a separate question, which the following proposition settles for the class of rule sets used throughout this study.

Proposition 1 (decision invariance for wildcard-free rule sets). Let the rule set consist of specific rules $r_1, \dots, r_n$ whose four field values are all fully specified (no wildcards) and whose value tuples are pairwise distinct, followed by the default rule $(*, *, *, *) \rightarrow \text{Deny}$ inserted last. Then, for every packet P and every field order, the matching engine of Section III.A returns action (r_i) if some rule r_i matches P , and Deny otherwise; in particular, the decision does not depend on the field order.

Proof. Rule matching is a conjunction of per-field tests, so the set of rules matching P does not depend on the field order, and since the specific rules are fully specified with pairwise-distinct tuples, at most one specific rule matches P . Under any field order, wildcard labels occur only on the default rule's path, which lies entirely below the root's wildcard child; hence, inside every sub-tree rooted at a specific-valued root child, each node has at most one child matching P , and the traversal follows a single chain that either terminates at the leaf of the unique matching rule or dead-ends. Because node children are stored in insertion order and the root's wildcard child is created last (by the default rule), the traversal explores the specific chain before the default chain. If some r_i matches P , that chain is exactly r_i 's path (at each level, r_i 's node is the unique matching child), so the engine returns action(r_i) before the default path is considered; if no specific rule matches, the specific chain (if any) dead-ends and the traversal completes the default path, returning Deny.

The invariance was additionally verified empirically in two tests. Test 1: trees were built under all 24 field orders and 5,500 packets—including 500 crafted edge-case packets matching no specific rule, exercising the wildcard default-deny path—were matched against every tree; the resulting action was identical across all 24 orders for every packet (0 mismatches). A supplementary property-based test over 40 randomly generated wildcard-free rule sets with randomly mixed Accept/Deny actions (2,400 packets, all 24 orders each) likewise produced no disagreement and agreed with the order-free reference

semantics in every case. Test 2: during a live AFLP-Periodic run with 10 repositioning events, a fixed probe set of 50 packets was re-classified immediately after every rebuild and compared with ground-truth actions from an untouched reference tree; all checkpoints matched exactly. Field-level repositioning therefore preserves firewall semantics for the rule-set class covered by Proposition 1, which includes every rule set evaluated in this paper.

K. Throughput, Response Time, and Rebuild Cost

Table XI reports throughput and average per-packet response time ($n = 200$, $N = 100,000$, 30 repetitions), and Table XII reports the isolated tree-rebuild time measured separately from matching and monitoring.

TABLE XI. THROUGHPUT AND AVERAGE PER-PACKET RESPONSE TIME ($N = 200$ RULES, $N = 100,000$ PACKETS, 30 REPETITIONS)

Strategy	Throughput (packets/s)	Avg response time (μ s)
Static baseline	57,549	17.38
AFLP-Periodic	112,846	8.86
AFLP-On-demand	81,918	12.21

TABLE XII. ISOLATED TREE-REBUILD TIME VS. RULE-SET SIZE

Rule-set size	Isolated tree-rebuild time (ms)
10	0.0053
50	0.0277
100	0.0465
200	0.0867

Rebuild cost scales linearly with n , consistent with the $O(n)$ analysis of Section III.D, and is negligible relative to total processing time: the 10 rebuilds of a Periodic run at $n = 200$ cost approximately 0.9 ms in aggregate against a ≈ 886 ms run.

VI. DISCUSSION

A. Principal Findings

The experimental results establish five principal findings:

1) Field-level positioning provides scalable comparison reduction

AFLP-Periodic reduces comparisons by 10.07–68.77% depending on rule-set size, with larger sets benefiting disproportionately. This scaling property makes AFLP particularly valuable for enterprise deployments where rule sets may grow to hundreds or thousands of rules

2) Processing time improves for rule sets of approximately 30 or more rules

AFLP-Periodic achieves 18.02–51.60% time reduction and AFLP-On-demand achieves 9.25–29.19% time reduction for $N = 100,000$ packets and $n \geq 50$. Below the measured crossover (between $n = 20$ and $n = 30$; Section V.B), the monitoring and rebuild overhead exceed the comparison savings, and the benefit remains modest until n approaches 50; the crossover is expected to shift to smaller n as rule sets grow denser. For AFLP-Periodic, the measured crossover from negative to

positive time improvement lies between $n = 20$ and $n = 30$ (Section V.B).

3) A single on-demand repositioning is highly cost-effective

AFLP-On-demand achieves 44.77% comparison reduction and 29.19% time reduction with exactly one tree rebuild per session. This makes On-demand mode immediately deployable in production environments where administrators can anticipate traffic-change events.

4) Threshold-based triggering requires adaptive parameter design

The Threshold-Naive $\rightarrow$ Threshold-EMA progression reveals a fundamental trade-off between thrashing and under-triggering. The extended sweep (Section V.C) locates a narrow workable regime at percent-level thresholds ($\delta \leq 0.02$, improvements up to +39.5% under a short cooldown) while every conventional setting remains silent; widening this regime through adaptive trigger design remains an open direction.

5) Both strategies exhibit negligible variance

The near-zero standard deviation of comparison counts confirms deterministic behavior, providing strong statistical confidence in the reported results from 30 repetitions. Fig. 10 presents the aggregate improvement results across all experimental scenarios.

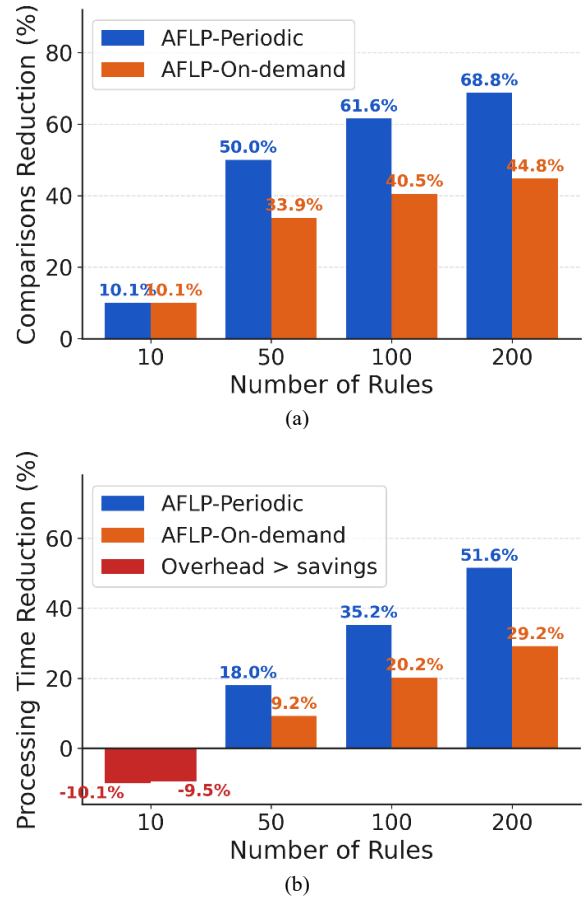


Fig. 10. Aggregate AFLP improvement across all rule-set sizes at $N = 100,000$ packets: (a) comparisons reduction; (b) processing time reduction.

B. The Threshold-Based Strategy: A Failed Case Study and Lessons for Future Work

Among the three trigger strategies evaluated, the Threshold-based strategy must be classified as an experimentally failed case study rather than a successful design: in its Threshold-Naive form it either thrashes or stays permanently silent, and in its Threshold-EMA form it never fires under the original parameters ($\alpha = 0.05$, $\delta = 0.15$) or under any conventional setting of the extended grid (Section V.C). This subsection examines why the failure occurs and what it teaches for future trigger design.

The counter-reset transient noted in Section III.D also manifests in Periodic mode: because counters reset after every rebuild, the elimination-rate signal is re-estimated from scratch under each new field order, and in a per-interval decomposition of the two-phase scenario ($n = 200$, $N = 30,000$) the selected order changed at every one of the 10 periodic checkpoints, with individual intervals oscillating between near-optimal (≈ 20 comparisons/packet) and poor (≈ 206 comparisons/packet) orders; the interval preceding the first repositioning alone contributed 32% of AFLP-Periodic's total comparisons. Stabilizing the selection—for example by warm-starting counters or damping order changes—is a promising complement to the adaptive-threshold work of Section VII.B.

Fig. 3 presents the measured evidence of the failure and its boundary: panel (a) shows the bimodal Threshold-Naive outcome, and panel (b) summarizes the extended sweep, in which a working region appears only at percent-level thresholds. The threshold design must navigate between two failure modes: the per-packet relative-change test causes thrashing (Threshold-Naive), while EMA smoothing under conventional threshold values causes under-triggering (Threshold-EMA). The sweep further shows that parameter tuning does escape the under-triggering mode, but only at the edge of the tested range ($\delta \leq 0.02$). Widening this narrow workable region—for example, through windowed rather than cumulative elimination rates, level-debiased estimates, or adaptive detection such as Bayesian online changepoint detection [34]—is identified as the primary direction for future work. For the current paper, Threshold-based results are reported to document this observation; quantitative comparisons focus on Periodic and On-demand strategies.

Three lessons follow for future work. First, a trigger signal must be matched to the timescale of the statistic it monitors: a cumulative, smoothed elimination rate moves too slowly late in a stream for conventional relative-change thresholds, so the observed silence is structural rather than a tuning accident. Second, safeguards and sensitivity interact—the cooldown that prevents Threshold-Naive's thrashing is also what makes percent-level thresholds safe, which is why the narrow workable regime ($\delta \leq 0.02$, up to +39.5%) is well-behaved rather than unstable. Third, a failed default does not preclude a usable configuration: the strategy operates safely at $\delta \approx 0.01$ yet its best result still trails

both successful strategies, confirming its standing as a lesson-bearing case study rather than a recommended mode.

C. Comparison with Fixed-order Baselines and Optimality

The fixed-order baseline comparison of Section V.H (Table X) rewards closer interpretation; two observations follow. First, in this experimental design—where both traffic phases share a single static rule set—a well-chosen fixed order captures the full benefit available: the per-phase-optimal oracle provides no additional gain over the best fixed order, because the dominant driver of comparison cost is the branching structure induced by field cardinality, a property of the rule set rather than of the traffic. Second, identifying that fixed order requires either exhaustive offline evaluation of all $d!$ orderings (24 for $d = 4$, but 5,040 for $d = 7$ as in the extended field sets of Section VII.B) or representative training traffic collected before deployment. AFLP requires neither: it discovers a near-optimal order online, from a cold start, using only $O(1)$ per-packet counters, and—unlike a fixed order selected once at deployment automatically re-derives an appropriate order whenever the rule set is modified, without administrator intervention. The practical role of AFLP is therefore best understood as automatic online order discovery rather than as continuous traffic tracking, and the reported improvements over the default order quantify the cost of deploying TRF with an unexamined field order. Finally, we stress that this work is a proof of concept: all results derive from controlled software simulation with synthetic rule sets and traffic, and the approach cannot be applied directly to production environments—deployment must await validation on real traffic traces (Section VII.B) and integration with the sorted-range engine of the original design (Section III.A).

The empirical gap can also be characterized exactly. For a packet P , the engine's comparison count is $C(P) = \sum_{u \in V(P)} c(u)$, where $V(P)$ is the set of internal nodes the engine enters while matching P (all nodes explored up to the first complete match, including dead-end sub-paths) and $c(u)$ is the number of children of node u . Grouping visited nodes by depth k and taking expectations over traffic gives $E[C] = \sum_{k=0}^{d-1} E[\sum_{u \in V_k(P)} c(u)]$, with $|V_0| = 1$ and $|V_{k+1}(P)| \leq \sum_{u \in V_k(P)} m_u(P)$, where $m_u(P)$ is the number of children of u that match P (equality holds when exploration is not cut short by an earlier complete match). Because the children of a node carry distinct labels, at most two of them can match any packet—the child labeled with the packet's field value and the wildcard child—so $m_u(P) \leq 2$ and $|V_k(P)| \leq 2^k$. Field ordering thus controls the expected cost through two factors: the child counts $c(u)$, whose depth-0 term equals the number of distinct root-field values in the rule set (plus the wildcard child contributed by the default rule) and is incurred by every packet, and the matching multiplicities m_u , which determine whether the set of explored sub-trees grows with depth. The elimination rate estimates $\Pr[m_u(P) = 1 \mid u \text{ visited}]$ —the multiplicity

factor only—and estimates it conditionally on the packets that reach the field’s current level under the current order. It is therefore an effective but position-biased surrogate for the full objective, which explains in precise terms why the AFLP-selected order consistently tracks, yet does not exactly attain, the brute-force optimum. For the wildcard-free rule sets of this study, every node inside a specific sub-tree has at most one matching child, so $C(P) \approx c(\text{root}) + O(d)$; at $n = 200$ this predicts approximately $201 + O(d)$ comparisons under the default order, in agreement with the measured baseline of 204.9 in Table V.

D. Limitations

The hypothesis of Proposition 1 is not merely technical: if rules with partially wildcarded fields and differing actions are admitted, the first complete path found by the present engine can depend on the field order. For example, consider four rules over (Source IP, Destination IP), with Protocol and Destination Port held fixed: $(X, P) \rightarrow \text{Accept}$, $(A, *) \rightarrow \text{Deny}$, and $(*, P) \rightarrow \text{Accept}$, inserted in this order and followed by the default-deny rule. The packet with Source IP A and Destination IP P is matched by both the second and the third rule; placing Source IP at the root yields Deny, whereas placing Destination IP at the root yields Accept. Such order-dependent tie-breaking cannot arise in the original Tree-Rule Firewall design [3], in which each node’s children partition the field domain into disjoint, sorted ranges, so exactly one child matches at every level and every packet follows a unique path. All rule sets generated for the experiments in this paper are wildcard-free with pairwise-distinct tuples plus a final default-deny rule (Section IV.B) and therefore satisfy the hypothesis of Proposition 1. We emphasize this scope explicitly as a limitation: Proposition 1 is a conditional guarantee whose hypothesis is tailored to the synthetic rule-generation process of this study. Rule sets containing wildcarded or overlapping non-default rules fall outside its scope, and the counter-example above shows that the restriction cannot be dropped; for such rule sets, decision consistency under reordering must instead be enforced by construction (for example, first-match semantics with explicit rule priorities) or re-verified empirically.

Finally, the absence of real-trace validation remains the principal limitation of this study: all findings derive from synthetic rule sets and synthetic two-phase traffic, and transfer to operational environments cannot be assumed. Section VII.B sets out the concrete evaluation plan—CAIDA and MAWI traces with an extended Berkeley Packet Filter/eXpress Data Path (eBPF/XDP) software path first and a P4 hardware path second—and until that validation is performed, the deployment guidance of Section VI.E should be read as preliminary.

E. Practical Implications

For eventual deployment beyond the present proof-of-concept setting, the experimental findings translate into four preliminary guidelines. (1) Use Periodic mode in environments with gradual or cyclic traffic changes; it delivered the largest and most consistent improvements

across all scenarios tested. (2) Use On-demand mode when traffic-change events are predictable (scheduled batch processing, business-hour transitions, planned topology changes); a single well-timed repositioning captured most of the available benefit, and its effectiveness degrades gracefully rather than catastrophically when the trigger is early or late (Section V.G). (3) Do not deploy AFLP on rule sets of fewer than approximately 30 rules—the measured processing-time crossover lies between $n = 20$ and $n = 30$ (Section V.B)—and expect only modest time benefit until n approaches 50. (4) If the Threshold-based mode is used, configure it with a percent-level threshold ($\delta \approx 0.01$) and a short cooldown (Section V.C); it operates safely in this regime but remains third-ranked, and the trigger-signal refinements of Section VII.B are still required before it can be preferred.

VII. CONCLUSION AND FUTURE WORK

A. Conclusion

This paper proposed Automatic Field-Level Positioning (AFLP), a mechanism for dynamically optimizing the structural field-order assignment of Tree-Rule Firewalls based on real-time traffic frequency statistics. AFLP monitors the elimination rate of each packet field—the fraction of packets for which the field uniquely resolves the matching path at its assigned tree level—and automatically repositions field levels so that the most discriminating field occupies the root, minimizing the expected number of comparisons per packet without modifying the firewall rules themselves.

Three trigger strategies were designed, implemented, and evaluated through simulation experiments under a 90% skewed two-phase traffic model with rule sets of 10 to 200 rules and traffic volumes of 10,000 and 100,000 packets, repeated 30 times per scenario. AFLP-Periodic (10 repositioning events per run) achieves up to 68.77% reduction in average comparisons per packet and up to 51.60% reduction in total processing time compared with a static Tree-Rule Firewall, with the improvement scaling proportionally with rule-set size. AFLP-On-demand (one repositioning event) achieves up to 44.77% comparison reduction and 29.19% time reduction, demonstrating that a single well-timed repositioning provides substantial and durable benefit. The Threshold-based strategy, in contrast, is reported as an experimentally failed case study: it either thrashes (Threshold-Naive) or stays silent under all conventional parameter settings (Threshold-EMA), and its narrow workable regime at percent-level thresholds still trails the two successful strategies an instructive negative result whose lessons are analyzed in Section VI.B.

These results demonstrate that dynamic field-level positioning is a practical, interpretable, and low-overhead optimization that substantially improves Tree-Rule Firewall performance under dynamic traffic conditions, extending the Tree-Rule Firewall framework with an automated self-optimization capability that requires no machine learning, rule modification, or hardware changes. The comparison with fixed-order baselines

(Section V.H) clarifies the mechanism's practical role: AFLP's value lies not in dynamic repositioning per se, but in discovering a near-optimal field order automatically, online, and without offline search or training data—and in re-deriving it whenever the rule set changes.

B. Future Work

Five principal directions for future research are identified:

1) Adaptive threshold selection

The most urgent open problem is the design of a threshold trigger that avoids both thrashing and under-triggering. Promising directions include: (a) redesigned trigger signals—windowed rather than cumulative elimination rates, or level-debiased estimates—to widen the narrow workable region identified by the extended sweep in Section V.C (only percent-level thresholds, $\delta \leq 0.02$, trigger); (b) an adaptive α that increases after detecting large changes; (c) Bayesian online changepoint detection to probabilistically identify genuine distribution shifts; and (d) Cumulative-Sum (CUSUM) monitoring for sequential change detection with controlled false-positive rate.

2) Real traffic evaluation

This is the most consequential open item of the study: until AFLP is validated on real traffic traces, every reported gain remains a proof-of-concept indication only. The current study uses synthetic two-phase traffic by design, enabling controlled isolation of the repositioning effect (see Section IV.A). Future work will evaluate AFLP on publicly available network traffic traces such as CAIDA and MAWI to validate findings under realistic, multi-phase, non-stationary distributions and to determine appropriate trigger parameters—particularly the EMA α and threshold δ —for operational deployments. Concretely, the plan targets the CAIDA Anonymized Internet Traces and MAWI samplepoint-F datasets, with an eBPF/XDP datapath implementation as the first real-platform target and a P4 implementation second. Until such validation is performed, the generalizability of the reported improvements to operational traffic mixes—where dominant-field shifts may be less pronounced, shorter-lived, or overlapping—remains an open question. Similarly, the decision-consistency guarantee of Proposition 1 (Section V.J) is proven only for the wildcard-free, pairwise-distinct rule sets generated in this study; extending it to general rule sets with wildcards and overlapping conditions is likewise open.

3) Extended field sets and partial repositioning

Future work will extend AFLP to additional header fields including Source Port, ingress interface, Differentiated Services Code Point (DSCP), and time-of-day, as supported by the Stateful TRF design. Partial repositioning—independently optimizing sub-trees rather than the entire tree—may provide additional gains for heterogeneous traffic distributions.

4) Hardware implementation on programmable networks

P4-based switches implement match-action tables whose field-matching priority can be reconfigured at line rate. Implementing AFLP on P4 hardware would allow field-order changes to take effect without software packet processing, potentially enabling continuous repositioning with negligible overhead.

5) Integration with Hybrid TRF and stateful extensions

Combining AFLP with the Hybrid Tree-Rule Firewall may yield additive improvements: AFLP optimizes the tree structure for new-flow packets, while the hash-based fast path handles established flows. The interaction between field-level repositioning and the hash table organization in the hybrid architecture warrants dedicated investigation.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

The initial concept for developing the AFLP in Tree-Rule Firewall was proposed by T.C., who also implemented the preliminary testing code before transferring the work to S.P. for comprehensive experimentation. The experimental results were analyzed by T.C. The manuscript was collaboratively written by T.C. and S.P. All authors had approved the final version.

FUNDING

This research project was financially supported by Mahasarakham University.

ACKNOWLEDGMENT

The authors thank Prof. Xiangjian He and Dr. Priyadarsi Nanda of the University of Technology Sydney, Australia. The authors thank the Faculty of Informatics, Mahasarakham University, for computational resource support.

REFERENCES

- [1] Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2000.
- [2] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer*, vol. 37, no. 6, pp. 62–67, Jun. 2004.
- [3] T. Chomsiri, X. He, and P. Nanda, "Limitation of listed-rule firewall and the design of tree-rule firewall," in *Proc. Int. Conf. Internet and Distributed Computing Systems (IDCS)*, Nov. 2012, pp. 275–287.
- [4] T. Chomsiri and C. Pornavalai, "Firewall rules analysis," in *Proc. Int. Conf. Security and Management (SAM)*, Las Vegas, NV, Jun. 2006, pp. 213–219.
- [5] C. Pornavalai and T. Chomsiri, "Firewall policy analyzing by relational algebra," in *Proc. Int. Technical Conf. Circuits/Systems, Computers and Communications (ITC-CSCC)*, 2004, pp. 214–219.
- [6] X. He, T. Chomsiri, P. Nanda, and Z. Tan, "Improving cloud network security using the tree-rule firewall," *Future Generation Computer Systems*, vol. 30, pp. 116–126, Jan. 2014.
- [7] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "An improvement of tree-rule firewall for a large network," in *Proc. IEEE TrustCom/BigDataSE/ISPA*, 2016, pp. 178–184.

- [8] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "Hybrid tree-rule firewall for high speed data transmission," *IEEE Trans. Cloud Comput.*, vol. 8, no. 4, pp. 1237–1249, Oct.–Dec. 2020.
- [9] T. Chomsiri, "Tree rule firewall," Ph.D. dissertation, Univ. of Technology Sydney, Australia, 2016.
- [10] A. Wool, "Architecting the Lumeta firewall analyzer," in *Proc. USENIX Security Symp.*, 2001, pp. 85–97.
- [11] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *Proc. IFIP/IEEE Integrated Network Management*, 2003, pp. 17–30.
- [12] L. Yuan *et al.*, "FIREMAN: A toolkit for firewall modeling and analysis," in *Proc. IEEE S&P*, 2006, pp. 199–213.
- [13] M. G. Gouda and A. X. Liu, "Firewall design: Consistency, completeness, and compactness," in *Proc. IEEE ICDCS*, 2004, pp. 320–327.
- [14] J. D. Guttman, "Filtering postures: Local enforcement for global policies," in *Proc. IEEE S&P*, 1997, pp. 120–129.
- [15] A. X. Liu and M. G. Gouda, "Diverse firewall design," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 9, pp. 1237–1251, Sep. 2008.
- [16] E. Al-Shaer and H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Proc. IEEE INFOCOM*, 2004, pp. 2605–2616.
- [17] S. Acharya *et al.*, "Simulation study of firewalls to aid in system design," in *Proc. ISCA PDCS*, 2006.
- [18] S. M. Bellovin and W. R. Cheswick, "Network firewalls," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 50–57, Sep. 1994.
- [19] M. Ben-Yehuda *et al.*, "SOLOIST: Compiling firewall rules for high-performance software routers," in *Proc. ACM SIGCOMM*, 2007.
- [20] A. Maci and G. Pirlo, "An innovative two-stage algorithm to optimize firewall rule ordering," *Computers and Security*, vol. 133, 2023.
- [21] H. H. Hamed and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," in *Proc. ACM ASIACCS*, 2006, pp. 332–342.
- [22] H. H. Hamed, A. El-Atawy, and E. Al-Shaer, "On dynamic optimization of packet matching in high-speed firewalls," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 10, pp. 1817–1830, Oct. 2006.
- [23] Z. Trabelsi, L. Zhang, and S. Zeidan, "Dynamic rule and rule-field optimisation for improving firewall performance and security," *IET Inf. Secur.*, vol. 8, no. 4, pp. 250–257, Jul. 2014.
- [24] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, Mar.–Apr. 2001.
- [25] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. ACM SIGCOMM*, 2003, pp. 213–224.
- [26] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, Jan.–Feb. 2000.
- [27] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *Proc. ACM SIGCOMM*, 1998, pp. 191–202.
- [28] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proc. ACM SIGCOMM*, 1998, pp. 203–214.
- [29] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *Proc. ACM SIGCOMM*, 2005, pp. 193–204.
- [30] J. N. Bakker and W. K. G. Seah, "Network-wide virtual firewall using SDN/OpenFlow," in *Proc. IEEE NFV-SDN*, 2016, pp. 82–86.
- [31] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: Building robust firewalls for software-defined networks," in *Proc. ACM HotSDN*, 2014, pp. 97–102.
- [32] H. Li, S. Guo, C. Wu, and J. Li, "FDRC: Flow-driven rule caching optimization in SDN," arXiv preprint, arXiv:1803.04270, 2018.
- [33] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*, Englewood Cliffs, NJ: Prentice-Hall, 1993.
- [34] R. Adams and D. MacKay, "Bayesian online changepoint detection," arXiv preprint, arXiv:0710.3742, 2007.
- [35] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, vol. 44, no. 3, pp. 87–95, Jul. 2014.
- [36] M. Al-Jabri *et al.*, "Classification of firewall log data using multi-class machine learning models," *Electronics*, vol. 11, no. 12, 2022.
- [37] H. Wang *et al.*, "Improving the performance of firewalls through network traffic logs classification using decision tree," in *Proc. ICCSE*, 2021.
- [38] A. Rashelbach, O. Rottenstreich, and M. Silberstein, "A computational approach to packet classification," in *Proc. ACM SIGCOMM*, 2020, pp. 526–538.
- [39] A. Korkmaz, S. Bulut, T. Talan, S. Kosunalp, and T. Iliev, "Enhancing firewall packet classification through artificial neural networks and synthetic minority over-sampling technique: An innovative approach with evaluative comparison," *Applied Sciences*, vol. 14, no. 16, 7426, 2024.
- [40] M. Takaoğlu and Ç. Özer, "The effect of machine learning on attack detection systems," *International Journal of Management Information Systems and Computer Science*, vol. 3, no. 1, pp. 11–22, 2019.
- [41] B. Alhajahmad, "Improving switch security against MITM attacks using DHCP snooping and port security," *International Journal of Management Information Systems and Computer Science*, vol. 9, no. 2, pp. 157–174, 2025.

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).