Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Port-Range Rule Management and Matching-Path Measurement for eBPF-Based Network Functions

Bo Xie

xiebo@mail.hfut.edu.cn

Wuhan Digital Engineering Research Institute

Research Article

Keywords: network function management, eBPF, rule processing, packet filtering, matching-path measurement, performance evaluation

Posted Date: October 30th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11234466/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Port-Range Rule Management and Matching-Path Measurement for eBPF-Based Network Functions

Abstract

eBPF is increasingly used to implement programmable network functions on Linux hosts, including boundary filtering, transparent proxying, and firewall-like access control. For network and systems management, such functions require not only correct policy installation but also predictable rule-processing behavior in the data path. This paper studies ZFW, an open-source eBPF-based network function, and identifies an insertion-order-sensitive port-range matching path. In the original implementation, port-range entries are kept in insertion order, so hot-port traffic may traverse many unrelated ranges before reaching the matching rule, while selected miss traffic cannot terminate early even when later ranges are impossible to match. We propose a lightweight management-plane optimization that combines ordered rule insertion and data-plane early termination. The user-space rule maintenance path keeps port-range indexes in ascending order of low_port, while the TC-BPF hot path stops scanning once the destination port is smaller than the current low_port. We further add scan-step instrumentation and reset/read interfaces to support measurement-driven operation and ablation analysis. Experiments covering hit position, rule scale, public-cloud access, local netns/veth environments, mixed workloads, and ablation variants show that the method reduces average scan steps from 100.00 to 1.00 in the core N = 100 scenario. Controlled local tests reduce average latency from 8.10 ms to 6.63 ms and from 3.98 ms to 2.63 ms under lower concurrency. The results show that lightweight rule organization can improve the predictability of eBPF network-function management, while end-to-end latency remains affected by external system and network noise.

Keywords: network function management; eBPF; rule processing; packet filtering; matching-path measurement; performance evaluation

1. Introduction

Extended Berkeley Packet Filter (eBPF) provides a safe and verifiable execution facility inside the Linux kernel. Its technical lineage can be traced back to the original Berkeley Packet Filter for kernel-level packet filtering [1], and modern Linux extends this model through XDP, TC-BPF, maps, helper functions, and the verifier [2-4]. These mechanisms are increasingly used to implement programmable network functions on cloud hosts and edge nodes, including boundary filtering, transparent proxying, access control, traffic pre-filtering, and observability. As a result, part of network-function management is moving from fixed kernel modules or external appliances toward programmable in-kernel rule installation and measurement paths.

For network and systems management, moving network-function logic into eBPF does not automatically make rule processing efficient or predictable. Operators still need to know how rule updates in the management plane affect the per-packet matching path in the data plane. Hot-path overhead is determined by the attachment point, rule organization, index structure, and matching procedure. If user-space rule maintenance does not preserve a useful order, the kernel program may scan many port-range entries that cannot match the packet being processed, increasing the cost and variability of policy enforcement.

This paper focuses on ZFW, an open-source eBPF-based network function with firewall-like rule processing. In its ingress and egress paths, ZFW first locates a rule tuple according to prefix and protocol fields, and then sequentially scans a port-range index table containing low_port/high_port intervals. The observation that rule organization affects the matching path is consistent with prior work on packet classification, Linux firewall mechanisms, firewall rule ordering, and rule-set optimization [8-14]. In the original ZFW implementation, the index table is maintained in insertion order. Therefore, the hit position of a hot-port rule depends on the rule insertion layout. If the hot rule is inserted near the tail of the index table, each packet must traverse many unrelated entries before reaching the matching interval. In selected miss cases, the system also lacks a condition to determine that later entries can no longer match. This path makes the average scan depth grow with both hit position and rule scale, reducing the predictability of managed eBPF network functions as the rule set grows.

Port-range reordering also has a semantic boundary. If overlapping port intervals exist under the same prefix and protocol, sorting only by low_port may change the original first-match priority. Therefore, the optimization studied in this paper assumes non-overlapping intervals, or rule sets in which conflicts have already been explicitly resolved by the policy layer. For overlapping intervals, the user-space policy logic should reject conflicting rules, split ranges, or preserve explicit priorities. This semantic boundary is essential for the correctness of the proposed method.

To address the insertion-order-sensitive scan path, we introduce two complementary mechanisms within the existing prefix-port mapping structure of ZFW. First, the user-space management path maintains port-range indexes in ascending order of low_port during rule insertion. Second, the kernel data path terminates scanning early when the destination port is smaller than the current low_port. To separate mechanism-level benefits from end-to-end deployment noise, we further implement scan-step statistics and construct four build variants for ablation analysis: baseline, sort-only, break-diagnostic, and full-opt. The break-diagnostic variant is used only for diagnosis; early termination without an ordering guarantee is not a safe deployment configuration.

The main contributions of this paper are summarized as follows.

- We identify and quantify an insertion-order-sensitive port-range scanning problem in an eBPF-based network function, showing how management-plane rule layout affects data-plane matching cost even after prefix matching has already selected a rule tuple.
- We propose a low-intrusion ordered-insertion and early-termination mechanism that keeps the original ZFW map layout and command-line rule format largely unchanged while reducing average scan steps for typical hot-port traffic and selected early-terminating miss traffic.
- We implement scan-step statistics, read/reset interfaces, and independent ablation build switches in the user-space management tool and TC-BPF hot path, enabling scripted rule installation, statistics reset, traffic generation, and result aggregation.
- We evaluate the method using hit-position, rule-scale, public-cloud, local netns/veth, mixed-workload, and ablation experiments, and distinguish direct matching-path management benefits from end-to-end latency effects.

2. Related Work

2.1. eBPF, XDP, and kernel programmable data planes

BPF was originally designed to execute user-level packet-capture filters efficiently inside the kernel [1]. Modern eBPF extends this model into a verifier-constrained in-kernel execution environment. XDP has shown that eBPF programs can run in the driver or early receive path to reduce packet processing before packets enter the full networking stack [2]. The Linux kernel documentation further specifies BPF program types, maps, helper functions, and verifier mechanisms, indicating that eBPF has become a mature kernel programmable infrastructure [3].

In network security and traffic processing, eBPF has also been used for in-kernel intrusion detection and traffic pre-filtering, demonstrating that partial rule checks can be moved into the kernel to reduce user-space processing pressure [4]. Unlike complete detection engines, this paper focuses on a local rule-management and matching-path optimization inside a TC-BPF network function. The goal is to reduce per-packet scan cost caused by port-range index organization.

2.2. Linux firewalls, Netfilter, and interval-set mechanisms

Linux has long provided packet filtering, NAT, logging, and packet transformation through Netfilter hooks in the protocol stack. iptables and nftables are both built within this ecosystem [5]. nftables provides sets, maps, verdict maps, and interval sets, enabling addresses, ports, and ranges to be represented by data structures such as hash tables and red-black trees [6]. ipset also supports set types such as bitmap:port, hash:ip,port, and hash:net,port, allowing large address or port conditions to be moved out of linear rule chains [7].

These systems show that Linux firewall implementations have long considered rule-set representation and lookup-path optimization. The proposed method shares the same broad goal of reducing unnecessary traversal in hot paths, but its scope is different. It does not replace Netfilter, nftables, or a general set engine. Instead, it optimizes a specific port-range scan path inside an existing ZFW/eBPF map structure by combining management-plane ordered insertion with data-plane early termination.

2.3. Packet classification and firewall rule-set optimization

Packet classification is a fundamental problem in firewalls, QoS systems, router access control, and SDN switches. Gupta and McKeown showed that multi-field packet classification can have high worst-case complexity and that rule organization strongly affects classification performance [8]. Taylor further summarized how rule scale, field distribution, and lookup structures influence matching efficiency [9]. Subsequent work such as HyperCuts, EffiCuts, and ClassBench advanced packet classification from the perspectives of decision-tree cutting, memory-throughput tradeoffs, and benchmark rule-set construction [10-12].

Firewall rule optimization also studies policy anomalies, rule ordering, and traffic-aware optimization. Firewall Policy Advisor identified that shadowing, redundancy, and conflicts among rules may affect policy correctness [13]. Dynamic rule-ordering methods use traffic characteristics to adjust rule order and reduce average matching cost [14]. This paper follows the insight that rule order affects the average matching path, but it does not dynamically reorder rules based on historical traffic. Instead, it uses the monotonicity of low_port to derive a verifiable early-termination condition.

2.4. Programmable network-function management and performance measurement

SDN, NFV, and P4 emphasize decoupling control logic, network functions, and forwarding paths, enabling network functions to be deployed and evolved in software [15-17]. Modern network management also requires measurement mechanisms that can distinguish control- or management-plane decisions from data-plane effects. High-speed packet generation and measurement work shows that system evaluation must distinguish mechanism-level indicators from end-to-end indicators and must account for noise introduced by traffic generators, links, and system scheduling [18]. Following this principle, this paper treats avg_scan_steps as the primary mechanism metric, while CPU usage, success rate, and latency are used as system-observation metrics.

2.5. Positioning of this work

Existing work mainly targets general classification algorithms, set engines, programmable network-function platforms, or complete security systems. This paper studies a narrower but practical network-management question: given an existing eBPF-based network function, how can rule organization in the management path and pruning in the hot path reduce matching cost for hot ports and selected miss traffic? Compared with rebuilding the rule engine, the proposed method is small and directly embeddable. Compared with generic rule-ordering techniques, it explicitly states the semantic boundary under non-overlapping intervals or conflict-resolved rule sets.

3. Problem Analysis

3.1. Rule organization as a management-plane issue in ZFW

ZFW maps destination prefixes, source prefixes, protocols, and related fields to a tproxy_tuple. Inside each tuple, an index_table stores the low_port values of multiple port ranges. Each low_port is then associated with a high_port, an action, and forwarding information through the existing map structure. From a management perspective, user-space rule insertion determines the layout of this index_table, while the TC-BPF ingress and egress programs consume the resulting layout in the data path. After a packet enters the hot path, the program first locates the corresponding tuple according to prefix and protocol fields, and then sequentially traverses the index_table to check whether the current port range covers the destination port.

This structure is simple and reuses the prefix-matching result. Its weakness is also clear: when the number of port-range entries under a single prefix grows, the hot-path matching cost depends on the position of the target rule in the index_table. If a hot-port rule is located near the tail, each request scans many irrelevant entries.

3.2. Performance bottleneck and semantic risk

In the original implementation, the index_table is maintained by appending entries in insertion order. As a result, rule order has two effects. First, it directly determines performance. If a hot-port rule is inserted at the tail of the index table, each match must linearly scan multiple unrelated entries before reaching the target interval. Average scan steps therefore grow with the hit position. This means that a management-plane insertion decision is reflected as data-plane scan cost.

Second, rule order may affect semantics. A broad allow rule inserted before a more specific deny rule may shadow the latter. Hence, insertion order can determine not only scan complexity but also whether a

policy is enforced as intended. We therefore abstract the problem into two questions: how to keep port-range entries logically ordered under a clear semantic boundary, and how to use this monotonicity to avoid scanning entries that can no longer match.

4. Optimization Design

4.1. Ordered insertion in the management path

To address unstable insertion-order behavior, we introduce ordered insertion in the user-space rule-writing path. When a new port-range rule is inserted into a tproxy_tuple, the system no longer appends the new low_port to the tail of index_table. Instead, it searches for the correct position according to ascending low_port and inserts the entry at that position. The rule-management path therefore maintains an index layout that is usable by the data-plane early-termination logic.

This mechanism has two effects. First, for non-overlapping intervals or rule sets whose conflicts have already been resolved by the policy layer, the ordered index preserves rule actions and provides useful monotonicity. If overlapping ranges or explicit priorities exist under the same tuple, sorting only by low_port is not sufficient for semantic equivalence; conflict detection, range splitting, or explicit priority handling is required in user space. Second, the ordered index provides the necessary precondition for early termination. Once the index_table is monotonic, the kernel program can infer whether later entries still need to be checked according to the relationship between the destination port and the current low_port.

4.2. Early termination

After rules are organized in ascending low_port order, if the current low_port is already greater than the destination port, all following low_port values will also be greater. No later interval can cover the destination port. Therefore, we add the following early-termination condition to the ingress and egress port-range scan loops:

if dst_port < low_port, break the scan loop immediately.

The correctness of this condition comes from low_port monotonicity. Once the current low_port is larger than the destination port, later low_port values can only be larger, so later ranges cannot match. If the index is not sorted by low_port, this inference does not hold and early termination may skip a later matching rule.

Under the same assumption, the range coverage check after the current low_port has been passed can be simplified from low_port <= dst_port <= high_port to dst_port <= high_port, because dst_port >= low_port is guaranteed by the ordered scan and early-termination logic. This simplification is valid only when the index is indeed sorted by low_port.

The optimization does not make all rule sets constant-time. Its benefit depends on the relative position of the destination port in the ordered list and on the distribution of miss traffic. In the hot-port scenarios evaluated in this paper, ordering moves the target rule to an earlier position and thus greatly reduces average scan depth. For miss traffic whose destination port is smaller than the minimum low_port, early termination reduces scan steps to 1. If the destination port is located near the end of the ordered list, or if a miss port is larger than all low_port values, the worst-case path remains O(N).

4.3. Scan-step statistics and ablation support

To quantify hot-path complexity, we add statistics to the ingress and egress scan loops. The implementation accumulates total_steps and total_scans, and user space computes avg_scan_steps. The user-space tool is extended with a --scan-stats interface for reading and resetting scan statistics. This interface makes the optimization observable to operators and experiment scripts instead of relying only on noisy end-to-end latency.

To analyze the contribution of each mechanism, we introduce two independent build switches: ZFW_DISABLE_ORDERED_INSERT_OPT disables ordered insertion, and ZFW_DISABLE_EARLY_BREAK_OPT disables early termination. These switches construct four variants. The baseline variant disables both mechanisms. The sort-only variant enables only ordered insertion. The break-diagnostic variant is used only to diagnose the effect of early termination under specific layouts and is not a safe standalone deployment configuration. The full-opt variant enables both mechanisms and is the recommended deployment version.

5. System Implementation

5.1. User-space rule maintenance

In the user-space zfw tool, we refactor the port-range index insertion logic. The original add_index() function appended entries to index_table. The optimized version searches for the insertion position according to the new low_port, inserts the entry into the correct location, and keeps duplicate-entry detection and index-length checks. This logic is compatible with the original map structure and does not change the existing rule format or command-line arguments.

5.2. Kernel hot-path matching

In zfw_tc_ingress.c and zfw_tc_outbound_track.c, we modify the destination-port scan loop. For each entry, the program first reads low_port. If early termination is enabled and dst_port < low_port, the loop exits directly. Otherwise, the program reads high_port and evaluates the range match. The implementation records the number of scan steps for hit paths, miss paths, and early-termination paths.

5.3. User-space statistics interface

The zfw user-space tool is extended with the --scan-stats option. When used with -L, it outputs total_steps, total_scans, and avg_scan_steps. When used with -F, it resets scan statistics. This interface is used in public-cloud experiments, local controlled experiments, and ablation experiments, and it also illustrates how a lightweight management interface can expose data-plane matching-path behavior.

5.4. Code and experiment-script closure

The implementation contains the mechanisms described in this paper. In user space, add_index() inserts indexes in ascending low_port order unless ordered insertion is disabled by a build switch. In the TC-BPF ingress and egress programs, scan_stats_map and egress_scan_stats_map maintain total_steps and total_scans in the scan loops. The user-space tool reads or resets these values through --scan-stats. The experiment scripts package rule generation and build-version selection, allowing baseline, sort-only, break-diagnostic, and full-opt variants to be built from the same code base.

5.5. Implementation boundary

The current implementation preserves the original MAX_INDEX_ENTRIES = 100 limit for the number of port-range entries under a single prefix. This constraint does not affect the experiments within N <= 100, but it limits evaluation beyond this scale. We do not claim that the current prototype covers larger single-prefix rule sets. For larger rule sets, hierarchical indexing, rule partitioning, or more compact range representations should be considered.

6. Experimental Design

6.1. Experimental environments

We use two experimental environments. The first is a public-cloud environment. The server runs on an Ubuntu cloud host with a Linux 6.8-series kernel and two visible CPUs. The eBPF programs are attached to the TC ingress and egress paths of the primary network interface. The client accesses a Python HTTP server through a public endpoint. This environment is used to confirm that the optimized version runs in a realistic deployment path for managed host network functions. Because public links, cloud scheduling, and short-lived TCP connections introduce measurement noise, this environment is not used as the sole basis for latency claims.

The second environment is a local controlled environment. We use ip netns and veth to construct client and server namespaces on the same Ubuntu host. The server binds to a private test endpoint and a fixed test port inside the isolated namespace, and the client namespace sends requests through the virtual link. This environment reduces public-link noise and helps observe how a shorter matching path affects local end-to-end latency.

6.2. Metrics

We use four metrics: avg_scan_steps, the average number of scanned entries per matching attempt; avg_cpu_usage_pct, average CPU utilization; avg_success_rate_pct, request success rate; and avg_latency_ms, average request latency. The primary mechanism metric is avg_scan_steps because it directly reflects matching-path depth. Latency and CPU utilization are used to observe whether the mechanism-level change is visible at the system level. Since public-cloud latency is affected by external noise, scan steps are used to prove the managed matching-path reduction.

6.3. Experiment groups

We design six experiment groups. The hit-position experiment fixes the total number of rules at 100 and changes the insertion position of the target rule. The rule-scale experiment fixes the target rule as the last inserted rule and varies N = 10, 25, 50, 75, and 100. The public-cloud core comparison uses N = 100, places the target rule at the last insertion position, sets concurrency to 10, and repeats the test 50 times. The local controlled comparison evaluates the baseline and optimized versions in the netns/veth environment. The mixed-workload experiment constructs hotspot, hitmiss, and miss-only workloads. The ablation experiment compares baseline, sort-only, break-diagnostic, and full-opt variants.

6.4. Workload generation and failure definition

In local controlled experiments, each round sends REQ = 2000 requests with CONC = 20 and repeats ROUNDS = 10 times unless otherwise stated. The public-cloud core comparison uses N = 100, the target

rule at the last insertion position, concurrency 10, and 50 repeated rounds. The hit-only workload sends all requests to the hot port 1500. The hotspot workload uses HOT_RATIO = 80 for hot-port requests and sends the remaining requests to non-allowed ports. The hitmiss workload generates hit and miss requests with equal probability. The miss-only workload sends all requests to non-allowed ports. Requests are generated by curl, with a connection timeout of 2 s and a total timeout of 5 s. HTTP status code 200 is counted as success; other status codes or timeouts are counted as failures.

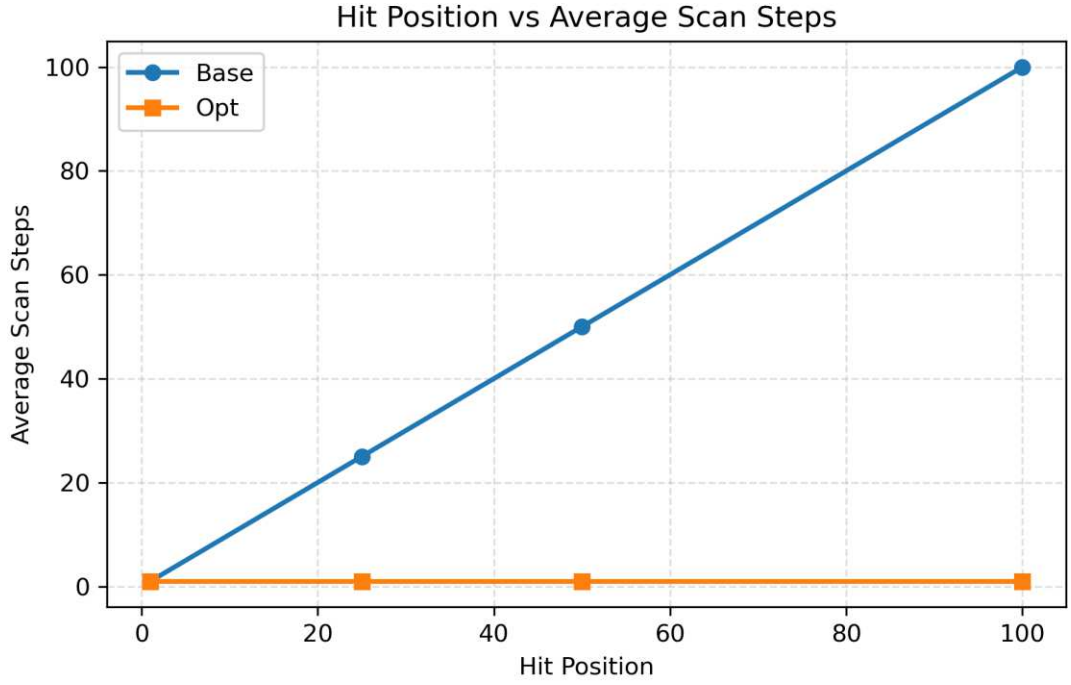
7. Experimental Results and Analysis

7.1. Hit-position experiment

Table 1 reports the average scan steps for different hit positions under N = 100. The baseline scan steps closely follow the target rule position, showing that the original matching cost is linear in the hit position. In contrast, the optimized version remains at 1.00 for all four positions. This result shows that the proposed method eliminates the scan-depth increase caused by placing a hot rule near the tail of the insertion sequence.

Table 1. Average scan steps under different hit positions (N = 100).

Hit position	Baseline avg_scan_steps	Optimized avg_scan_steps	Reduction
1	1	1	0%
25	25.01	1	96%
50	50.01	1	98%
100	100	1	99%

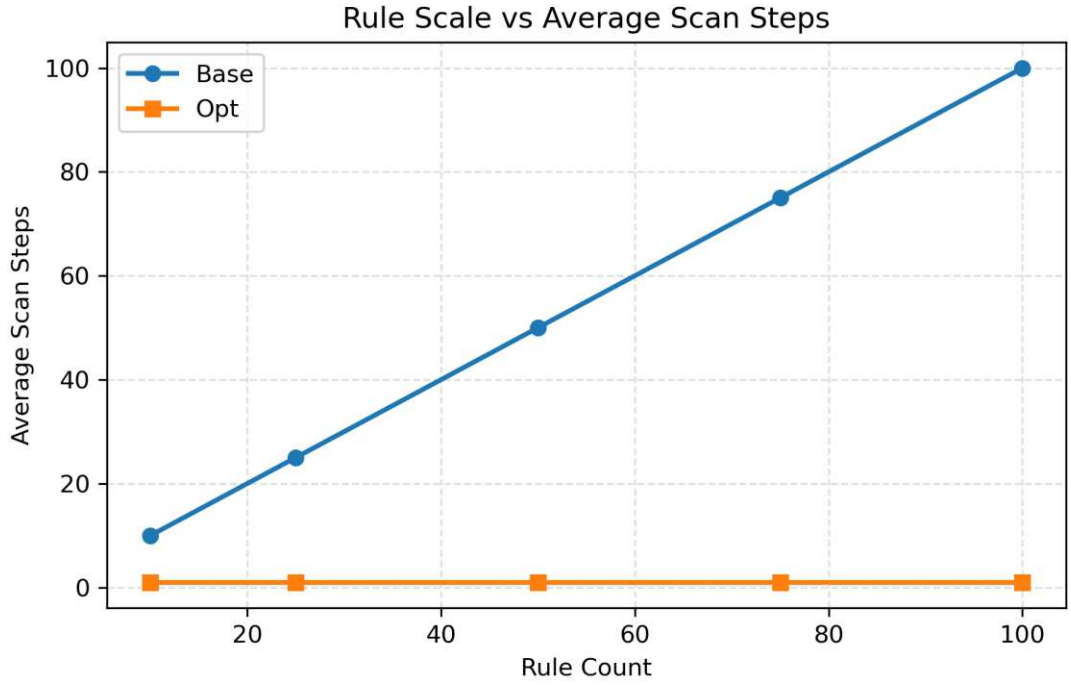


7.2. Rule-scale experiment

Table 2 reports the average scan steps under different rule scales. In the evaluated setting, the target rule is moved to the early part of the index by the optimized ordering. The baseline scan steps grow almost linearly with the number of rules, while the optimized version remains at 1.00 for N <= 100. This result demonstrates that the method substantially reduces average scan steps for this hot-rule pattern. It does not imply constant worst-case complexity for all port distributions or all rule sets.

Table 2. Average scan steps under different rule scales.

Number of rules N	Baseline avg_scan_steps	Optimized avg_scan_steps	Reduction
10	10	1	90%
25	25	1	96%
50	50	1	98%
75	75	1	98.67%
100	100	1	99%



7.3. Public-cloud core comparison

Table 3 gives the core comparison in the public-cloud environment. The optimized version reduces average scan steps from 100.00 to 1.00, corresponding to a 99.00% reduction. Meanwhile, average latency decreases only from 270.36 ms to 265.29 ms, or 1.88%. This observation indicates that in a realistic deployment path, the main benefit of the optimization lies in reducing the matching complexity of a managed eBPF network function, while end-to-end latency is still dominated by link and system-level factors outside the matching loop.

Table 3. Public-cloud core comparison.

Version	avg_scan_steps	avg_cpu_usage_pct	avg_success_rate_pct	avg_latency_ms
Baseline	100	3.74	97.48	270.36
Optimized	1	3.85	97.13	265.29

7.4. Local controlled experiments

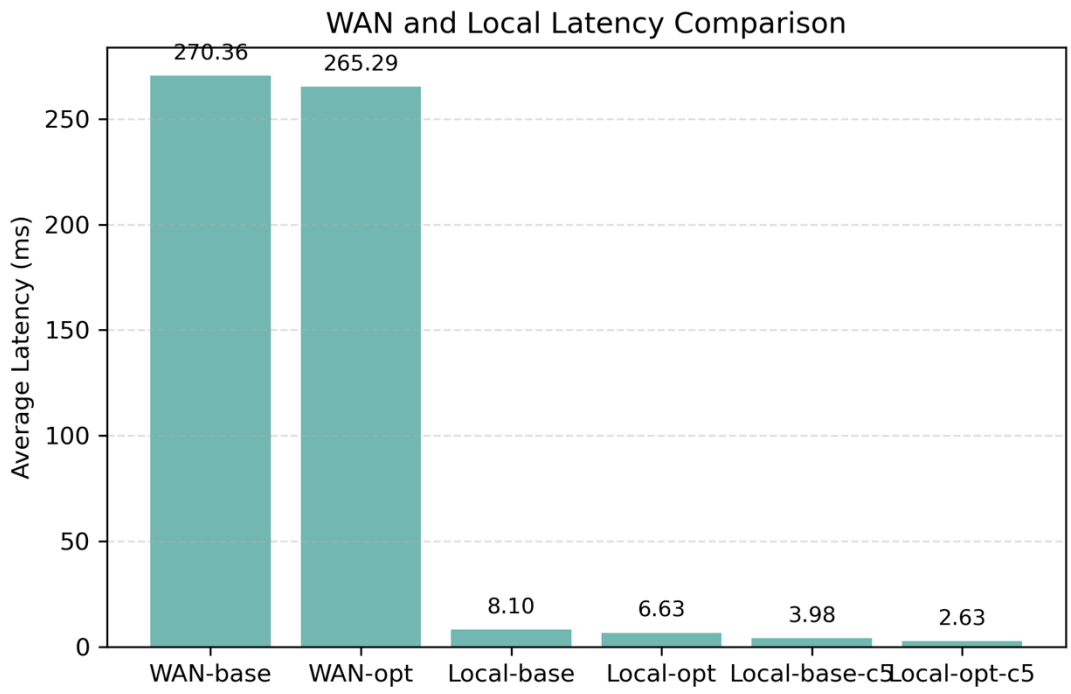
Table 4 reports the hit-only comparison in the local netns/veth environment. Compared with the baseline, the optimized version reduces average latency from 8.10 ms to 6.63 ms, a decrease of 18.15%. Table 5 further reports the low-concurrency case, where latency decreases from 3.98 ms to 2.63 ms. These results show that a shorter matching path can produce more visible end-to-end latency improvements in a controlled local environment. However, CPU usage is close to saturation in the local stress tests, so CPU utilization mainly reflects the generated load and should not be interpreted as an isolated optimization benefit.

Table 4. Local netns/veth hit-only comparison.

Version	avg_scan_steps	avg_cpu_usage_pct	avg_success_rate_pct	avg_latency_ms
Baseline	100	92.20	100	8.10
Optimized	1	92.18	100	6.63

Table 5. Low-concurrency local hit-only comparison.

Version	avg_scan_steps	avg_cpu_usage_pct	avg_success_rate_pct	avg_latency_ms
Baseline	100	91.74	100	3.98
Optimized	1	91.90	100	2.63

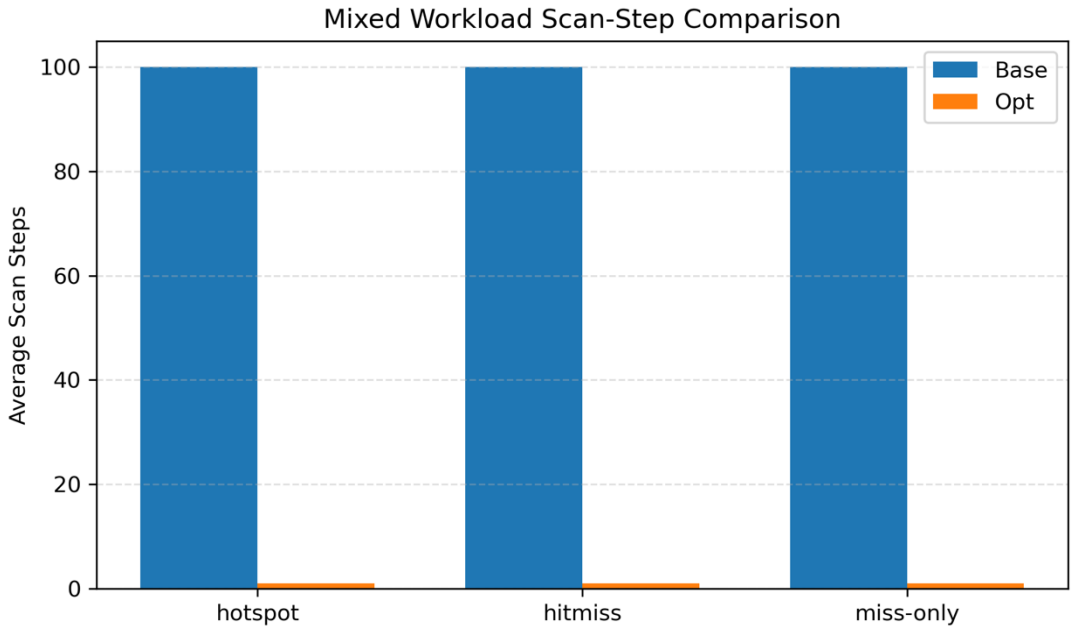


7.5. Mixed-workload experiment

Table 6 reports three mixed workloads. The optimized version reduces average scan steps to 1.00 in all three workload modes. The hotspot mode mainly sends hot-port requests, hitmiss mixes hit and miss requests, and miss-only sends requests to non-allowed ports. Differences in success rate come from the request composition and the failure definition in short-connection tests. Therefore, success rate is not used as the optimization benefit itself; it is used to confirm that each workload behaves as intended.

Table 6. Mixed-workload results.

Workload	Version	avg_scan_steps	avg_cpu_usage_pct	avg_success_rate_pct	avg_latency_ms
hotspot	Baseline	100	35.60	80.30	2.04
hotspot	Optimized	1	34.37	79.80	2.01
hitmiss	Baseline	100	16.81	50.37	1.84
hitmiss	Optimized	1	17.02	49.85	1.86
miss-only	Baseline	100	9.26	0	0
miss-only	Optimized	1	8.95	0	0

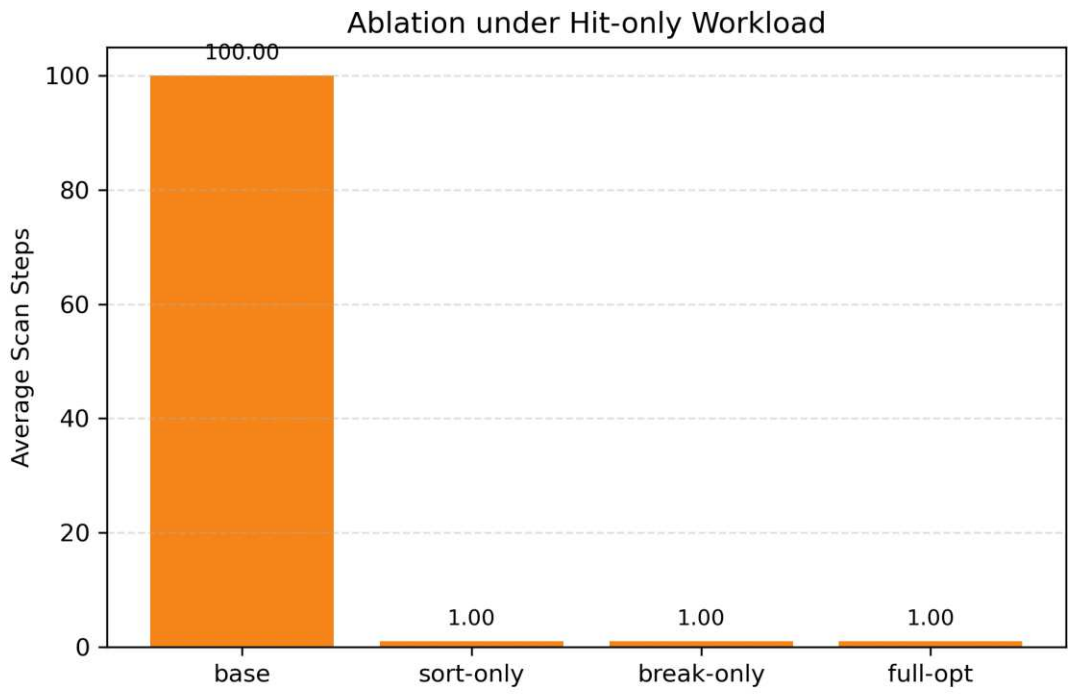


7.6. Ablation experiment

We first evaluate the four variants in the hit-only workload, as shown in Table 7. In this scenario, because the hot port is placed at a position that can be reached early under the tested layouts, sort-only, break-diagnostic, and full-opt all reduce scan steps to 1.00. This result only shows that ordering or early hit conditions affect scan depth; it does not prove that early termination without ordering is safe for deployment. The full-opt latency in Table 7 is higher than that of sort-only and break-diagnostic, but all three variants have the same average scan steps and the current data do not report variance or tail latency. We therefore do not interpret this difference as a stable performance regression.

Table 7. Ablation results under hit-only workload.

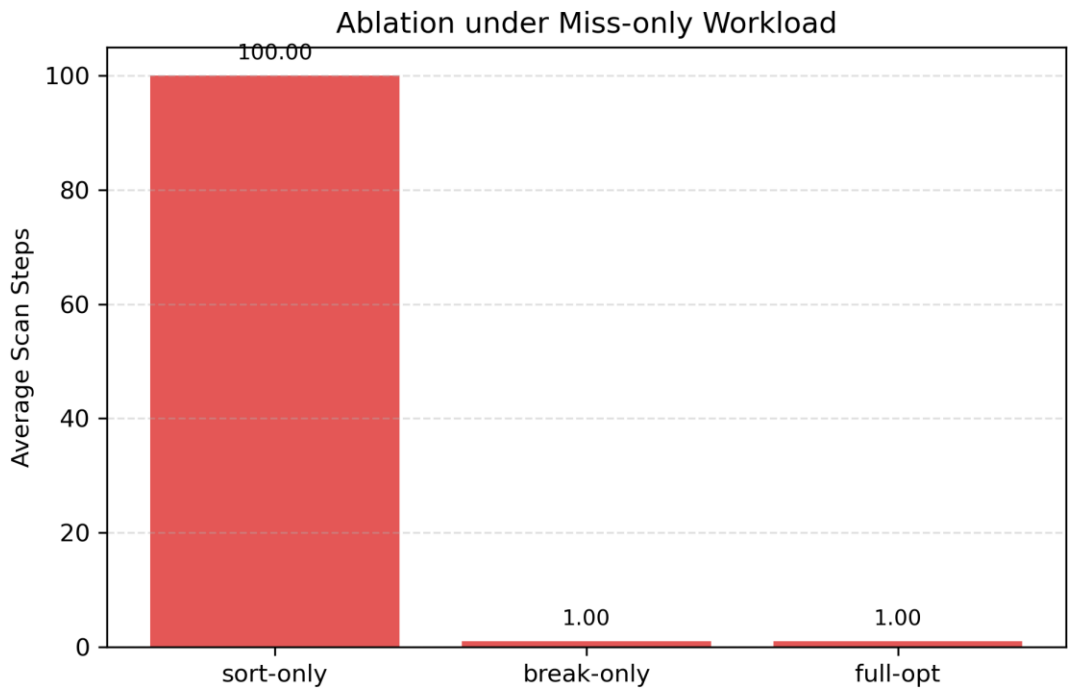
Version	avg_scan_steps	avg_cpu_usage_pct	avg_success_rate_pct	avg_latency_ms
Baseline	100	92.09	100	6.46
sort-only	1	92.16	100	6.74
break-diagnostic	1	92.11	100	7.00
full-opt	1	92.15	100	8.88



To further analyze the independent contribution of early termination, we compare sort-only, break-diagnostic, and full-opt under the miss-only workload, as shown in Table 8. Ordered insertion alone does not reduce scan steps for all miss cases. When the miss port is smaller than the minimum low_port and early termination is enabled, full-opt reduces scan steps to 1.00. The break-diagnostic result should be interpreted only as a diagnostic comparison, not as a deployable configuration, because early termination without an ordering guarantee may skip a later matching rule.

Table 8. Ablation results under miss-only workload.

Version	avg_scan_steps	avg_cpu_usage_pct	avg_success_rate_pct	avg_latency_ms
sort-only	100	8.51	0	0
break-diagnostic	1	9.10	0	0
full-opt	1	8.90	0	0



7.7. Reproducibility

To reduce measurement irreproducibility, rule installation, statistics reset, request generation, and result aggregation are scripted. Server-side scripts build different variants, initialize rule sets, and reset statistics. Local namespace scripts construct isolated client/server environments, generate traffic, and summarize CSV outputs. Public-cloud matrix scripts repeat tests across rule scales, hit positions, and build variants. Each round resets scan statistics, runs a fixed number of requests with fixed concurrency, and then reads total_steps, total_scans, and avg_scan_steps through the user-space statistics interface. Since scan steps are counted inside the eBPF hot path, they directly reflect the matching-path length for each rule and workload configuration.

7.8. Summary of results

The experimental results lead to three observations. First, the proposed method consistently reduces hot-port matching depth. Across hit positions, rule scales, and workload modes, the optimized version has much lower average scan steps than the baseline, and in multiple representative scenarios the value decreases from 100.00 to 1.00. Second, the optimization benefit appears differently across environments. For avg_scan_steps, the reduction reaches 96% or higher, and up to 99% in the core scenarios. In the public-cloud environment, however, average latency improves only by 1.88%. The local improvements of 18.15% and 33.92% indicate that public-link noise and external system factors can mask matching-path optimization in end-to-end measurements. Third, mixed-workload and ablation results show that the method is not limited to a single hot-hit case. Ordered insertion mainly improves hot-hit positions, while the difference between sort-only and full-opt in the miss-only workload shows the independent contribution of early termination for selected miss distributions.

8. Discussion and Limitations

8.1. Semantic boundary

The method relies on one key assumption: port-range rules under the same prefix, protocol, and direction are non-overlapping, or their conflicts have already been explicitly resolved by the policy layer. Under this assumption, maintaining the index in ascending low_port order does not change rule actions. If the rule set contains overlapping intervals, explicit priorities, or first-match semantics, simple sorting may change the result. Therefore, the method should not be described as a semantic-preserving reordering algorithm for arbitrary rule sets. It is a path optimization for non-overlapping or conflict-resolved rule sets.

8.2. Complexity boundary

Ordered insertion and early termination reduce average scan steps under selected traffic distributions. They do not eliminate linear scanning in all cases. If the destination port is near the end of the ordered index, or if a miss port is larger than all low_port values, matching may still scan O(N) entries. Therefore, the 1.00 scan-step results should be interpreted under the constructed hot-port and early-terminating miss scenarios, not as a general constant-complexity claim for arbitrary rule sets.

8.3. Experimental generalization

The current prototype is limited by MAX_INDEX_ENTRIES = 100 and has not evaluated larger single-prefix rule sets. The evaluation also does not include stronger external baselines such as nftables/ipset interval sets, manually optimal reordering, binary search, or more general packet-classification structures. The public-cloud experiment demonstrates deployability in a realistic path for a host-based eBPF network function, but its end-to-end latency is affected by link noise. These factors limit the generalization of the current results to larger rule sets and other network-function implementations.

8.4. Threats to validity

Internal validity is affected by public-link variation, cloud scheduling, short TCP connections, and mpstat sampling windows. We therefore prioritize avg_scan_steps when interpreting mechanism-level benefits. Construct validity is limited because avg_scan_steps measures the average number of scanned entries in the matching path and is not equivalent to end-to-end latency. External validity is limited by the ZFW single-prefix port-range index structure and the MAX_INDEX_ENTRIES = 100 bound. Reproducibility validity is mitigated by scripting build, rule generation, traffic generation, and statistics collection, but an anonymized patch, build commands, and raw CSV files should be provided as supplementary material when required.

9. Conclusion

This paper studied port-range rule management and matching-path measurement for an eBPF-based network function. We proposed two lightweight mechanisms: ordered insertion in the user-space rule-management path and early termination in the TC-BPF hot path. To quantitatively evaluate the optimization, we implemented scan-step statistics and ablation build switches, and constructed experiments covering hit position, rule scale, public-cloud deployment, local controlled environments, mixed workloads, and ablation variants.

The results show that, under the tested hot-port and selected miss scenarios, the optimization substantially reduces average scan steps. In hit-position and rule-scale experiments, the optimized version keeps average scan steps at 1.00. In the public-cloud core scenario, average scan steps decrease from 100.00 to 1.00. In local controlled experiments, average latency decreases by 18.15% and 33.92% in two settings. At the same time, public-cloud latency improvement is much smaller than scan-step improvement, showing that end-to-end performance is still affected by link and system noise. Therefore, the core conclusion is a reduction in managed matching-path complexity, not a uniform end-to-end latency improvement in all deployment environments.

Overall, the proposed method provides a low-cost optimization for eBPF-based network-function rule management while keeping the existing ZFW data structure largely unchanged. It is suitable for non-overlapping port intervals or rule sets whose conflicts have been explicitly handled. Future work includes hierarchical indexes for larger rule sets, dynamic hot-rule reordering, multi-field joint optimization, and more complete performance comparisons with existing Linux firewall and network-function management mechanisms.

Statements and Declarations

Funding

No specific funding was received for this work.

Competing Interests

The author declares that there are no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

Data will be made available on request.

Author Contributions

Bo Xie contributed to conceptualization, methodology, formal analysis, investigation, software, validation, visualization, writing of the original draft, and review and editing.

Acknowledgements

The author thanks the open-source eBPF community for publicly available tools and documentation.

References

[1] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in Proc. USENIX Winter 1993 Conference, San Diego, CA, USA, 1993, pp. 259-270.

[2] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, et al., "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in Proc. CoNEXT 2018, New York, NY, USA: ACM, 2018, pp. 54-66, doi: 10.1145/3281411.3281443.

[3] The Linux Kernel Documentation, "BPF Documentation." https://docs.kernel.org/bpf/ (accessed May 7, 2026).

[4] S. Y. Wang and J. C. Chang, "Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel," Journal of Network and Computer Applications, vol. 198, p. 103283, 2022, doi: 10.1016/j.jnca.2021.103283.

[5] The Netfilter.org Project, "Netfilter/iptables project homepage." https://www.netfilter.org/ (accessed May 7, 2026).

[6] The nftables Project, "Sets - nftables wiki." https://wiki.nftables.org/wiki-nftables/index.php/Sets (accessed May 7, 2026).

[7] The ipset Project, "ipset(8): administration tool for IP sets." https://manpages.debian.org/trixie/ipset/ipset.8.en.html (accessed May 7, 2026).

[8] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, pp. 147-160, 1999, doi: 10.1145/316194.316217.

[9] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," ACM Computing Surveys, vol. 37, no. 3, pp. 238-275, 2005, doi: 10.1145/1108956.1108958.

[10] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," in Proc. ACM SIGCOMM 2003, New York, NY, USA: ACM, 2003, pp. 213-224, doi: 10.1145/863955.863980.

[11] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "EffiCuts: Optimizing Packet Classification for Memory and Throughput," in Proc. ACM SIGCOMM 2010, New York, NY, USA: ACM, 2010, pp. 207-218, doi: 10.1145/1851275.1851208.

[12] D. E. Taylor and J. S. Turner, "ClassBench: A Packet Classification Benchmark," IEEE/ACM Transactions on Networking, vol. 15, no. 3, pp. 499-511, 2007, doi: 10.1109/TNET.2007.893156.

[13] E. Al-Shaer and H. Hamed, "Firewall Policy Advisor for Anomaly Discovery and Rule Editing," in Proc. IFIP/IEEE IM 2003, 2003, pp. 17-30.

[14] H. Hamed and E. Al-Shaer, "Dynamic Rule-ordering Optimization for High-speed Firewall Filtering," in Proc. ASIACCS 2006, New York, NY, USA: ACM, 2006, pp. 332-342, doi: 10.1145/1128817.1128867.

[15] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2015, doi: 10.1109/JPROC.2014.2371999.

[16] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 236-262, 2016, doi: 10.1109/COMST.2015.2477041.

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, et al., "P4: Programming Protocol-independent Packet Processors," ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 87-95, 2014, doi: 10.1145/2656877.2656890.

[18] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in Proc. IMC 2015, New York, NY, USA: ACM, 2015, pp. 275-287, doi: 10.1145/2815675.2815692.