Reducing Host Load Via Commodity NIC Offloading

ALIYA ALSAFA, Stanford University, USA
THOMAS HAILE, Stanford University, USA
NIKITA TYUNYAYEV, UCLouvain, Belgium
THEA ROSSMAN, Stanford University, USA
ZAKIR DURUMERIC, Stanford University, USA

Abstract

Traffic analysis systems must process packets at line rate, yet a large and growing fraction of network traffic is encrypted ciphertext that carries no information useful to analysis tools. These packets consume CPU cycles and PCIe bandwidth indistinguishably from traffic that does contain useful information, imposing unnecessary load on systems that are already strained by increasing network bandwidth and slowing per-core performance gains. Prior approaches to reducing host load through NIC offload rely on programmable datapaths, on-device processors, and stateful operations available only on specialized hardware.

We show that meaningful host load reduction is achievable using only widely available commodity NIC primitives, namely header-based packet filtering and configurable flow tables. Because encrypted traffic is concentrated in elephant flows — a small fraction of long-lived, high-throughput flows that account for the vast majority of bytes — our system can selectively install drop rules on just these flows to eliminate their ciphertext packets before they cross the PCIe bus, with no loss of analyzable information. On our network tap, targeting just 0.3% of flows covers 74% of total bytes. At most ~12,000 elephant flows are active at any moment, well within the rule capacity of commodity NICs such as the NVIDIA ConnectX-5 and Intel E810. As a proof of concept, dropping TLS, QUIC, and SSH traffic after handshake completion reduces host-visible bandwidth by 23% with zero unintentional packet loss. In preliminary evaluation, a classifier-guided rule installation strategy consistently achieves 60–70% bandwidth reduction in offline trace replay, outperforming random selection's 25% under the same rule budget, with classifier precision and recall stable in the 0.80–0.85 range over 36 hours of live traffic.

1 Introduction

As network bandwidth continues to scale [13], traffic analysis systems such as intrusion detection systems must process an ever-increasing volume of packets at line rate, placing substantial demand on CPU cycles and PCIe bandwidth [26, 29]. At the same time, the end of Dennard scaling has slowed per-core performance gains, making it increasingly difficult to rely on general-purpose CPUs to handle growing packet rates [8, 13]. Traffic analysis systems such as Zeek [3] and Iris [30] already exhibit degraded performance under high traffic loads [2, 3, 32], and this gap will only widen.

What makes this problem of expanding network bandwidth especially striking is that a large fraction of network traffic is encrypted ciphertext, which carries no information useful to analysis tools, yet consumes host resources indistinguishably from traffic that does [6, 23, 24]. Once a flow transitions to ciphertext, its packets are opaque to traffic analysis tools, yet they continue to consume CPU cycles and PCIe bandwidth at the same rate as any other packet [31].

A large body of prior work aims to reduce host load through NIC offload, pushing filtering, flow reassembly, and pattern matching onto the NIC [19, 21, 26, 28]. These approaches can substantially reduce the amount of bytes reaching the host, but they rely on rich NIC functionality — such as programmable datapaths, on-device processors, and stateful operations — that is only available on expensive and more advanced hardware [10, 13, 14, 36].

Authors' Contact Information: Aliya Alsafa, Stanford University, Stanford, California, USA, aalsafa@stanford.edu; Thomas Haile, Stanford University, Stanford, California, USA; Nikita Tyunyayev, UCLouvain, Louvain-la-Neuve, Belgium; Thea Rossman, Stanford University, Stanford, California, USA; Zakir Durumeric, Stanford University, Stanford, California, USA.

In this work, we address this problem of wasted host resources on unanalyzable ciphertext traffic by measuring what host resources can be saved using only a minimal set of NIC primitives: header-based packet filtering and configurable flow tables. Our goal is to reduce host-visible bandwidth without any loss of analyzable information.

To achieve this goal, our system installs packet drop rules on commodity NICs to filter encrypted traffic before it reaches the host. We target ciphertext traffic, since once a flow transitions to encrypted payload, its subsequent packets are useless to analysis tools yet continue to consume host resources. Once the TLS or QUIC handshake is completed, every subsequent packet in that flow is confirmed as ciphertext that has no information useful to analysis tools. By dropping these packets at the NIC, we eliminate unnecessary host load with zero loss of analyzable information, using only widely available hardware primitives.

Realizing this in practice requires overcoming three challenges. First, NIC flow tables are small, so the system cannot install rules for every flow. We address this by exploiting the well-documented existence of elephant flows on network traffic [12, 20], where a small fraction of flows accounts for the vast majority of bytes [11, 33]. We confirm this on our own network and use it to prioritize rule installation on the flows that yield the greatest resource savings per rule. Second, the system must decide when to evict rules and avoid blocking new flows when five-tuples are reused. We address this through a FIFO-based aging mechanism with a timeout tuned to observed flow turnover on our network. Third, the overhead of flow tracking and rule management must not itself exceed the resources saved. We address this by running the rule manager on dedicated cores off the packet capture path.

We evaluate our system on real network traffic, measuring its impact on host-visible bandwidth and rule installation overhead, as well as the cost of the rule management mechanism. We also characterize what fraction of ciphertext the system successfully drops. Our results show that significant resource savings are achievable using only widely available NIC functionality.

2 Background and Related Work

2.1 NIC Functionality and Offloading

Widely available, inexpensive commodity NICs support a core set of packet processing primitives, namely header-based filtering and configurable match-action flow tables [10, 13]. Incoming packets are matched against installed flow rules — entries that pair a header-based match condition with an action — based on header fields (e.g., source and destination IP addresses, ports, and protocol), and a corresponding action is applied, such as dropping the packet, forwarding it to a specific receive queue, or modifying a header field [10, 28].

Prior work exploits richer NIC functionality to offload computation from the host. SmartNICs and data processing units (DPUs) augment this baseline with general-purpose on-device processors, enabling computation at the device level [5, 8, 13, 14]. FPGAs provide programmable datapaths capable of stateful operations such as flow reassembly and arbitrary pattern matching [15, 28]. These capabilities have been used to offload tasks including deep packet inspection, intrusion detection pre-filtering, and application-layer parsing [8, 15, 26]. For example, systems targeting intrusion detection and prevention have offloaded deep packet inspection and TCP reassembly to FPGA-capable SmartNICs [24, 31, 35], while stateful pre-filtering of reassembled flows has been implemented on FPGAs and SmartNICs [16, 28, 34]. TLS pre-filtering has similarly been implemented in FPGA to accelerate IDS pipelines by stripping encrypted payload packets before they reach the host [15].

In contrast to this prior work, we aim to achieve similar reductions in host-visible bytes using only widely accessible NIC primitives [15, 25, 32]. We rely exclusively on the header filtering and

flow tables, and as we discuss in the following sections, this minimal functionality is sufficient to achieve meaningful reductions in bytes going to the host for traffic analysis workloads.

2.2 Flow Table Architecture and Behavior

We focus on commodity NICs that support a match-action flow table, specifically the NVIDIA Mellanox ConnectX-5 and Intel E810. Incoming packets are matched against installed flow rules, typically based on header fields, and a corresponding action is applied [1, 10]. Supported actions typically include dropping packets, forwarding to specific receive queues, and modifying header fields [10]. While this basic model is consistent across vendors, the organization, capacity, and performance characteristics of flow tables vary significantly by device [10, 18, 36].

On the NVIDIA Mellanox ConnectX-5, flow tables are organized into a multi-table pipeline [10]. A root table (Table 0) serves as the entry point for all packets, while non-root tables (Tables 1–16) can be used to implement staged processing logic. Packets may traverse multiple tables depending on match results, enabling flexible classification and action chaining. However, this flexibility comes with performance trade-offs: prior work has shown that distributing rules across multiple tables can significantly degrade throughput compared to concentrating rules in a single table [10].

Flow table capacity on the ConnectX-5 is also constrained by on-device memory [10, 27]. While a single non-root table can sustain line-rate throughput with a large number of entries, the total number of rules that can be installed is bounded [10]. Rule installation incurs non-trivial latency that varies by table, and newly installed rules may not take effect immediately. These constraints directly inform our design decisions, which we discuss in Section 4.

2.3 Elephant Flows and Heavy Hitters

Prior work has shown that on most networks a small fraction of flows, commonly referred to as *elephant flows* or *heavy hitters*, account for the majority of transmitted bytes [12, 20]. Recent analyses of campus and datacenter networks suggest that as few as 0.2–0.4% of flows may account for up to 80% of total traffic [11, 12]. On our own network, we see a very similar result, where 0.3% of flows accounts for 74% of bytes, as we show in Section 3. This skew is important because it identifies a small subset of flows that are suitable candidates for NIC offloading, which is necessary given hardware constraints. As we discuss in Section 2.2, we leverage this insight to maximize dropped packets while minimizing installed flow rules.

Identifying heavy flows quickly allows them to be offloaded or dropped before they consume significant resources. This has motivated prior work on heavy-hitter detection under partial information, where decisions must be made using only the first few packets of a flow [11, 33]. For example, prior work has proposed sketch-based multi-stage filtering techniques that progressively eliminate low-frequency flows (e.g., Sequential Zeroing [7]), NIC/FPGA-based hierarchical detection pipelines [11, 17], and machine learning classifiers that use 5-tuple and transport-layer features to predict heavy flows from the first packet, significantly reducing flow processing and table usage while covering most traffic [9, 22].

We opt for a machine learning classifier to predict heavy flows early on; see Section 4.

3 Motivation

3.1 Targeting Elephant Flows

While commodity NICs can support header-based filtering and flow table rules, rule capacity and installation overhead are non-trivial constraints [10, 14, 36]. In this section, we characterize traffic on our local network tap to show that these constraints are not a barrier in practice. We establish

that a small, identifiable subset of flows accounts for the vast majority of bytes, and the number of rules needed to target them is well within hardware limits.

Once a flow transitions to ciphertext, its subsequent packets are opaque to traffic analysis tools. No further inspection is possible, and no analyzable information remains [31]. Yet these packets continue to consume PCIe bandwidth and CPU cycles at the same rate as any other packet. After reconstructing the handshake, we can determine that every post-handshake packet categorized as an encrypted protocol is a candidate for dropping at the NIC with no loss of analyzable information.

We cannot, however, install a drop rule for every encrypted flow naively, as it is not feasible in practice. NIC flow tables are small, and as we show in Section 2.2, rule installation incurs non-trivial latency [10]. Installing rules indiscriminately for short-lived, low-volume flows would exhaust the rule budget while recovering negligible host resources, and the overhead of tracking and managing those rules could itself exceed the savings. Furthermore, managing a large number of flow rules introduces its own overhead: each rule requires tracking, installation, and eventual expiration, all of which consume host resources. The system must therefore be selective, targeting only the flows where the savings justify the cost of a rule.

3.2 Traffic Characterization

Figure 1 shows the distribution of flow duration and total payload bytes on our network tap. Traffic is strongly bimodal: the vast majority of flows are short-lived and low-throughput, while a small cluster of long-lived, high-throughput flows accounts for a disproportionate share of bytes. Concretely, targeting just 0.3% of flows would cover 74% of total bytes, using a threshold of >1 Mbps and >20 s to define elephant flows.

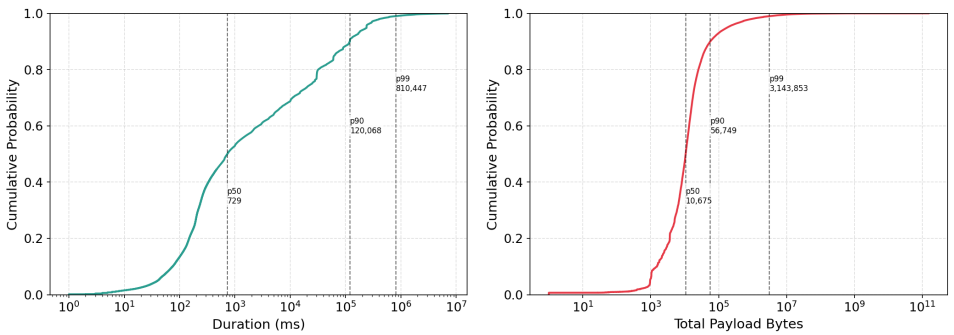

Fig. 1. CDF of flow duration (left) and total payload bytes (right) across all flows on our local network. The vast majority of flows are short-lived and low-volume; however, the distributions have heavy tails. This bimodal character motivates selective rule installation: installing a drop rule for every flow would waste NIC rule budget on flows that contribute negligibly to host load, while targeting only the long-lived, high-volume tail captures the bulk of bytes with a small number of rules.

Several properties of these flows make them well-suited for NIC offloading. First, the majority of elephant flows on our tap are TCP/TLS or QUIC, accounting for a disproportionate share of elephant flow bytes. Since TLS and QUIC ciphertext onset is detectable from packet headers and the reconstructed handshake [15], every packet after the handshake can be safely dropped with no loss of analyzable information. Second, 88.7% of elephant flows are download-dominated, with more than 75% of bytes traveling in one direction, meaning a single drop rule per flow is sufficient to eliminate the bulk of the associated host load with no reverse-direction rule needed. Third, Figure 2 shows that at any given moment, only approximately 10,000–12,000 elephant flows are active

simultaneously, which is well within the rule capacity of commodity NICs such as the NVIDIA ConnectX-5 and Intel E810, as we characterize in Section 2.2.

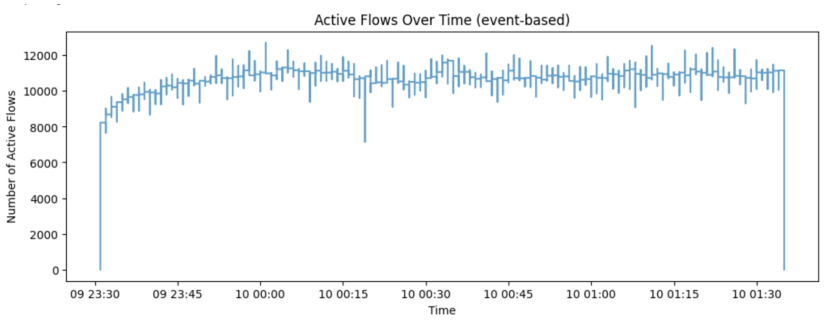

Fig. 2. Number of active elephant flows over time, sampled in two-minute intervals. The active count remains in the range of 10,000–12,000.

3.3 Detecting Elephant Flows Early

The remaining challenge in realizing this in practice is identifying elephant flows early before the bulk of their bytes have already crossed the PCIe bus. Simple header-based heuristics are insufficient for this. Port number alone does not discriminate: while most elephant flows use destination port 443, the majority of port-443 flows are short-lived and low-throughput (Figure 3), so a blanket rule would exhaust the rule budget on flows that contribute negligibly to host load. Burstiness of inter-arrival times in packets similarly fails — its distribution is nearly identical across flow categories regardless of duration.

We therefore use a random forest classifier that predicts whether a flow will become an elephant flow from its early packets, using only features observable in packet headers and the TLS handshake. If we can accurately identify these flows early, we can target 0.3% of flows to cover 74% of bytes, with at most ~12,000 rules active at any time [9, 22]. We describe the classifier and the full system design in Section 4.

4 Design and Implementation

4.1 System Overview

Our system prototype is implemented on top of Iris [30], a high-performance traffic analysis framework built on DPDK [1]. At a high level, the pipeline has four stages. First, Iris captures packets at line rate and tracks per-flow state. Second, a TLS and QUIC handshake detector identifies when a flow transitions to ciphertext. Third, a rule decision module determines whether the flow qualifies for a drop rule. Finally, a rule manager asynchronously installs and expires rules on the NIC. A flow triggers a rule insertion only when it satisfies both conditions: it has been classified as an elephant flow, and its TLS or QUIC handshake has completed. Flows that satisfy both criteria receive a drop rule; all other packets continue to reach the host as normal.

The rule manager runs on dedicated worker cores, off the RX path, so that rule insertion calls do not stall the packet capture pipeline. The NIC handles matching and dropping entirely in hardware, eliminating the PCIe transfer and CPU cost of dropped packets with no CPU involvement after rule installation [4].

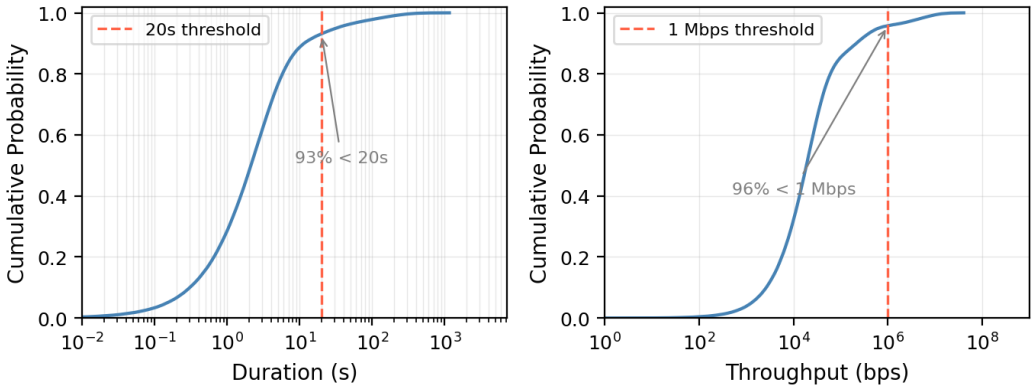

Fig. 3. CDF of flow duration (left) and throughput (right) for port-443 flows. Despite being the dominant elephant flow port, the vast majority of port-443 flows are short-lived and low-throughput — well below the 20 s and 1 Mbps elephant flow thresholds. Naively installing drop rules for all port-443 traffic would exhaust the rule budget on flows that contribute negligibly to host load.

4.2 Identifying Encrypted Elephant Flows

A flow becomes a candidate for a drop rule when two conditions are jointly satisfied: it has completed its TLS or QUIC handshake, and it has been predicted to be an elephant flow.

4.2.1 TLS and QUIC Handshake Detection. The system monitors the sequence of TLS record types in each flow's packets to detect handshake completion. Once the handshake concludes, every subsequent packet carries encrypted application data that is opaque to traffic analysis tools [31] and can be dropped with no loss of analyzable information. The transition is identifiable by inspecting the reconstructed handshake from packet payloads [15].

4.2.2 Elephant Flow Classification. Detecting handshake completion alone is not sufficient to justify installing a drop rule. As we show in Section 3, the majority of TLS flows are short-lived and low-volume; installing rules for all of them would exhaust the rule budget while recovering negligible host resources. The system therefore uses a machine learning classifier to predict, from the first few packets of a flow, whether it will become an elephant flow before committing a rule table entry.

The classifier operates on features derived from packet headers, including inter-arrival times, packet sizes, byte volumes, TCP flags, source and destination IP subnet, and fields from the TLS ClientHello and ServerHello messages observable during the handshake [9, 22]. It produces a binary prediction early enough that a rule can be installed before the bulk of the flow's bytes arrive at the host. A false positive wastes one rule table entry on a flow that turns out to be small; a false negative allows a large encrypted flow to continue reaching the host. The impact of classifier accuracy on end-to-end resource savings is evaluated in Section 5.

4.3 Rule Installation

Once the classifier predicts an elephant flow and the TLS or QUIC handshake is complete, the rule manager installs a 5-tuple drop rule on the NIC covering the dominant direction of the connection. As established in Section 3, 88.7% of elephant flows are download-dominated, meaning a single unidirectional rule eliminates the bulk of the associated host load without consuming a second rule table entry for the reverse direction.

Rule installation is dispatched asynchronously to a dedicated worker core. Because NIC rule installation incurs non-trivial latency and newly installed rules may not take effect immediately [10], the system observes a short grace period before confirming that packets from the flow are no longer reaching the host. Packets that arrive during this window are processed normally; we measure the magnitude of this residual load in Section 5.

4.4 Rule Management and Aging

The total number of active rules is bounded by the NIC's flow table capacity, which we characterize empirically in Section 5. To operate within this budget, the rule manager maintains a FIFO expiration queue. When the budget is exhausted and a new rule must be installed, the oldest installed rule is evicted. This policy is motivated by the flow turnover we observe on our network: active elephant flows turn over continuously, and flows that have been in the table longest are most likely to have already terminated. A timeout of approximately two minutes is consistent with the flow duration distribution characterized in Section 3, where the 90th percentile elephant flow lasts under 120 seconds.

The key constraint on the design is that the overhead of flow tracking, classification, rule installation, and expiration must not itself consume more host resources than the drop rules save [4]. We address this in three ways: the rule manager runs on dedicated cores off the RX path; classification only runs on flows that have completed a TLS or QUIC handshake, substantially reducing the number of flows that require evaluation; and rule lookups use an efficient hash-based data structure to minimize per-flow overhead. We evaluate this tradeoff directly in Section 5.

5 Evaluation

We present initial evaluations of our prototype system, characterizing the NIC hardware constraints that motivate our design, the feasibility of dropping ciphertext traffic at the NIC, and the benefit of using a classifier to target rule installation. These results are intended as a proof of concept; end-to-end system evaluation on live traffic at line rate is described as future work in Section 5.3.3.

Specifically, we seek to answer three questions: (1) What are the rule budget and insertion speed constraints of commodity NICs, and do they accommodate our target workload? (2) Does dropping encrypted traffic after handshake completion meaningfully reduce host-visible bandwidth with no loss of analyzable information? (3) Does classifier-guided rule installation outperform a naive baseline under the same rule budget?

5.1 NIC Characterization

5.1.1 Rule Insertion Capabilities. We first benchmark two widely available commodity NICs — the NVIDIA ConnectX-5 and the Intel E810 — to characterize the rule budgets and insertion speeds our system must operate within [10]. We push random 5-tuple drop rules and generate traffic that does not match any of them, with the packet generator producing 64-byte packets at 20 MPPS. The goal is to measure forwarding performance as a function of installed rule count.

Figure 4 summarizes our results. The ConnectX-5 sustains line-rate forwarding with up to 1 million installed rules, while the E810 supports a maximum of 26,236 5-tuple rules before forwarding degrades. Insertion speed on the ConnectX-5 exceeds 256K rules per second, compared to a cap of 4K rules per second on the E810 — a more than 64× difference. We note that the E810 supports more rules when matching on fewer fields (e.g., source IP only), but 5-tuple matching is required for per-connection drop semantics. Despite lower capacity and insertion speed, the E810 shows no packet loss during rule installation, confirming it is a viable deployment target.

Importantly, both NICs comfortably accommodate our target workload. As established in Section 3, at most ∼12,000 elephant flows are active simultaneously on our network tap — well within

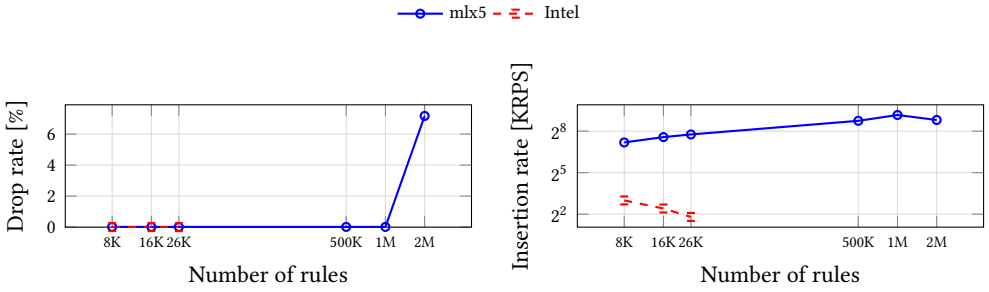

Fig. 4. Rule capacity and insertion speed for the ConnectX-5 and E810. The ConnectX-5 sustains line-rate forwarding with up to 1M rules installed, while the E810 supports at most 26K 5-tuple rules. Insertion speed on the ConnectX-5 exceeds 256K rules/s, more than 64× faster than the E810's cap of 4K rules/s.

the E810's 26K limit and far below the ConnectX-5's 1M capacity. This confirms that commodity NIC rule budgets are not a barrier to our approach.

5.1.2 Rule Insertion Latency. Rule installation latency is non-uniform across both the rule index and the table used. Figure 5 shows the per-rule insertion time when inserting 1 million rules sequentially into an initially empty table. Two patterns are visible: large latency spikes at rule indices 96, 384, 1536, 6144, 24576, 98304, and 393216, which we attribute to internal table resizing events (the table doubles in capacity when full); and smaller periodic spikes around every 20K rules from internal table management overhead.

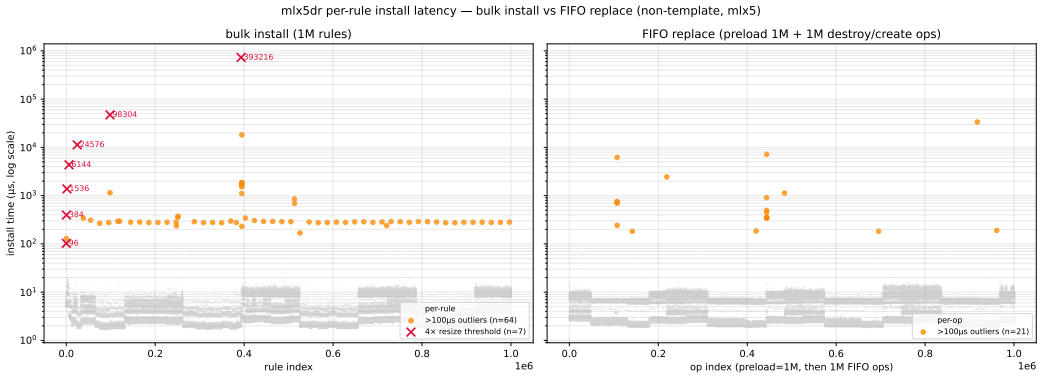

Fig. 5. Per-rule insertion latency for 1 million sequential rule insertions (left) versus a FIFO replace strategy on a pre-populated table (right). Pre-populating eliminates resizing spikes and substantially reduces tail latency.

To avoid these spikes in operation, we pre-populate the flow table to its target capacity before the system starts. At runtime, each rule installation is a FIFO replace: delete the oldest rule and insert the new one. The right panel of Figure 5 shows that this strategy eliminates both the resizing spikes and the periodic management overhead, with far fewer outliers. This motivates the FIFO aging design described in Section 4.

Figure 6 shows that latency also varies by table. We install drop rules in non-root tables (Tables 2 and 3), which incur consistently near-zero latency. The root table (Table 0) incurs significantly

higher installation latency, with large initial spikes that stabilize after the first ~20 insertions, which is why we avoid it for drop rules [10].

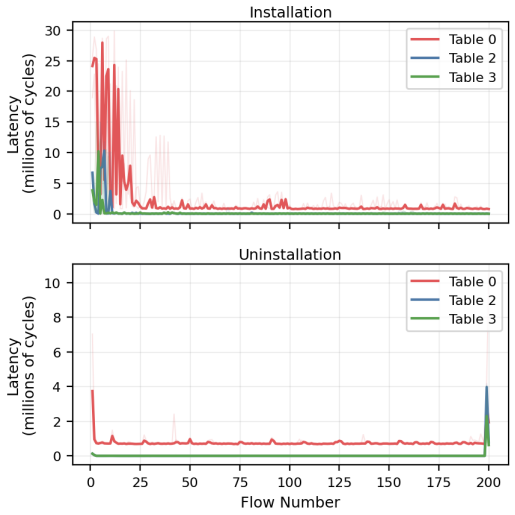

Fig. 6. Rule installation (top) and uninstallation (bottom) latency across Tables 0, 2, and 3 on the NVIDIA ConnectX-5, measured in millions of CPU cycles. Table 0 (root) incurs substantially higher and more variable latency than non-root tables, motivating our use of non-root tables for drop rules.

5.2 Preliminary End-to-End Results

To answer our second question of whether dropping encrypted traffic after handshake completion meaningfully reduces host-visible bandwidth, as well as to establish proof of concept, we run the system on live traffic from our network tap and measure the reduction in bytes reaching the host when drop rules are active, verifying that savings come with no loss of analyzable information. We compare two rule strategies: dropping all TCP/UDP flows after the first 10 packets, and dropping TLS, QUIC, and SSH flows after handshake completion. In both cases, the system runs on the same hardware with no changes to the Iris analysis pipeline.

Table 1 reports ingress and host-visible throughput for each strategy. Dropping TLS/QUIC/SSH after handshake completion reduces host-visible traffic from 140.2 Gbps to 107.8 Gbps, eliminating 23% of ingress bandwidth with zero unintentional packet loss. The more aggressive TCP/UDP strategy reduces host-visible traffic from 157.1 Gbps to 83.6 Gbps (47% reduction), at the cost of dropping analyzable pre-handshake traffic. Our target strategy — TLS/QUIC/SSH post-handshake — achieves meaningful savings while preserving all information that analysis tools can use.

5.3 Classifier Evaluation

5.3.1 Impact of Classifier on Bandwidth Reduction. A key design question is whether a classifier-guided rule installation strategy outperforms a simpler FIFO baseline that installs rules for whichever flows happen to be active, without regard for predicted size. The bandwidth reduction results reported here are emulated; we simulate rule installation decisions over captured traffic traces rather than running the classifier inline on live traffic. Prior work has demonstrated that TLS-aware inline filtering is feasible at line rate [15, 32]. Figure 7 compares bandwidth reduction over

Table 1. Preliminary end-to-end results. Dropping TLS/QUIC/SSH traffic after handshake completion, until the flow table budget is exhausted, reduces host-visible bandwidth by 23% with no unintentional packet loss — though this figure is well below the theoretical maximum achievable by correctly identifying and targeting elephant flows.

Strategy	Ingress	Host-visible	Packet Loss
Drop TCP/UDP after 10 pkts	157.1 Gbps	83.6 Gbps	0%
Drop TLS/QUIC/SSH post-handshake	140.2 Gbps	107.8 Gbps	0%

approximately two hours of offline trace replay under both strategies, subject to the same rule budget; these results represent a theoretical upper bound on achievable reduction.

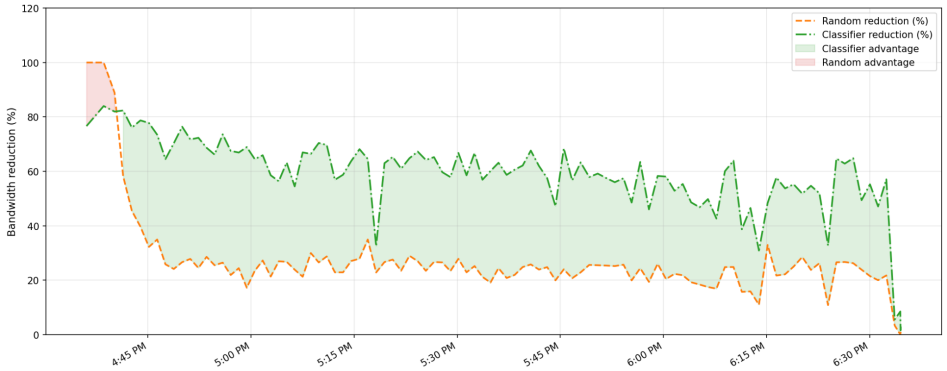

Fig. 7. Bandwidth reduction over time comparing random rule installation (orange) and classifier-guided installation (green), subject to the same rule budget. The classifier consistently achieves 60–70% reduction versus ~25% for random selection, demonstrating that rule budget is most effectively spent on flows the classifier identifies as elephant flows. These results are derived from offline trace replay and represent a theoretical upper bound on classifier-guided bandwidth reduction.

The classifier-guided strategy consistently achieves 60–70% bandwidth reduction throughout the experiment, while random installation achieves approximately 25%. The gap between the two — shown as the green shaded region — represents the benefit of selective rule installation. This confirms that the rule budget is a binding constraint, and that spending it on classifier-predicted elephant flows rather than arbitrary flows yields substantially greater resource savings.

5.3.2 Classifier Features and Accuracy. The classifier uses permutation importance to rank features by their contribution to prediction accuracy. Figure 8 shows the top features by permutation importance. The most informative feature is `client_sni_len`, the length of the Server Name Indication field in the TLS ClientHello, at approximately 4.7% importance. Source and destination IP subnet (`src_ip_subn`, `dst_ip_subn`) and responder inter-arrival time statistics (`resp_iat_mean`, `resp_iat_std`, `resp_iat_median`) follow, alongside originator payload bytes and several other TLS handshake fields. All features are derived from packet headers and the reconstructed TLS handshake, requiring very minimal payload inspection.

Figure 9 shows classifier precision and recall over approximately 36 hours of live traffic, evaluated against three definitions of elephant flow: by total payload bytes, duration, and packet count. Across all three definitions, both precision and recall remain stable in the range of 0.80–0.85 throughout

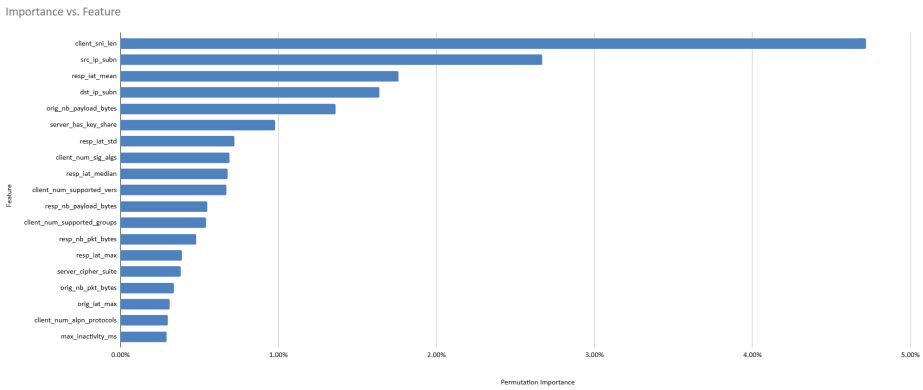

Fig. 8. Top 20 features by permutation importance for the elephant flow classifier. `client_sni_len` is the dominant predictor, followed by network-layer subnet features and responder inter-arrival time statistics. All features are observable from packet headers and the TLS handshake with minimal payload inspection.

the evaluation period, with no significant degradation over time. The duration-based definition shows somewhat higher variance in precision, with occasional dips to 0.75, but remains well above the 0.5 baseline. This stability across definitions and over time suggests the classifier generalizes well to unseen traffic.

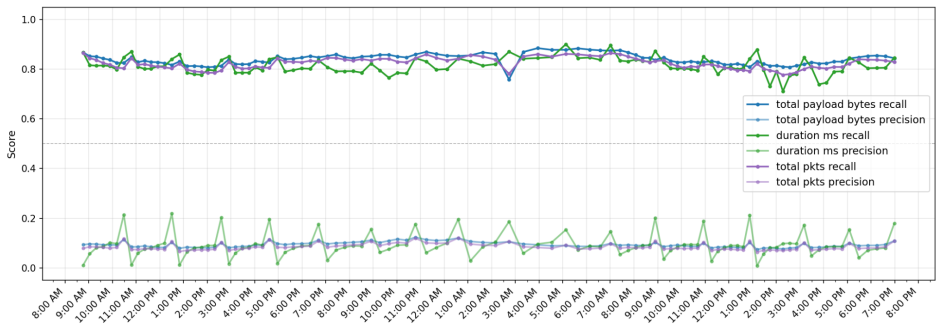

Fig. 9. Classifier precision and recall over 36 hours of live traffic, evaluated against three elephant flow definitions (total payload bytes, duration, packet count). Precision and recall are stable in the 0.80–0.85 range across all definitions, with no significant degradation over time. The periodic spikes occurring approximately every two hours correspond to classifier retraining intervals.

In the context of our system, a false positive wastes one rule table entry on a flow that turns out to be small; a false negative allows a large encrypted flow to continue reaching the host. At 0.80–0.85 precision and recall, the classifier strikes a reasonable balance: the rule budget is spent predominantly on flows that deliver large savings, and the majority of elephant flows are correctly identified and offloaded.

5.3.3 Next Steps. These results constitute an initial proof of concept. Several evaluations remain for future work. First, the classifier has been evaluated offline and in controlled experiments; we have not yet measured its overhead at line rate on live traffic. We nonetheless expect this to be

feasible: prior work has demonstrated that TLS-aware inline filtering is feasible at line rate [15, 32], and our system need only invoke the classifier on a small subset of connections — those completing a TLS, QUIC, or SSH handshake. Second, the end-to-end system — including the full pipeline from flow tracking through rule installation and expiration — has not been benchmarked for CPU and PCIe utilization under sustained load. Third, a longitudinal evaluation over days rather than hours would characterize how well the FIFO aging mechanism interacts with real flow turnover. We are actively pursuing these evaluations and are encouraged by the preliminary results presented here.

6 Conclusion

Traffic analysis systems face a growing gap between network bandwidth and host processing capacity. This gap is made worse by the fact that an increasing fraction of traffic is encrypted ciphertext — carrying no information useful to analysis tools, yet consuming host resources at the same rate as any other packet. We showed that meaningful host load reduction is achievable using only the primitives found on commodity NICs: header-based packet filtering and configurable flow tables. Our system exploits the well-documented heavy-tailed structure of real traffic — targeting just 0.3% of flows on our network to cover 74% of total bytes — to selectively install drop rules for the encrypted traffic that matters most. Even naively dropping all TLS, QUIC, and SSH traffic after handshake completion reduces host-visible bandwidth by 23% with zero unintentional packet loss, a promising result that required no per-flow intelligence whatsoever. A classifier-guided strategy pushes this to 60–70% in offline trace replay, outperforming random selection's 25% under the same rule budget. These results demonstrate that significant, lossless reductions in host-visible traffic are achievable without any specialized NIC hardware, making this approach broadly deployable across the commodity infrastructure already in use at network taps and monitoring points.

References

[1] 2023. DPDK: Data Plane Development Kit. https://www.dpdk.org/.

[2] 2023. Suricata Open Source IDS/IPS. https://www.suricata.io/.

[3] 2023. Zeek Network Security Monitor. https://zeek.org/.

[4] Luca Deri, Alfredo Cardigliano, and Francesco Fusco. 2024. Advancements in Traffic Processing Using Programmable Hardware Flow Offload. *arXiv preprint* (2024). arXiv:2407.16231.

[5] Tristan Döring, Henning Stubbe, and Kilian Holzinger. 2021. SmartNICs: Current Trends in Research and Industry. In *Seminar IITM WS 20/21, Network Architectures and Services*. TU Munich. doi:10.2313/NET-2021-05-1_05

[6] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J. Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception. In *Network and Distributed System Security Symposium (NDSS)*. doi:10.14722/ndss.2017.23456

[7] Abdessalam Elhaddad et al. 2017. High Speed Elephant Flow Detection Under Partial Information. *arXiv preprint* (2017). arXiv:1701.01683.

[8] Sergio Elizalde, Ali AlSabeh, Ali Mazloum, Samia Choueiri, Elie Kfoury, Jose Gomez, and Jorge Crichigno. 2025. A survey on security applications with SmartNICs: Taxonomy, implementations, challenges, and future trends. *Journal of Network and Computer Applications* 242 (2025), 104257. doi:10.1016/j.jnca.2025.104257

[9] Rohan Garg, Theophilus Benson, and George Varghese. 2017. Detecting Heavy Flows in the SDN Match and Action Model. *arXiv preprint* (2017). arXiv:1702.08037.

[10] Gerald Q. Maguire Jr. 2021. What You Need to Know About (Smart) Network Interface Cards. In *Passive and Active Measurement (PAM)*. Springer, 319–336. doi:10.1007/978-3-030-72582-2_19

[11] Pedro R. Torres Jr., Alberto García-Martínez, Marcelo Bagnulo, and Eduardo Parente Ribeiro. 2021. An Elephant in the Room: Using Sampling for Detecting Heavy-Hitters in Programmable Switches. *IEEE Access* 9 (2021). doi:10.1109/ACCESS.2021.3092281

[12] Piotr Jurkiewicz, Grzegorz Rzym, and Piotr Borylło. 2021. Flow Length and Size Distributions in Campus Internet Traffic. *Computer Networks* 189 (2021), 107836. doi:10.1016/j.comnet.2021.107836

[13] Elie F. Kfoury, Samia Choueiri, Ali Mazloum, Ali AlSabeh, Jose Gomez, and Jorge Crichigno. 2024. A Comprehensive Survey on SmartNICs: Architectures, Development Models, Applications, and Research Directions. *IEEE Access* 12 (2024), 107297–107336. doi:10.1109/ACCESS.2024.3437203

[14] Mikhail Khalilov, Marcin Chrapek, Siyuan Shen, Alessandro Vezzu, Thomas Benz, Salvatore Di Girolamo, Timo Schneider, Daniele De Sensi, Luca Benini, and Torsten Hoefler. 2024. OSMOSIS: Enabling Multi-Tenancy in Datacenter SmartNICs. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, 247–263.

[15] Vlastimil Košař, Lukáš Šišmiš, Jiří Matoušek, and Jan Kořenek. 2023. Accelerating IDS Using TLS Pre-Filter in FPGA. In *2023 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 436–442. doi:10.1109/ISCC58397.2023.10218049

[16] Sheng Lan, Ying Li, Zhongxian Liang, Wenjun Li, Yao Xin, Ying Wan, Hui Li, and Weizhe Zhang. 2026. MegaTurbo: A Scalable FPGA-based Engine for MegaFlow Classifier in Open vSwitch. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '26)*. ACM, 288–299. doi:10.1145/3748173.3779197

[17] Ângelo Cardoso Lapolli, João Artur Becker Lapolli, Leandro Bertholdo, and Lisandro Zambenedetti Granville. 2020. Identifying Elephant Flows Using Dynamic Thresholds in Programmable IXP Networks. *Journal of Internet Services and Applications* 11, 1 (2020). doi:10.1186/s13174-020-00131-6

[18] Jiaxin Lin, Zhiyuan Guo, Mihir Shah, Tao Ji, Yiying Zhang, Daehyeok Kim, and Aditya Akella. 2025. Enabling Portable and High-Performance SmartNIC Programs with Alkali. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association.

[19] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 243–259.

[20] Péter Megyesi and Sándor Molnár. 2013. Analysis of Elephant Users in Broadband Network Traffic. In *19th Open European Summer School (EUNICE)*. 37–45. doi:10.1007/978-3-642-40552-5_4

[21] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. AccelTCP: Accelerating Network Applications with Stateful TCP Offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, 77–92.

[22] Masoud Moshref, Minlan Yu, and Ramesh Govindan. 2017. Optimal Elephant Flow Detection. *arXiv preprint* (2017). arXiv:1701.04021.

[23] Szilveszter Nadas et al. 2018. *Challenges in Network Management of Encrypted Traffic.* Technical Report. EU H2020 MAMI Project. arXiv:1810.09272.

[24] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez, and Peter Steenkiste. 2015. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 199–212. doi:10.1145/2785956.2787482

[25] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 117–130.

[26] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. Floem: A Programming System for NIC-Accelerated Network Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, 663–679.

[27] Boris Pismenny, Liran Liss, Adam Morrison, and Dan Tsafrir. 2022. The Benefits of General-Purpose On-NIC Memory. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*. ACM, 1130–1147.

[28] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani Brunella, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Bianchi. 2019. FlowBlaze: Stateful Packet Processing in Hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, 531–548.

[29] Luigi Rizzo. 2012. netmap: A Novel Framework for Fast Packet I/O. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, 101–112.

[30] Thea Rossman, Diana Qing, Gerry Wan, and Zakir Durumeric. 2026. Iris: Expressive Traffic Analysis for the Modern Internet. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association.

[31] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 213–226. doi:10.1145/2785956.2787502

[32] Lukáš Šišmiš and Jan Kořenek. 2023. Analysis of TLS Prefiltering for IDS Acceleration. In *Passive and Active Measurement (PAM 2023) (Lecture Notes in Computer Science, Vol. 13882)*. Springer, 85–109. doi:10.1007/978-3-031-28486-1_5

[33] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Symposium on SDN Research (SOSR '17)*. ACM. doi:10.1145/3050220.3050239

[34] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Songyuan Sui, Khalid Manaa, Omer Shabtai, Yonatan Piasetzky, Matty Kadosh, Arvind Krishnamurthy, T. S. Eugene Ng, and Ang Chen. 2023. Unleashing SmartNIC Packet Processing

Performance in P4. In *ACM SIGCOMM 2023 Conference*. ACM, 1028–1042. doi:10.1145/3603269.3604882

[35] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C. Hoe, Vyas Sekar, and Justine Sherry. 2020. Achieving 100Gbps Intrusion Prevention on a Single Server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 1083–1100.

[36] Annus Zulfiqar, Ali Imran, Venkat Kunaparaju, Ben Pfaff, Gianni Antichi, and Muhammad Shahbaz. 2025. Gigaflow: Pipeline-Aware Sub-Traversal Caching for Modern SmartNICs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25)*. ACM. doi:10.1145/3676641.3716000