Decentralized Traffic Correlation Using Programmable Switches

by

Gurjot Singh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Gurjot Singh 2026

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

A substantial portion of this thesis is based on my previously published conference paper [54] (G. Singh, A. Dhanani, and D. Barradas, "RevealNet: Distributed Traffic Correlation for Attack Attribution on Programmable Networks," in Proceedings of the 23rd International Symposium on Network Computing and Applications (NCA), 2025, pp. 209–218, doi: 10.1109/NCA67271.2025.00042.), for which I am the first author. All major chapters of this thesis—including the solution design, and results—are directly derived from this paper. In several places, the text is reproduced verbatim, while in others it has been revised or expanded to provide additional detail, clarifications, and context appropriate for a masters thesis. Overall, this thesis represents a consolidated and expanded presentation of my original research contributions, building upon the published work while maintaining continuity with the original paper.

Abstract

Attributing network attacks to their sources is challenging as adversaries employ proxy chains, virtual private networks, and anonymity infrastructures to obscure their origins. Traffic correlation techniques mitigate this challenge by linking flows observed at multiple network vantage points using invariant characteristics such as timing and packet volume. However, existing attack attribution systems largely rely on centralized architectures that aggregate flow features at dedicated correlators, introducing computational and communication overheads that hinder scalability in high-speed networks.

This thesis discusses RevealNet, a decentralized framework for attack attribution that leverages P4-programmable switches to perform traffic correlation directly within the network fabric. RevealNet distributes feature extraction and correlation across cooperating networks, reducing dependence on centralized processing and minimizing telemetry offloading. Upon detection of a malicious flow, flow features are disseminated to participating switches, which locally correlate them against outgoing traffic using lightweight similarity metrics. To operate within the constraints of programmable data planes, RevealNet employs compact flow feature representations based on traffic aggregation matrices and sketching techniques designed for integer-only computation. The framework further incorporates heuristic optimizations that exploit temporal alignment and traffic-volume similarity to reduce correlation complexity and limit false positives.

Experimental evaluation conducted over a prototype of our framework using multiple real-world attack datasets demonstrates that RevealNet achieves attack attribution accuracy comparable to state-of-the-art centralized systems while significantly improving scalability. Notably, compact flow feature representations achieve accuracy comparable to complete flow representations, substantially reducing memory requirements without sacrificing attribution performance. Overall, RevealNet's distributed design reduces bandwidth overhead by up to 96% when deployed on a testbed consisting of 20 P4-enabled switches and enables programmable switches to correlate a significantly larger number of flows concurrently, demonstrating that attack attribution can be effectively decentralized within programmable network infrastructures.

Acknowledgements

First and foremost, I extend my deepest gratitude to Waheguru Ji (God), for providing me with the strength, wisdom, and serenity needed throughout my research journey and the writing of this thesis.

I would like to express my heartfelt thanks to my family. To my dear parents, whose love and guidance have been my constant light in the darkest of times, and to my brother, whose support and encouragement have been unwavering. Your faith in my abilities has been the bedrock of my resilience and determination.

Special thanks go to my supervisor, Diogo Barradas, for his invaluable guidance, patience, and expertise. Your mentorship has been instrumental in shaping both my research and my professional growth. I am deeply appreciative of the knowledge and insights you have shared with me.

I am also immensely grateful to Alim Dhanani for his assistance and support in the experimental setup.

In closing, I thank everyone who has been a part of this journey, directly or indirectly. Your support has been a source of motivation and inspiration, and for this, I am eternally grateful.

Table of Contents

Author's Declaration ii

Statement of Contributions iii

Abstract iv

Acknowledgements v

List of Figures viii

List of Tables ix

1 Introduction 1

1.1 Overview 1

1.2 Contributions 4

1.3 Thesis Outline 4

2 Related Work 5

2.1 Collaborative Intrusion Detection 5

2.2 Traffic Correlation 6

2.2.1 Flow correlation with fine-grained information 8

2.2.2 Flow correlation with coarse-grained information 11

2.3 P4 Switches as a Platform for Traffic Analysis 14

2.4 The Attack Attribution Problem 18

3 Methodology **21**

3.1 Architecture Overview . . . 21

3.2 Operational Workflow . . . 23

3.3 Flow Identification and Tracking . . . 25

3.4 Sketching and Correlation . . . 27

3.4.1 Traffic aggregation matrix . . . 28

3.4.2 Flow sketching . . . 28

3.4.3 Correlation . . . 33

3.5 Heuristic Optimizations for Attack Attribution . . . 34

3.5.1 Creation time heuristic . . . 34

3.5.2 Packet count heuristic . . . 35

3.5.3 Sequential application and complexity reduction. . . . 35

3.5.4 Implementation . . . 37

4 Evaluation **38**

4.1 Evaluation Goals and Metrics . . . 38

4.2 Evaluation Methodology . . . 39

4.3 Evaluation Results . . . 41

5 Conclusion **46**

5.1 Concluding Remarks . . . 46

5.2 Limitations and Future Work . . . 47

References **48**

List of Figures

2.1 Centralized correlation architecture for attack attribution. 19

3.1 RevealNet's decentralized correlation architecture for attack attribution. . . 22
3.2 RevealNet's dynamic flow identification and matching mechanism. 26
3.3 Heuristic impact on Coskun Sketch 36

4.1 Bitcoinminer (unperturbed). 42
4.2 Bitcoinminer (w/perturbations). 42
4.3 Length of TAM. 42
4.4 Correlation effectiveness of RevealNet across datasets. 43

List of Tables

3.1 Storage requirements 33

4.1 Summary of malicious/benign traffic datasets used in our experiments. . . 39
4.2 Number of feature vector comparisons required for each heuristic. 44
4.3 Total flows that can be stored in a P4 switch (for different methods). . . . 44
4.4 Communication overhead per sketch 45

Chapter 1

Introduction

1.1 Overview

In recent years, network attackers have increasingly relied on proxies [37], VPNs [6], and anonymity networks [25,38], to conceal their identities while engaging in malicious network activities. These relay-based anonymization tools route traffic through multiple intermediary servers, thereby obscuring an attacker's original IP address (i.e., a so-called *stepping-stone* attack [67]). Consequently, traditional approaches to identify the source of an attack, such as analyzing a flow's 5-tuple data, fail to trace malicious traffic effectively. This makes it challenging for network operators to perform *attack attribution* (i.e., to locate the true source of attacks), thereby preventing coordinated response efforts (e.g., via information sharing between ISPs), legal action, or better insights into attackers' tactics [14].

Beyond concealing an attacker's origin, stepping-stone attacks also introduce significant operational asymmetry between defenders and adversaries. While attackers can dynamically rotate relays, tunnel traffic across jurisdictions, and exploit shared infrastructure, defenders are often constrained to observing traffic only at a limited number of vantage points. As a result, identifying causal relationships between ingress and egress flows becomes increasingly difficult, particularly when attackers deliberately shape traffic to blend in with benign background flows. This asymmetry places a heavy burden on attribution mechanisms, which must operate under incomplete visibility while maintaining high accuracy. To uncover the sources behind malicious and anonymized traffic [18, 57], researchers have increasingly relied on traffic correlation techniques. These techniques aim to deanonymize malicious sources of traffic by analyzing and matching their traffic patterns (such as a flow's packets' timing and direction, and/or communication volume [35, 41]) as

observed by multiple *probe nodes* spread across the network [9, 51]. By correlating observations made at different locations, these approaches seek to establish whether two flows belong to the same underlying communication session, even when intermediate relays obscure addressing information. Prior work has explored a wide range of correlation strategies, spanning statistical techniques [32, 40, 50] and more recent machine learning-based methods [10, 41, 44]. While these approaches have demonstrated promising accuracy, they typically rely on a centralized architecture in which probe nodes export flow features to a dedicated *correlator node*. This correlator aggregates features across vantage points and performs the computationally intensive correlation task.

Such centralized designs introduce two fundamental scalability bottlenecks. First, exporting flow-level features—often at fine temporal granularity—imposes substantial bandwidth overheads, particularly in high-throughput environments. Second, concentrating correlation logic at one or a few correlator nodes creates computational hot spots, limiting the system's ability to scale with the number of monitored flows or probe locations. Although decentralized variants have been proposed [42, 50], these typically involve partitioning correlation tasks across multiple correlators rather than eliminating centralized processing altogether. As a result, probes still transmit flow features in bulk, and scalability challenges—especially computational ones—remain only partially addressed.

Addressing these limitations is particularly challenging in modern high-speed networks, such as software-defined and programmable infrastructures deployed in 5G and edge environments. In these settings, traffic volumes grow rapidly with link speeds, while operators demand increasingly fine-grained telemetry and rapid reaction times to uphold performance and security guarantees [5, 33, 43]. Offloading large volumes of telemetry data to external servers is often impractical due to bandwidth constraints, latency sensitivity, and operational complexity [64]. Consequently, there is growing interest in pushing security and monitoring functionality closer to the data plane.

To mitigate telemetry overheads in attack attribution, researchers have investigated feature aggregation [35] and compression techniques that produce compact *flow sketches* [15, 42]. These sketches summarize essential traffic characteristics in a fixed-size representation, significantly reducing storage and communication costs while preserving sufficient information for correlation. Sketching techniques thus represent an important step toward scalable attribution. However, sketches alone do not resolve a key architectural limitation: in most existing designs, sketches are still exported to centralized or semi-centralized correlators for comparison. As a result, while sketches reduce bandwidth usage, they do not fundamentally change where correlation occurs or how correlation workloads scale with network size. In large deployments, even compact sketches can accumulate to substantial data volumes, and centralized correlators remain a performance bottleneck.

This thesis introduces RevealNet, a framework for attack attribution that fundamentally rethinks where and how flow correlation is performed. Rather than treating programmable switches solely as passive probes, RevealNet leverages their increasingly rich data-plane capabilities to perform both feature extraction and correlation in a decentralized manner. By enabling switches to act as autonomous probe and correlator nodes, RevealNet eliminates the need for special-purpose correlation servers and minimizes inter-domain data exchange.

At the core of RevealNet is the observation that P4-programmable switches, including both hardware and software targets—such as Intel Tofino [27], AMD Pensando [1], and P4-programmable software switches like BMv2 [49] and T4P4S [48]—are now widely deployed in high-performance networks and already support sophisticated security functions with minimal overhead [23, 68]. Despite this, their potential for performing traffic correlation remains largely unexplored. This raises a natural question: can programmable switches support correlation-enabling primitives—such as line-rate flow sketches and lightweight similarity metrics—so as to enable fast, distributed correlation in the control plane and reduce reliance on centralized analysis?

Answering this question is non-trivial. Programmable switches operate under strict constraints on memory, computation, and arithmetic expressiveness. They lack floating-point support, provide limited per-flow state, and must process packets at deterministic, high speeds. RevealNet addresses these challenges by carefully adapting sketching techniques to the switch data plane, using integer-only arithmetic, fixed-size memory structures, and online update rules that avoid per-packet complexity explosions. In addition, RevealNet incorporates lightweight heuristics to prune irrelevant flows early, further reducing correlation overheads. Our evaluation demonstrates that RevealNet achieves attribution accuracy comparable to centralized systems while significantly improving scalability. By decentralizing correlation and minimizing data exchange, RevealNet allows programmable switches to track more flows and dramatically reduces inter-network communication. In a decentralized deployment spanning 20 cooperating networks, RevealNet reduces bandwidth overheads by up to 96% while maintaining effective attack attribution.

Scope of This Work This research investigates whether traffic correlation architectures that traditionally rely on centralized correlators can be shifted to a decentralized design using programmable switches. Due to the computational and memory constraints of P4 switches discussed above, this work employs simpler feature representations that are easier to compute in the data plane, which may affect correlation accuracy. Future work can explore richer feature representations to improve accuracy while preserving the scalability benefits of the proposed approach.

1.2 Contributions

The contributions of this thesis can be summarized as follows:

- We design RevealNet, a decentralized attack attribution framework that leverages P4-programmable switches as autonomous probe and correlation nodes.
- We adapt prominent flow sketching techniques to the constraints of programmable switch data planes, enabling efficient online sketch construction and comparison using integer-only operations.
- We implement RevealNet on the `bmv2` reference P4 switch and evaluate its accuracy, computational cost, and communication overhead in realistic multi-network scenarios.

1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 discuses the need for collaborative security apparatuses, then explores surveys related work on traffic correlation and attack attribution, with particular emphasis on prior approaches that leverage fine-grained and coarse-grained flow information. The chapter further discusses the role of P4-programmable switches as a platform for traffic analysis and situates this work within the context of the attack attribution problem.

Chapter 3 presents the design underlying RevealNet. We begin with an overview of the system architecture and operational workflow, followed by a detailed description of flow identification and tracking mechanisms. The chapter then introduces compact flow feature representations, including traffic aggregation matrices and sketch-based techniques, and explains how these representations are used for flow correlation. Finally, the chapter describes the heuristic optimizations employed by RevealNet to reduce correlation complexity and improve scalability, along with their implementation details.

Chapter 4 evaluates the effectiveness and efficiency of RevealNet. We describe the evaluation goals, metrics, and experimental methodology, and present results that quantify correlation accuracy, computational overhead, and communication costs under various configurations and workloads. Chapter 5 concludes the thesis by summarizing the main findings, discussing the limitations of the proposed approach, and outlining promising directions for future work.

Chapter 2

Related Work

In this chapter, we review prior work related to traffic correlation and attack attribution. We begin by reviewing the design of collaborative intrusion detection systems (§2.1) which serves as an inspiration for this work. Then, we survey existing traffic correlation techniques (§2.2), distinguishing between approaches that rely on fine-grained packet-level information and those based on coarse-grained flow-level features. We then discuss recent efforts that leverage P4-programmable switches and data-plane programmability for traffic analysis and network security applications (§2.3). Finally, we situate this work within the specific attack attribution problem (§2.4), highlighting the limitations of existing solutions and motivating the need for scalable, decentralized attribution mechanisms.

2.1 Collaborative Intrusion Detection

Traditional intrusion detection systems (IDS) operate primarily in isolation, relying on locally observed host or network data to identify malicious activity. While effective for detecting well-defined or localized threats, such systems are fundamentally limited when facing distributed, low-rate, or coordinated attacks whose indicators are only observable when correlating data across multiple vantage points. This limitation has motivated the development of *Collaborative Intrusion Detection Systems* (CIDS), where multiple autonomous entities cooperate by sharing information to improve detection accuracy [39].

Collaborative intrusion detection is a specific instantiation of the broader collaborative security paradigm, in which independent nodes exchange alerts, features, or analytical results to collectively identify security threats. In a CIDS setting, participating entities

may include hosts, network sensors, Internet service providers, or organizational domains, each contributing partial observations of network traffic or system behavior. By aggregating evidence across these entities, collaborative systems can detect attack patterns that remain inconspicuous when analyzed in isolation.

Prior work [65] has shown that collaboration significantly enhances the detection of distributed attacks such as coordinated scanning, botnet activity, command-and-control (C2) communication, and traffic correlation attacks. However, enabling collaboration introduces a number of fundamental challenges. First, sharing raw traffic traces, logs, or alerts may lead to privacy leakage, as such data often contains sensitive information about users, network topology, or organizational behavior. Second, collaborative systems must remain robust against untrusted or compromised participants that may inject false or misleading information. Third, scalability and communication overhead become critical concerns as the number of participants and the volume of exchanged data increase.

Existing CIDS architectures span centralized, decentralized, and hybrid designs. Centralized approaches simplify correlation and decision-making but suffer from single points of failure and increased privacy risks. Fully decentralized approaches improve robustness and scalability, but complicate coordination, trust management, and global inference. Hybrid architectures attempt to balance these trade-offs, yet still rely on carefully designed information-sharing mechanisms to prevent unnecessary exposure of sensitive data.

These observations also motivate the problem setting considered in this thesis. In many collaborative intrusion detection deployments, a local IDS can flag suspicious activity, but attributing the true source of an attack (e.g., through proxy chains or VPNs) often requires correlating traffic observations across multiple administrative domains. Existing collaborative security frameworks therefore highlight the need for mechanisms that support cross-domain evidence aggregation while controlling communication overhead and limiting the exposure of sensitive telemetry.

2.2 Traffic Correlation

Traffic correlation techniques can be used to analyze traffic patterns and link together flows which are observed at the entry and exit nodes of proxy chains [28]. In practice, these attacks exploit the fact that—even though anonymity networks such as Tor encrypts traffic—observable characteristics such as timing, volume, and direction remain statistically consistent as the traffic traverses the network. By monitoring both ends of a user's communication path, an adversary can compare these patterns and determine which incoming and

outgoing flows correspond to one another, effectively deanonymizing the user. Johnson et al. [28] show that such correlation is surprisingly feasible for realistic adversaries, including malicious Tor relays, autonomous systems, or Internet exchange points, all of which may be able to observe traffic at multiple points along a circuit. Their analysis demonstrates that even passive observation is sufficient to link flows with high confidence, revealing that users are far more vulnerable to end-to-end correlation than previously believed.

While many correlation techniques were introduced with the aim of gauging the privacy provided by anonymity systems [9, 40, 46] and mix networks [45], others were developed with the specific intent to trace stepping-stone network attackers [15,55,63]. These diverse techniques share a common goal: to determine whether two observed traffic streams correspond to the same underlying communication, despite encryption, routing obfuscation, or relay-based indirection. Over time, researchers have leveraged increasingly sophisticated statistical features—such as timing distributions, packet counts, and flow-level behavioral signatures—to improve correlation robustness in adversarial settings. While the literature uses slightly different taxonomies, we can roughly distinguish two main axes:

Passive vs. active correlation. In passive correlation, traffic is observed without any modification, and flows are matched by analyzing naturally occurring patterns in timing, size, or rate. This is the predominant setting in high-speed networks and the primary focus of our work. In contrast, active correlation techniques intentionally introduce identifiable patterns—such as controlled delays or packet bursts—into traffic to facilitate later matching. While active correlation can improve robustness under certain conditions, it is often more detectable and harder to deploy in operational environments.

Below, we describe two main classes of prominent *passive flow correlation* techniques. The first category comprises approaches that operate on fine-grained per-packet data, providing high accuracy by examining detailed temporal and size-based characteristics of individual packets, though at the expense of substantial storage and processing overhead. The second category encompasses techniques that rely on coarse-grained per-flow summaries, which significantly reduce data collection and retention requirements but typically sacrifice some degree of precision and sensitivity to subtle traffic patterns. Together, these classes illustrate the fundamental trade-off between resource efficiency and correlation fidelity that underpins much of the passive traffic-analysis literature.

2.2.1 Flow correlation with fine-grained information

Most studied traffic-correlation techniques rely on fine-grained, per-packet information. These methods operate directly on raw packet-arrival timestamps, exploiting subtle timing dependencies that survive encryption, padding, and routing indirection. Zhu et al. [71] demonstrate this principle clearly: by recording precise per-packet arrival times at both the input and output links of a mix, they compute average traffic rates over short intervals to construct pattern vectors that capture the temporal structure of flows. Their work shows that even when mixes reorder packets, batch them, or otherwise perturb traffic, the underlying timing behavior—driven by protocol dynamics such as TCP congestion control—creates distinctive signatures that leak through. Using these fine-grained features, their correlation framework extracts rate-based vectors for each flow, then applies statistical distance metrics, such as mutual information or frequency-domain matched filtering, to evaluate how strongly an observed output aggregate depends on a given input stream. The striking result is that this per-packet granularity enables near-perfect flow linkage when enough data is collected, even under heavy cross-traffic and across a range of batching strategies. Palmieri [50] used wavelet-based analysis to capture timing, size, and rate variations across flows. His approach applies discrete wavelet multi-resolution decomposition to traffic time series, allowing fine- and coarse-scale fluctuations in packet counts, byte volumes, inter-arrival times, and related features to be isolated and compared. By transforming each flow into a hierarchy of wavelet coefficient vectors, the method exposes hidden structural patterns that remain consistent even after the delays, padding, and noise introduced by anonymizing overlays.

Recently, researchers adopted deep learning approaches to improve flow correlation, pushing accuracy well beyond that of traditional statistical methods. DeepCorr [41], DeepCoFFEA [44], and ESPRESSO [10] progressively advance the state of the art by introducing increasingly expressive learned correlation functions. DeepCorr uses convolutional neural networks (CNNs) to learn correlation functions, DeepCoFFEA further introduces feature embedding and voting mechanisms, and ESPRESSO combines transformer models and CNNs to capture both global and local traffic patterns.

DeepCorr [41] was the first to demonstrate that a supervised CNN can directly learn a Tor-specific correlation function from raw traffic sequences. The system encodes each bidirectional flow as ordered vectors of packet directions, sizes, and inter-arrival times observed at the guard and exit positions, and processes these paired sequences through a CNN architecture that jointly extracts local timing–size patterns from both sides. The network is trained to output high similarity scores for matching entry/exit flows and low ones otherwise, enabling highly accurate correlation even under substantial network noise,

limited traffic observation, and circuit churn. DeepCorr's evaluation shows that the learned model achieves around 96% accuracy with only a few hundred packets per flow—far surpassing classical metrics such as mutual information or Pearson correlation under the same constraints.

DeepCoFFEA [44] builds on this insight by recasting correlation as a deep metric-learning problem: rather than relying on a single global classifier, it learns a compact embedding space in which flows from the same Tor connection lie close together, while unrelated flows remain distant. A Siamese CNN extracts embeddings from short flow segments, and correlation is performed by computing distances across many small windows, with a voting mechanism that aggregates these per-window decisions. This enables DeepCoFFEA to remain robust at Internet scale, drastically reducing false positives even when correlating against millions of candidate flows, and to retain high accuracy in the presence of pluggable transports such as `obfs4` [20].

ESPRESSO [10] makes a notable conceptual shift in deep-learning-based flow correlation by introducing Transformer-based global processing into the correlation pipeline. While prior systems such as DeepCorr and DeepCoFFEA rely almost entirely on convolutional networks—which excel at extracting short-range, local patterns—ESPRESSO observes that Tor traffic contains important long-range temporal relationships that CNNs alone cannot adequately model. To address this, ESPRESSO incorporates a MultiHeadAttention Transformer layer that operates over the full aggregated feature sequence, allowing the model to jointly attend to distant portions of the flow and learn dependencies spanning the entire trace. This global attention mechanism enables the model to integrate information across far-apart time windows, improving sensitivity to subtle high-level structures that persist through Tor's timing noise and packet-shuffling behavior. The ablation results in the paper clearly show that removing the Transformer component sharply increases false positives (up to three orders of magnitude) and reduces overall accuracy, confirming that global modeling is essential. In contrast, local-only CNN processing fails to capture the sequential coherence of the full trace, whereas the Transformer enriches embeddings with context from the whole sequence, producing more discriminative representations. By bringing Transformer-style global reasoning to flow correlation, ESPRESSO advances the field beyond localized pattern matching and demonstrates that long-range temporal dependencies are key to reducing FPR and improving robustness under obfuscation.

Brighente et al. [8] provide a critical reassessment of flow correlation attacks by evaluating the real-world effectiveness of both deep-learning-based and non–deep-learning approaches (that use coarse grained information see §2.2.2) under modern Tor deployments and defenses. In contrast to prior studies reporting high accuracy in controlled or outdated settings, the authors systematically re-evaluate ESPRESSO and SUMo [35] using a newly

collected dataset generated with the latest Tor version and realistic usage scenarios, including pluggable transports, Tor-over-VPN configurations, and circuit-splitting countermeasures. Their results show that, when evaluated under realistic conditions and with careful accounting for false positives, both attacks perform poorly, achieving correlation accuracies below 20%. The authors further demonstrate that previously reported high accuracy figures are largely driven by severe class imbalance and misleading evaluation metrics, where models attain high overall accuracy by predominantly predicting non-correlation. Moreover, the study highlights how contemporary Tor defenses—such as Vanguard, Conflux, circuit rotation, and widespread bridge usage—substantially undermine the assumptions underlying modern flow correlation models.

Huang et al. [26] propose FlowCoPCL, an enhanced deep-learning-based flow correlation attack on Tor that addresses temporal inconsistency in prior CNN-based approaches such as DeepCorr and DeepCoFFEA. The authors observe that packet arrivals in Tor flows are irregular in time, causing CNN filters to operate over packet sequences that span unequal temporal durations and resulting in unreliable flow embeddings. To mitigate this issue, FlowCoPCL introduces a time-based patching mechanism that partitions each flow into fixed-duration time intervals, within which packet sizes and inter-packet delays are extracted and padded to a uniform length. These temporally consistent patches are processed by a patch-based feature embedding network, followed by a convolutional architecture that produces a single embedding per flow. The embedding networks are trained using a contrastive learning framework adapted from SimCLR [11], where correlated Tor–exit flow pairs are treated as positive samples and unrelated flows as negatives, enabling efficient metric learning and reducing computational overhead at inference time. Flow correlation is then performed by computing cosine similarity between the learned embeddings, and experimental results demonstrate that FlowCoPCL significantly outperforms prior deep-learning-based attacks, achieving high true positive rates at very low false positive rates and improved robustness against lightweight traffic defenses.

While the above approaches achieve high reported accuracy, they rely on the collection, transmission, and processing of fine-grained traffic features—such as packet direction, size, and timing—across entire traces, which makes them costly to deploy at scale (§2.4, [42]). As a result, despite their technical effectiveness, these methods may be impractical for large operators or adversaries seeking to monitor high-throughput networks or a large number of concurrent flows simultaneously.

2.2.2 Flow correlation with coarse-grained information

Collecting and storing fine-grained traffic features for flow correlation at choke points (e.g., ISP border routers) is increasingly challenging due to the high volume and speed of traffic, which strain storage and processing resources. To overcome this, a line of work has explored *coarse-grained* or *compressed* representations that summarize flows while preserving the statistical structure required for correlation.

Coskun et al. [15] propose one of the earliest and most influential approaches, introducing an online sketching algorithm that maintains a compact *packet-timing sketch* for each flow. Their method processes packets in a streaming fashion, updating a fixed-length integer vector using linear projections of inter-arrival times and time-slot counters. Mathematically, each sketch is updated as

$$s_k \leftarrow s_k + f_k(\Delta t),$$

where s_k denotes the k-th entry of the flow's sketch vector, Δt is the inter-arrival time between consecutive packets, and $f_k(\cdot)$ is a randomized hash-based projection function associated with index k. These sketches preserve pairwise similarity under bounded jitter and chaff, allowing the distance between two flows' sketches to approximate their true timing-distance with high probability. Coskun et al. further prove an expectation bound on the sketch difference under delay and chaff models, showing that correlated flows satisfy

$$\mathbb{E}[\|s^{(i)} - s^{(j)}\|_1] \leq B(\delta, \alpha),$$

where $s^{(i)}$ and $s^{(j)}$ denote the sketch vectors of flows i and j, respectively, δ bounds the maximum packet delay, α represents the chaff injection rate, and $B(\cdot)$ is a function capturing the resulting upper bound on expected sketch divergence.

Nasr et al. [42] introduce *compressive traffic analysis*, which adapts compressed-sensing theory to traffic features. Let $x \in \mathbb{R}^N$ denote a raw flow feature vector of dimension N, such as a histogram over packet counts or inter-packet delays. They generate a compressed representation

$$y = \Phi x, \quad \Phi \in \mathbb{R}^{M \times N},\ M \ll N,$$

where Φ is a random projection matrix that maps the original N-dimensional feature space into a lower-dimensional space of size M. Since many traffic features are sparse or compressible, and because Φ satisfies the Restricted Isometry Property (RIP) with high probability, Euclidean distances are approximately preserved:

$$(1-\epsilon)\|x_i - x_j\|_2 \leq \|\Phi x_i - \Phi x_j\|_2 \leq (1+\epsilon)\|x_i - x_j\|_2,$$

where x_i and x_j are the original feature vectors of flows i and j, respectively, and ϵ controls the distortion introduced by the projection. Thus flow correlation can be performed *directly* on the compressed vectors y_i, eliminating the need to reconstruct x_i. This drastically lowers storage and communication overhead at large observation points and speeds up correlation by orders of magnitude. Their experiments show that compressive correlation maintains true-positive rates comparable to full-feature correlation.

Lopes et al. [35] propose SUMo, a coarse-grained flow correlation attack targeting Tor onion services that deliberately avoids fine-grained packet alignment and deep learning. Given an encrypted flow composed of packets with timestamps t_p and sizes s_p, SUMo discretizes time into fixed intervals of length Δ and represents each flow as a sequence of aggregated packet volumes $B_i = \sum_{p:\, t_p \in [i\Delta,(i+1)\Delta)} s_p$. Correlation is performed over sliding windows of consecutive bins, where the method tests whether the client-side volume observed within a window can be explained as a subset sum of the corresponding onion-service bin volumes within a bounded tolerance, rather than by exact packet-level matching. This formulation discards packet order and fine-scale timing information while preserving coarse volumetric structure across time, making the approach robust to Tor-induced jitter, padding, and session multiplexing. Machine learning is employed only in a preliminary filtering stage based on summary statistics of traffic bursts and inter-arrival times to identify onion-service-related flows and reduce the candidate search space, while the final correlation decision relies entirely on deterministic, coarse-grained volumetric matching.

Fiore and Palmieri [21] propose a distance-based network activity correlation framework for de-anonymizing traffic in anonymization overlay networks such as Tor. In their formulation, traffic observed at the ingress and egress of the overlay is represented as multi-dimensional time series, where each trace consists of regularly sampled traffic properties (e.g., packet rate, bit rate, average packet size, and inter-arrival time) measured over fixed intervals. Correlation is cast as a bipartite matching problem between source-side and destination-side traces, where each potential association is weighted by a distance function that quantifies dissimilarity between the corresponding time series. The authors evaluate several distance metrics, including the Wasserstein distance, Kullback–Leibler divergence, and Wave-Hedges distance, applied either directly to the temporal sequences or to empirical distributions derived from them. Associations are inferred by selecting, for each source trace, the destination trace with minimum distance, with normalized inverse distances used to estimate confidence scores. To improve scalability and enable near–real-time operation, it also supports correlation over sliding windows of the time series, trading temporal resolution for reduced computational cost. By relying on coarse-grained traffic statistics and interpretable distance measures rather than learned models, this approach emphasizes scalability, and deployability on lightweight network monitoring infrastructure.

Zhu et al. [72] propose TOScorr, a Transformer-based flow correlation attack specifically designed for Tor onion service traffic. Unlike earlier deep-learning approaches that rely on fine-grained packet sequences, TOScorr first aggregates packet-level observations into coarse-grained Traffic Aggregation Matrices (TAMs), where packets are counted per direction over fixed time slots. These matrices provide a compact representation that preserves coarse temporal and directional patterns while discarding exact packet ordering and fine timing. TOScorr then applies a metric-learning architecture combining convolutional layers for local pattern extraction with Transformer self-attention to capture long-range temporal dependencies across the aggregated sequence. Client-side and onion-service flows are mapped into a shared low-dimensional embedding space using triplet loss, and correlation is determined via cosine similarity across multiple sliding windows with majority voting. Although TOScorr operates on aggregated features during correlation, it still assumes initial access to packet-level timing and direction information to construct the TAM representation. The approach demonstrates high accuracy on onion service datasets, illustrating the effectiveness of combining coarse-grained aggregation with expressive deep-learning models for robust flow correlation.

In Nasr et al. [42] and Lopes et al. [35], the proposed compact representations are still derived from packet-level observations, requiring the initial collection of per-packet timing and size information before aggregation. A similar observation applies to the distance-based correlation framework of Fiore and Palmieri [21]. Although their method ultimately operates on coarse-grained representations—namely multi-dimensional time series of aggregated traffic statistics computed over fixed sampling intervals—these features are likewise derived from packet-level observations and thus require packet collection prior to aggregation. The authors further explore correlation over sliding windows of the feature sequences, explicitly trading temporal resolution for reduced computational and storage overhead; however, the coarse representations themselves are not generated directly online. A comparable design choice is made by Zhu et al. [72], whose TOScorr attack aggregates packet-level timing and direction observations into Traffic Aggregation Matrices before applying deep-learning-based correlation. While TOScorr performs correlation over sliding windows of these aggregated representations and avoids storing complete packet traces during inference, it nevertheless assumes access to packet-level metadata during feature construction and does not generate coarse features directly online.

Taken together, these works reinforce the view that effective flow correlation need not rely on persistent storage of fine-grained packet traces during inference. Nevertheless, they also highlight a remaining gap in prior approaches: coarse-grained representations are typically constructed only after packet-level collection. If such representations can instead be generated in an online fashion, memory requirements at traffic collection nodes

can be substantially reduced while still supporting effective correlation. Inspired by these observations, we conjecture that coarse-grained and compressed methods can enable flow attribution using only limited on-the-fly flow data (§3.4).

2.3 P4 Switches as a Platform for Traffic Analysis

This section provides a foundational overview of P4-programmable switches and explains why they have emerged as a compelling substrate for high-speed traffic analysis. We first describe the architectural model of programmable switches, focusing on the division of responsibilities between the data and control planes, followed by a deeper explanation of the P4 processing pipeline. We then discuss the fundamental hardware and programming constraints that shape what is feasible to compute at line rate.

Data and control planes Traditional network devices implement a fixed-function data plane responsible for forwarding packets and a control plane that installs forwarding rules and manages routing protocols. In such architectures, the data plane is rigid: packet parsing, classification, and modification are hard-coded into ASIC logic, limiting the ability to introduce new behaviors or measurement functionality. Programmable switches disrupt this model by exposing a reconfigurable data plane that operators can program using domain-specific languages such as P4. The central idea is to decouple *how* packets are processed (defined in the data plane program) from *which* specific rules or entries are active at runtime (controlled by the control plane). The control plane remains responsible for populating match-action tables, reacting to traffic dynamics, and orchestrating coordinated responses across the network. However, unlike in fixed-function switches, the data plane is no longer static; it is explicitly defined by the operator and can perform custom measurement, filtering, and stateful operations while maintaining line-rate throughput.

P4-programmable switch architecture A P4-programmable switch, such as Intel Tofino [27] or AMD Pensando Elba [1], exposes a multi-stage packet-processing pipeline optimized for deterministic latency and high throughput. When a packet arrives, it first enters a programmable parser that extracts protocol headers and metadata. The extracted fields then traverse a sequence of *match–action stages*, each of which performs a bounded amount of computation. Within each stage, the switch evaluates a set of match-action tables. A table specifies which packet fields to match on (e.g., exact match, prefix match, or ternary match) and maps each matched entry to a user-defined *action*. Actions can rewrite

headers, execute arithmetic or bitwise operations, and update stateful memory objects such as registers or counters. Although designers refer to the data plane as "programmable," its programmability operates under strict timing and resource constraints: every stage must complete its work within a single clock cycle, and at terabit-per-second rates, this budget is extremely tight. The control plane interacts with the data plane by inserting, updating, and deleting table entries at runtime. This allows policies to adapt to traffic changes without recompiling the P4 program. For example, the control plane may install new monitoring rules, modify feature extraction configurations, or trigger mitigation actions in response to observed anomalies. Thus, the combination of a programmable pipeline and a flexible control interface enables rapid reconfiguration of in-network analytic functions.

Constraints and hardware limitations Despite their expressiveness, P4 switches impose strict limitations that fundamentally shape the design of any in-network processing system. For example, hardware pipelines do not support dynamic data structures such as queues, lists, or resizable hash maps; the P4 language deliberately omits features like dynamic memory allocation and complex control structures that cannot be efficiently implemented in fixed-function pipelines [29,47]. Instead, all state must be allocated statically at compile time, which forces designers to bound memory usage tightly and to rely on approximate structures when tracking large numbers of flows.

Second, P4 hardware lacks floating-point arithmetic, division, modulo, and other complex operations. Only simple integer operations and bit manipulations can be executed at line rate. This is a critical limitation for applications that require numerical feature extraction or statistical computations; designers must approximate these operations using integer scaling, lookup tables, or multi-stage decompositions [4].

Third, memory is limited: modern P4 switches typically provide on the order of 100–256 MB of SRAM, spread across pipeline stages [3]. The memory within each stage is also divided across multiple functional units, leading to tight per-stage resource budgets. This fragmentation makes it non-trivial to store per-flow features or maintain rich flow state without compression or lossy encodings.

Finally, performing many operations in a single stage is infeasible. Each stage has a fixed computational budget, allowing only a small number of ALU operations and at most one stateful memory access per packet. As a result, designers must carefully pipeline computations across stages while respecting data dependencies. This challenge has been documented extensively in past systems, which rely on clever "hacks" such as bit slicing, multi-stage hashing, register partitioning, or repurposing metadata to propagate intermediate results [3,61,69]. In summary, P4 switches provide a powerful yet highly constrained

computational model for processing packets at line rate. Their architectural separation of data and control planes, combined with a deeply pipelined match–action architecture, enables scalable per-packet analytics and feature extraction. However, the absence of dynamic memory, limited arithmetic capabilities, and strict timing constraints require careful design and often force applications to adopt approximate or compressed representations of flow state. These properties guide the design of our system, which leverages the available primitives to compute flow-level features efficiently within the switch pipeline (§3).

Traffic analysis on P4 switches P4-programmable switches have emerged as a compelling platform for high-speed traffic analysis due to their ability to process every packet at line rate while extracting meaningful flow-level information. A large body of work [12, 70] demonstrates that the data-plane programmability offered by P4 enables deployments of security and monitoring functions that were previously feasible only on software-based middleboxes or end hosts. As networks continue to scale and link speeds approach 400 Gb/s and beyond, the shift toward in-network analytics has become increasingly attractive: relocating key components of the traffic analysis pipeline into the switch itself reduces the data movement overhead, increases responsiveness, and enables finer-grained insights into real-time traffic dynamics.

A substantial portion of prior work focuses on leveraging P4 switches for traffic classification. Early efforts demonstrated that switches can compute lightweight per-flow or per-packet features and export them for downstream ML classifiers, yielding substantial speedups compared to host-based feature collection. For instance, Lopes et al. [35] and Xie et al. [60] showed that protocol- and behavior-based traffic classification is feasible within the tight constraints of the data plane by relying on simple, integer-based feature aggregations. More recent systems, such as in-network hierarchical classifiers [59] and NetBeacon [69], push this boundary further by incorporating coarse ML logic into the switch pipeline itself. These systems collectively illustrate that, with careful design, programmable switches can operate as high-speed pre-processors for ML-driven traffic analysis tasks.

Beyond classification, P4-based designs have been adopted for detecting covert channels [61], spotting fine-grained evasion attempts, and supporting real-time mitigation of volumetric attacks such as DDoS [34, 58]. These tasks rely on the ability of the switch to update stateful data-plane structures (e.g., registers, counters, sketches) in response to packet arrivals, enabling tracking of flow dynamics, burstiness patterns, and protocol irregularities. The constraints of P4 hardware encourage approximate yet highly efficient representations; in practice, these enable detection mechanisms that would be prohibitively expensive to run at line rate on commodity CPUs.

Achieving scalable traffic analysis in programmable switches requires managing state for thousands or millions of concurrent flows. To address this, researchers have proposed new memory management mechanisms and optimized data structures designed specifically for the P4 programming model. SuperFaST [66], for example, introduced scalable flow-entry aggregation and estimated-flow management techniques that reduce memory pressure while maintaining high fidelity in measurement outputs. Such systems showcase the evolution of P4 program design from naïve per-flow storage to carefully engineered abstractions that strike a balance between accuracy and hardware feasibility.

A particularly exciting direction in recent years involves pushing ML-style inference into the switch pipeline. NetBeacon [69] demonstrates that simplified decision structures, such as threshold-based classifiers, can be embedded into the data plane for near-instantaneous detection of anomalies. Brain–on–Switch [62] extends this idea further by exploring whether neural-network-inspired operations—heavily simplified to meet hardware constraints—can be adapted to match–action pipelines. These explorations provide valuable insights into the limits and opportunities of performing intelligence directly at line rate, enabling forms of traffic analysis that go beyond simple feature extraction.

Despite this progress, to the best of our knowledge, *P4 switches have not yet been used for generalized flow correlation tasks.* Flow correlation—linking flows that belong to the same communication session or attacker activity—is a challenging problem because it typically requires extracting and comparing richer, time-dependent features across distributed vantage points. The closest work to our scenario is DELTA [30], which configures P4 switches to identify VoIP call establishments and exchange metadata to infer communicating endpoints. While DELTA demonstrates that coordinated, multi-switch inference is feasible, it does not compute packet-level or temporal features that would generalize to correlation across arbitrary flow types or adversarial contexts.

This gap highlights an unexplored opportunity: programmable switches may serve as the foundation for a decentralized and highly scalable flow-correlation substrate capable of supporting security tasks such as attack attribution, multi-hop tracing, and cross-domain incident reconstruction. Given their ubiquitous deployment in modern datacenters and backbone networks, P4 switches are uniquely positioned to operate as in-network sensors that produce compact, time-synchronized feature summaries with negligible overhead. In our work, we explore this possibility and design a switch-assisted correlation architecture (§3) that operationalizes flow-correlation logic within the constraints of existing programmable-switch hardware while distributing the computational load across multiple vantage points in the network.

2.4 The Attack Attribution Problem

Our scenario models a networked environment such as the Internet, consisting of a set of interconnected networks $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$. Each network is managed by an operator, e.g., an Internet Service Provider (ISP), a cloud service provider (CSP), or a university campus/enterprise network administrator. These networks typically deploy local intrusion detection systems (IDSes) in firewalls, servers, or end-hosts, and may share telemetry or flow-level metadata to support collaborative cybersecurity operations. For conciseness, we refer to networks willing to exchange such information as *cooperating networks*. Cooperation is especially important in modern networked environments, where attacks frequently involve distributed infrastructures, multi-hop relays, or compromised proxies, making local visibility alone insufficient for attack investigation and response.

When an IDS signals an attack on a given network N_i, we assume that the objective of the network operator is not merely to identify and respond to the event (e.g., by dropping the offending flow and/or temporarily preventing further communication from a suspicious IP address), but to carry out *attack attribution*. In this context, we define attribution as the process of uncovering the true IP address $a \in \mathcal{N}_{\mathsf{IP}}$ launching the attack, despite the possibility that the attack traffic has traversed layers of indirection such as VPNs, forwarding proxies, stepping stones, or compromised hosts. This is a significantly harder task than straightforward intrusion detection, as it requires correlating observations across heterogeneous administrative domains.

To perform attack attribution, a network operator extracts traffic features from the malicious flow—denoted f_m^i—as observed in the attacked network N_i. These features can include timing signatures, inter-arrival distributions, packet size patterns, or other behavioral indicators that remain stable across different vantage points. The operator then attempts to correlate these features with those of other flows $f_k \in \mathcal{F}$ originated within cooperating networks in $\mathcal{N} \setminus \{N_i\}$. Here, $f_k^j \in \mathcal{F}(N_j, a)$ denotes a flow originated within network N_j whose source IP is a. Intuitively, such flows may represent earlier stages of the same attack, outbound connections from a compromised device, or parallel flows generated as part of coordinated malicious activity.

Let $\rho(f_m^i, f_k^j)$ denote the correlation between the features of f_m^i and f_k^j. Correlation may reflect similar traffic timing and volume characteristics, packet-burst rhythms, or application-layer behavior. A sufficiently high correlation score suggests that the two flows may belong to the same communication path or originate from a common attacker-controlled source, albeit observed at different points in the network.

Hence, the *attack attribution* problem can be formalized as identifying the most likely

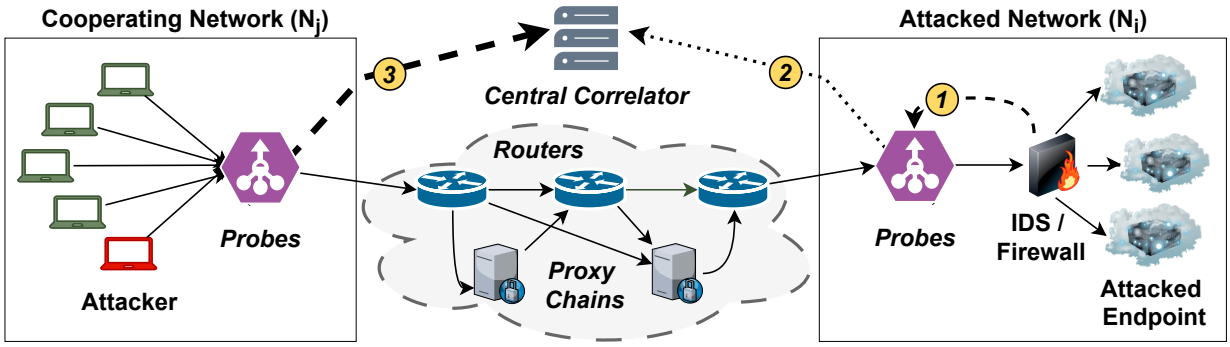


Figure 2.1: Centralized correlation architecture for attack attribution.

attacker's IP address $\hat{\mathcal{A}}$, as follows:

$$\hat{\mathcal{A}} = \arg\max_{a \in \mathcal{N}_{\text{IP}}} \sum_{N_j \in \mathcal{N} \setminus \{N_i\}} \sum_{f_k^j \in \mathcal{F}(N_j, a)} \rho(f_m^i, f_k^j) \quad \text{subject to } \rho(f_m^i, f_k^j) \geq \eta$$

where η is a tunable flow similarity threshold used to filter out noise and weak correlations. The optimization effectively aggregates evidence across cooperating networks to infer the attacker identity, making the attribution process robust to network noise and partial visibility.

Scalability challenges of centralized attack attribution. Figure 2.1 illustrates a conventional centralized flow-correlation architecture for performing attack attribution. When an attack is detected in network N_i, its IDS instructs a local probe (step 1) to forward the malicious flow's feature vector to a central correlation server (step 2). Meanwhile, all cooperating networks continuously forward feature vectors for their outgoing flows to the correlator (step 3). The correlator aggregates these vectors, computes similarity scores between the attacking flow and candidate flows across networks, and ultimately infers the most likely attacker.

While conceptually straightforward, this architecture suffers from inherent scalability problems. The central correlator must process feature vectors for *all* flows observed by cooperating networks, yielding a computation and communication burden proportional to $|f_k|$, the total volume of outgoing flows across the collaboration. As network sizes grow—or as operators deploy richer, higher-dimensional feature sets—the correlator faces increasing overheads in both memory footprint and processing time. High-throughput networks may produce millions of flows per second, quickly overwhelming even well-provisioned servers.

Furthermore, transporting all feature vectors to a single correlator introduces considerable bandwidth consumption and requires high-capacity links to avoid congestion. The centralization also creates a single point of failure and a performance bottleneck, where delays or outages at the correlator can disrupt the attribution for all participating networks.

In the next chapter, we present RevealNet, a distributed framework that addresses these limitations by decentralizing correlation, reducing communication overhead, and leveraging programmable data planes to pre-process flows closer to their points of observation.

Chapter 3

Methodology

This chapter introduces RevealNet, a distributed correlation framework aimed at enabling a coalition of operators managing a set of cooperating networks to identify the network location of an attack's perpetrator. RevealNet leverages P4 switches to simultaneously act as *probe nodes* (responsible for collecting flows' traffic features) and *correlator nodes* (responsible for computing correlation scores), thus efficiently distributing correlation workloads across the network. In contrast to conventional architectures that centralize flow-feature collection and correlation at one or a few backend servers, RevealNet embeds computation directly into the datapath devices already present in participating networks. This design minimizes data movement, reduces bandwidth consumption, and ensures that feature extraction and correlation occur as close as possible to the point of observation.

Next, we describe RevealNet's overall architecture before outlining its workflow and providing a comprehensive overview of its core design features.

3.1 Architecture Overview

Figure 3.1 illustrates the overall architecture of RevealNet. The framework comprises three key components that operate in coordination to perform distributed correlation for attack attribution. We detail each of these roles below.

Programmable switches P4 switches represent the central computational substrate for RevealNet, serving a dual purpose within the framework. First, as *probe nodes*, they extract flow-level features during normal packet-forwarding operations. Because these switches are

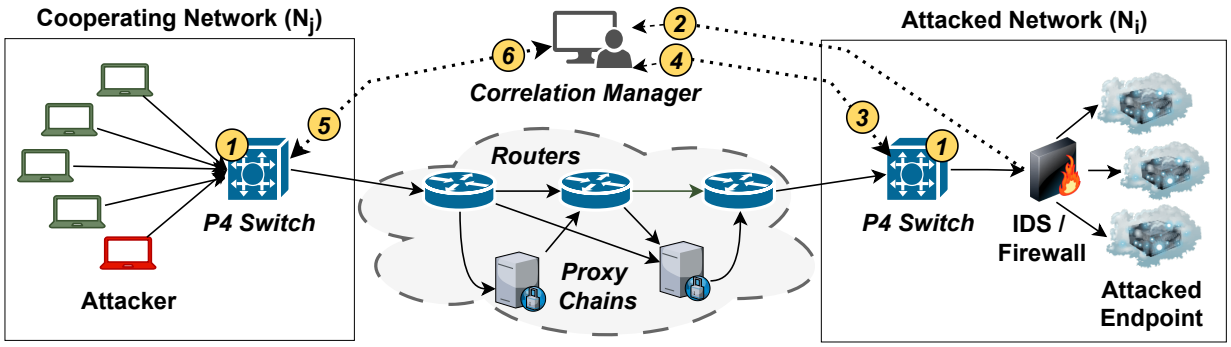


Figure 3.1: RevealNet's decentralized correlation architecture for attack attribution.

typically already deployed at the borders of participating networks (e.g., as top-of-rack or edge routers), RevealNet can reuse existing infrastructure to collect detailed traffic measurements without requiring additional hardware or changes to network topology. Through per-packet processing operations available in the data plane, P4 switches continuously aggregate features for active flows using register arrays and match–action logic (§2.3).

Second, these same switches also operate as *correlator nodes*. When instructed by the correlation manager, the switch's control plane retrieves the relevant feature vectors that have been computed and stored in the data plane. The control plane then performs similarity computations between the malicious flow features and features from locally observed flows. Performing correlation locally avoids exporting large feature sets to external servers, significantly reducing communication overhead. Moreover, keeping feature extraction and correlation tightly coupled at the switch improves timeliness: operators can begin correlating flows almost immediately after an IDS alert is raised.

Intrusion detection systems We assume that each cooperating network already operates one or more intrusion detection systems (IDSes)—including firewalls, signature-based IDS appliances, or ML-driven detectors deployed on middleboxes or servers. These systems continuously monitor incoming and outgoing traffic for suspicious or malicious activity. When an IDS detects an attack targeting assets within a network N_i, it issues an alert and passes a reference to the malicious flow to the correlation manager. This alert includes identifiers that allow the correlation manager to request the corresponding flow's features from the network's front-facing P4 switch. In this sense, the IDS acts as the trigger that initiates RevealNet 's distributed correlation workflow. Although the IDS itself does not participate in correlation, its timely detection is essential for ensuring that the offending flow's features can still be retrieved accurately from the switch before flow state expires.

Correlation manager The correlation manager serves as a lightweight orchestration layer that coordinates the distributed correlation process across all cooperating networks. Unlike the centralized correlator in traditional architectures, the manager does *not* perform any similarity computations. Instead, it acts as the control hub of the system. Upon receiving an alert from an attacked network's IDS, the manager retrieves the attacking flow's feature vector from that network's probe node and distributes this vector to all cooperating networks' correlator nodes. This distribution ensures that each network only receives the features necessary to perform local correlation tasks, rather than the full spectrum of flows observed across the coalition.

Because the correlation manager handles limited metadata and small feature vectors, it remains computationally lightweight and does not form a bottleneck. We envision this manager being operated by a neutral organization—such as a trusted consortium, an inter-organizational security alliance, or a national cybersecurity center [56]—to provide a governance layer that facilitates cooperation without revealing sensitive internal traffic data. Once each correlator node completes its local correlation computation, it reports back its results to the manager, which then aggregates them and forwards a final attribution report to the originally attacked network.

In summary, RevealNet distributes both data collection and correlation workloads across P4 switches deployed at cooperating networks, while relying on an IDS to initiate the process and a lightweight correlation manager to coordinate it. This combination enables a decentralized attribution mechanism that avoids the scalability and bandwidth limitations inherent to centralized analysis systems, paving the way for efficient cross-network collaboration in high-speed environments.

3.2 Operational Workflow

We now outline the operational workflow of our framework. Figure 3.1 illustrates each of the steps composing RevealNet's sequential workflow, highlighting how detection, feature extraction, and distributed correlation interact across the cooperating networks.

Flow features' extraction In the steady state, all probe nodes within cooperating networks continuously generate compact flow-level feature representations for every active flow traversing their respective switches (step **1**). These feature vectors capture temporal and behavioral characteristics that can later be used for flow correlation. Due to the stringent memory constraints of programmable switches, feature vectors are stored ephemerally and

may be evicted based on LRU-like policies as flows terminate or new flows arrive [3]. This enables the switch to sustain high-throughput environments without exceeding its register memory budget. Importantly, RevealNet does not mandate a single canonical representation: as discussed in §3.4, the underlying feature vector format is adaptable, allowing different designs to trade memory consumption for correlation fidelity depending on operator requirements.

Attack detection Each cooperating network deploys an IDS that monitors traffic for signs of malicious activity. Once the IDS in an attacked network N_i identifies an ongoing attack, it extracts the corresponding 5-tuple $\langle$*Src. IP, Dst. IP, Src. Port, Dst. Port, Proto*$\rangle$ to uniquely identify the offending flow. The IDS then forwards this tuple to the correlation manager (step 2). At this stage, no features are exchanged—only minimal metadata—ensuring that the initial alerting process remains lightweight and does not incur unnecessary bandwidth or computational overhead.

Request of attacking flow's features Upon receiving the alert and the 5-tuple identifiers, the correlation manager proceeds to obtain the attacking flow's feature vector. It issues a request to the probe node (i.e., the P4 switch) in the attacked network for the flow's feature summary stored within its data plane registers (step 3). Since the switch continuously maintains up-to-date feature vectors during steady-state operation, it can promptly retrieve the feature vector associated with the specified 5-tuple, assuming the flow is sufficiently recent and has not been evicted.

Propagation of the attacking flow's features Following the correlation manager's request, the probe node in the attacked network returns the attacking flow's feature vector (step 4). This vector contains a compact but representative snapshot of the flow's characteristics, serving as the reference point for the distributed correlation process. The correlation manager stores this vector temporarily and uses it solely to coordinate the subsequent attribution steps. No raw packet traces or large datasets are exchanged, preserving both scalability and privacy.

Correlation directive Once the correlation manager obtains the attacking flow's features, it dispatches a correlation directive to all cooperating networks. In this step, the attacking flow's feature vector is transmitted to each P4 switch that will serve as a correlator node (step 5). Alongside the feature vector, the correlation manager also distributes key metadata about the attacking flow—such as approximate start time, end time, and

total communication volume. These metadata help correlator nodes proactively filter their local flow records to identify which flows are temporally plausible or behaviorally compatible candidates for correlation. This pre-filtering step is crucial to reducing the search space and improving efficiency, especially in high-throughput environments with many concurrent flows. After disseminating these inputs, the correlation manager instructs each correlator node to initiate its local correlation routine (§3.5, §3.4).

Flow correlation and reporting Each correlator node computes similarity scores between the attacking flow and its own locally observed flows. Flows whose correlation scores exceed a configurable threshold are flagged as potential matches. These matches, represented by their associated 5-tuples and any relevant metadata, are then reported back to the correlation manager (step 6). The correlation manager aggregates the results from all cooperating networks, enabling the attacked network to infer the most probable origin of the malicious activity. At this point, no high-volume data is exchanged—only concise results—which ensures the attribution process remains efficient and scalable even as the number of cooperating networks grows.

Overall, this workflow enables RevealNet to distribute both data-plane feature extraction and control-plane correlation across multiple networks, significantly reducing centralized bottlenecks while preserving the accuracy required for attack attribution.

3.3 Flow Identification and Tracking

A key operation underlying the execution of RevealNet is that of performing efficient and reliable flow identification within P4 switches, so that accurate per-flow feature vectors can be computed and stored in the switch's data plane without disrupting line-rate forwarding. While conceptually simple, achieving robust flow identification in programmable switches is challenging due to strict memory constraints, limited computational primitives, and the inability to dynamically allocate state. Prior work demonstrates these limitations clearly. Existing ML-based flow analysis systems for the data plane frequently sidestep the problem of unique flow identification altogether, leaving the task to an unspecified mechanism external to the switch (e.g., [3]). Others require that the first packet of every new connection be punted to the control plane for classification or mapping (e.g., [61]), which imposes significant latency at flow establishment. In high-speed networks—particularly those requiring microsecond-level responsiveness—these delays are unacceptable.

Moreover, naive flow indexing strategies that rely on hashing the 5-tuple directly into a register array suffer from collisions and potential state corruption [19, 24]. Collisions not

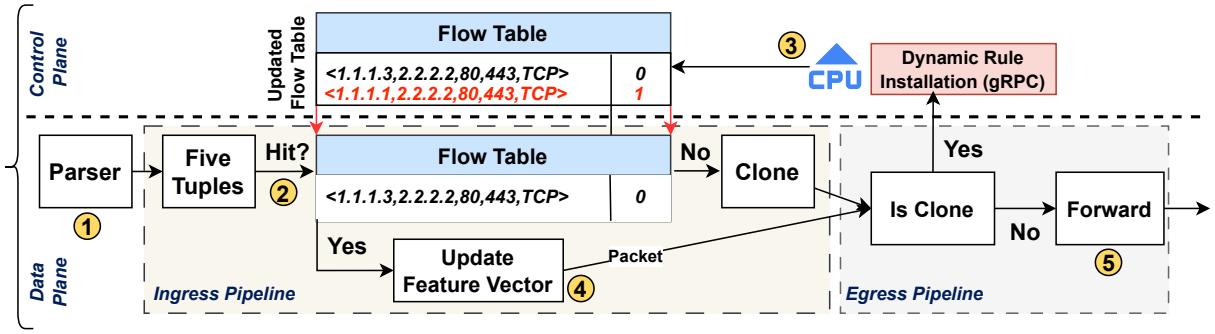


Figure 3.2: RevealNet's dynamic flow identification and matching mechanism.

only degrade accuracy but may compromise downstream correlation results, making precise flow-to-state mapping essential for the correctness of RevealNet. The challenge, therefore, is to construct a mechanism capable of establishing a one-to-one association between each active flow and a dedicated region of memory in the feature table, all while preserving line-rate processing and ensuring that no packets are delayed or misclassified during the identification phase.

RevealNet's flow identification pipeline We now detail how RevealNet addresses this challenge through a carefully designed identification and matching pipeline that separates fast-path forwarding from control-plane coordination, guaranteeing accurate state allocation without compromising performance. Figure 3.2 illustrates the sequence of steps that compose this mechanism.

- *Packet parsing.* The switch's packet-processing pipeline begins with parsing, during which the packet header fields comprising the 5-tuple are extracted (step 1). This tuple forms the flow's unique identifier and is passed downstream as metadata to subsequent match–action stages. Parsing occurs entirely in the data plane and does not impact forwarding latency.
- *Flow table lookup.* After parsing, the switch checks whether the extracted 5-tuple corresponds to an existing flow (step 2). To this end, RevealNet introduces a dedicated *flow table* that maps flow identifiers to indices in the *feature table*. The feature table is a logical organization of the switch's register memory into rows, each storing the compact feature vector of a distinct active flow. If the flow table contains a matching entry, the switch retrieves the corresponding index and updates the feature vector for the flow. If the entry does *not* exist, the switch must allocate a new row for this previously unseen flow.
- *Packet cloning and rule installation.* Flow table updates require control-plane involvement, as P4 programs cannot insert new table entries autonomously. To ensure

that packet forwarding remains uninterrupted, RevealNet performs a selective packet cloning operation (step 3). The first packet of a new flow is forwarded along the normal data path without delay, while a clone of the packet is sent to the control plane via a designated clone session. This clone triggers the installation of a new entry in the flow table through a remote procedure call, binding the flow's 5-tuple to an unused position in the feature table.
It is important to emphasize that only the first packet of a new flow initiates this process. Any additional packets that arrive before the rule is installed are intentionally ignored for feature updates: they are forwarded but do not contribute to the flow's feature vector. This prevents race conditions or accidental writes to undefined memory locations. The resulting behavior ensures that flows are consistently and correctly mapped to their feature rows, without introducing stalls or back-pressure into the data plane. Although initial packets may be excluded from feature computation, our experiments show that this omission has negligible influence on downstream correlation accuracy due to the temporal length and richness of typical flows in attribution scenarios.

- *Feature vector updates and packet forwarding.* Once the flow table entry has been installed, subsequent packets belonging to that flow are processed normally, and the switch updates the corresponding row of the feature table upon each packet arrival (step 4). These updates accumulate information such as inter-arrival timing indicators, packet size summaries, and counts, depending on the compact representation employed. After executing the feature-update logic, the packet resumes its traditional forwarding path and is transmitted out the appropriate egress port according to the IPv4 forwarding table (step 5).

Overall, RevealNet 's flow identification and tracking pipeline achieves a critical balance between precision and performance. It ensures that flows are assigned unique, collision-free state in the feature table while avoiding the pitfalls of naive hash-based indexing or blocking operations that would otherwise slow packet processing. By limiting the control-plane interaction to a single cloned packet per flow and ensuring that the main pipeline remains non-blocking, RevealNet provides a practical and scalable mechanism for high-speed, large-scale flow tracking in programmable data plane environments.

3.4 Sketching and Correlation

Traffic correlation techniques rely on a set of flow features that are derived from per-packet observations yet remain largely invariant to transient network conditions. Common

examples include packet arrival timestamps, packet sizes, and aggregate communication volume. While such fine-grained information is valuable for correlation, retaining per-packet state for each active flow inside a programmable switch's data plane is impractical: the resulting memory footprint would severely limit the number of concurrent flows that can be tracked and correlated by RevealNet at any given time.

To address this challenge, RevealNet adopts compact, flow-level feature representations that summarize packet dynamics over time while remaining amenable to efficient in-switch processing. We build upon existing approaches originally proposed for traffic correlation and related ML-based security tasks, adapting them to the architectural constraints of modern programmable switches.

3.4.1 Traffic aggregation matrix

As a first step, we adopt a traffic aggregation matrix (TAM) representation [52]. In this model, packets belonging to a flow are grouped into consecutive time bins of fixed duration t, for a total observation window of T seconds. For each bin, the TAM records summary statistics of the traffic observed during that interval. In RevealNet, we follow prior work [35] and construct a single-row TAM per flow, where each entry stores the number of packets transmitted during the corresponding time bin. This yields a fixed-length feature vector that captures coarse-grained temporal patterns while consuming substantially less memory than storing individual packet metadata.

Although TAMs already provide a compact abstraction of flow behavior, storing full TAM vectors for a large number of active flows may still strain the limited SRAM available in programmable switches. As discussed in §4.3, this trade-off motivates the need for further compression mechanisms that preserve the discriminative structure of the feature vectors while minimizing per-flow state.

3.4.2 Flow sketching

To further reduce the memory footprint of TAM feature vectors, RevealNet employs sketching techniques based on random projection methods [15, 42]. Let the TAM feature vector of a flow be denoted by

$$\mathbf{f} = [f_1, f_2, \ldots, f_n],$$

where n is the number of time bins. Sketching algorithms transform $\mathbf{f}$ into a lower-dimensional representation

$$\mathbf{f}_c = [f_{c1}, f_{c2}, \ldots, f_{cm}],$$

with $m \ll n$, such that key similarity relationships between flows are approximately preserved in the compressed domain.

A key advantage of sketching in the context of programmable switches is that sketches can be maintained *online*: rather than materializing the entire TAM and compressing it afterward, the sketch is incrementally updated as packets arrive. This enables per-packet processing with constant-time updates and avoids storing high-dimensional vectors in the data plane. The sketch dimensionality m serves as a tunable parameter, allowing RevealNet to trade memory consumption and computational cost against correlation accuracy.

Coskun et al. [15]. propose an online sketching technique that operates over a packet-count-based TAM representation. Traffic is first binned into discrete time slots, yielding a conceptual packet-count vector. Rather than explicitly storing this vector, the method applies a linear random projection as packets arrive, generating a compact sketch without ever materializing the full TAM.

Formally, let $B \in \{-1, +1\}^{m \times n}$ denote a random projection matrix whose entries are independently drawn from a Bernoulli (Rademacher) distribution. For a given flow, the sketch vector $\mathbf{s} \in \mathbb{Z}^m$ is updated online according to

$$s_k \leftarrow s_k + B_{k,j},$$

for all $k \in \{1, \ldots, m\}$, whenever a packet arrives in time bin j. Each packet therefore contributes a constant-time update to every sketch dimension, and the cumulative effect of all packets in a bin corresponds to projecting the packet-count vector along the columns of B.

Data-plane realization In RevealNet, we implement this sketching mechanism directly in the P4 data plane. Each flow maintains a per-flow bin pointer j, which identifies the active TAM bin, and a per-flow timestamp used to advance the bin when the elapsed time exceeds a fixed bin width Δ. Upon packet arrival, the switch increments the packet counter of the current bin and updates all m sketch entries using a single integer add or subtract operation per entry, depending on the corresponding projection matrix value.

To ensure data-plane feasibility, the projection matrix B is stored as a register-backed table whose entries are encoded as single bits: a value of 1 represents $+1$, while a value of 0 represents -1. This encoding allows each sketch update to be implemented using only integer arithmetic and register accesses, operations that are well supported by commodity P4 targets. The projection matrix is generated once in the control plane and installed identically across cooperating switches to ensure consistency during correlation.

Algorithm 1 Instantiate and install Bernoulli projection matrix for Coskun-style sketching (control plane).

Require: n: number of time bins (PCV length), m: sketch length
1: **for** $k \leftarrow 0$ to $m-1$ **do**
2: **for** $j \leftarrow 0$ to $n-1$ **do**
3: Sample $B_{k,j} \in \{-1, +1\}$ uniformly at random
4: Encode $B_{k,j}$ as a bit: $\text{enc}(+1) = 1$, $\text{enc}(-1) = 0$
5: WriteRegister(`B_table`, $k \cdot n + j$, $\text{enc}(B_{k,j})$)
6: **end for**
7: **end for**
8: Install the same `B_table` at all cooperating switches

Binary sketches and correlation metric Following Coskun et al., RevealNet additionally supports a binarized variant of the sketch to reduce per-flow state. After updating the integer sketch vector, each sketch coordinate may be mapped to a single bit via a sign (or threshold) function, yielding $\mathbf{b} \in \{0,1\}^m$. In this setting, flow correlation is performed using Hamming distance:

$$d_H(\mathbf{b}^{(1)}, \mathbf{b}^{(2)}) = \sum_{k=1}^{m} \mathbb{I}\left[b_k^{(1)} \neq b_k^{(2)}\right],$$

where $\mathbb{I}[\cdot]$ is an indicator function. Smaller values of d_H indicate higher similarity between the flows' sketches. As shown in §4.3, this binary sketch option provides a favorable memory–accuracy trade-off in high-scale deployments, while still enabling efficient similarity computation. Algorithm 1 summarizes the control-plane initialization of the Bernoulli projection matrix implemented by RevealNet for Coskun-style sketching.

Nasr et al. [42]. propose a compressive-sensing-based correlation mechanism in which a high-dimensional traffic feature vector is compressed into a lower-dimensional sketch using a sensing matrix $\Phi \in \mathbb{R}^{m \times n}$. Under appropriate conditions on Φ (e.g., Restricted Isometry Property-like behavior), distances and similarity relationships between original vectors are approximately preserved in the compressed domain, enabling correlation to be performed directly on the sketches.

In the original formulation, a flow feature vector $\mathbf{f} \in \mathbb{R}^n$ is compressed as:

$$\mathbf{y} = \Phi \mathbf{f},$$

where $\mathbf{y} \in \mathbb{R}^m$ is the sketch. In RevealNet, we apply this idea to packet-count TAM vectors and adapt it to an *online* in-switch computation model. Specifically, we maintain a per-flow time-bin pointer j and update the compressed sketch incrementally on each packet arrival, without materializing the full $\mathbf{f}$. For a packet arriving in time bin j, the data plane updates each sketch coordinate k as:

$$y_k \leftarrow y_k + \Phi_{k,j}.$$

Over the lifetime of the flow, the cumulative effect of these per-packet updates yields a sketch equivalent to applying Φ to the flow's packet-count vector.

Data-plane realization and online updates Our P4 implementation closely mirrors the above update rule. Each flow maintains (i) a TAM-style packet counter for the current bin, (ii) an m-dimensional integer sketch, and (iii) lightweight timing metadata used to advance the active time bin when the elapsed time exceeds a fixed bin width Δ. Upon each packet arrival, the switch reads $\Phi_{k,j}$ from a register-backed sensing table and adds it to the corresponding sketch coordinate, performing only integer additions and register reads/writes. As in the Coskun realization, the bin pointer j is advanced when $(t-\tau) \geq \Delta$, where t is the packet arrival timestamp and τ is the per-flow last-bin timestamp.

Sensing matrix instantiation and quantization A practical challenge is that P4 targets do not support floating-point arithmetic, whereas sensing matrices Φ are typically defined over real numbers (e.g., i.i.d. Gaussian entries). To address this, RevealNet generates Φ in the control plane as a Gaussian random matrix and normalizes it (in our prototype, by $1/\sqrt{m}$) to improve stability. The resulting real-valued entries are then quantized to integers and installed into the switch as a register array.

Concretely, our prototype applies an affine scaling to map Φ's entries into an integer range supported by the target registers. Let α denote a scale factor and β an offset (derived from the observed minimum value), and let $\Phi_{k,j}$ be a real-valued entry. We store:

$$\widetilde{\Phi}_{k,j} = \lfloor \alpha(\Phi_{k,j} - \beta) \rceil \,,$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. When sketches are read out for correlation, the control plane can approximately recover the real-valued domain by inverting the mapping, i.e., $\widehat{\Phi}_{k,j} \approx \widetilde{\Phi}_{k,j}/\alpha + \beta$, and applying the corresponding rescaling to the aggregated sketch values. This design preserves data-plane feasibility while retaining the structure required for similarity comparison.

Algorithm 2 Instantiate and install sensing matrix for Nasr-style sketching (control plane).

Require: n: number of time bins, m: sketch length
Require: R: maximum representable integer (register range)
Sample $\Phi \in \mathbb{R}^{m \times n}$ with i.i.d. Gaussian entries
Normalize $\Phi \leftarrow \Phi/\sqrt{m}$
Compute $\beta \leftarrow \min(\Phi)$, $\alpha \leftarrow R/(\max(\Phi) - \min(\Phi))$
for $k \leftarrow 0$ to $m-1$ **do**
    for $j \leftarrow 0$ to $n-1$ **do**
        $\widetilde{\Phi}_{k,j} \leftarrow \lfloor \alpha(\Phi_{k,j} - \beta) \rceil$
        WriteRegister(`Sensing_table`, $k \cdot n + j$, $\widetilde{\Phi}_{k,j}$)
    end for
end for
Store (α, β) in the controller for later sketch decoding/comparison

Correlation metric Consistent with Nasr et al. [42], correlation in RevealNet compares sketches using cosine similarity. Given two sketches $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$, cosine similarity is computed as:

$$\cos(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}) = \frac{\mathbf{y}^{(1)} \cdot \mathbf{y}^{(2)}}{\|\mathbf{y}^{(1)}\|_2 \, \|\mathbf{y}^{(2)}\|_2}.$$

In our prototype, this computation is performed in the control plane after reading per-flow sketches from the switch registers. The sketch length m remains a tunable parameter controlling memory footprint and correlation fidelity. Algorithm 2 summarizes the control-plane sensing-matrix installation implemented by RevealNet for the Nasr-style sketch.

We next examine the memory footprint imposed by the different feature representations supported by RevealNet, with the goal of quantifying their suitability for deployment on programmable switches with limited SRAM. In particular, we compare the storage requirements of full TAM feature vectors against those of the sketch-based representations described above, and further account for the additional per-flow metadata required by RevealNet 's heuristic optimizations. Table 3.1 summarizes the storage requirements associated with each feature representation considered in RevealNet, expressed in terms of the number of concurrent flows f, the TAM length n, and the sketch dimensionality m. The table highlights the substantial memory savings achieved by sketch-based representations when compared to storing full TAM feature vectors, particularly in scenarios with a large number of active flows. For both Coskun and Nasr sketches, the dominant static cost arises from storing the projection or sensing matrix, which requires $n \times m$ integers and is shared across all flows observed by a switch. This cost is incurred once and amortized

Table 3.1: Storage requirements in terms of f (number of flows), n (TAM feature vector length), and m (sketch length). Each integer is assumed to be 32 bits.

Method / Storage	**Proj. Matrix**	**Flows**	**Total (bits)**	**Total (bits) w/ heur. (§3.5)**
Nasr (integer sketch)	$n \times m$ (integers)	$f \times m$ (integers)	$32\,(n \times m + f \times m)$	$32\,(n \times m + f \times m + f) + 48 \times f$
Coskun (integer sketch)	$n \times m$ (integers)	$f \times m$ (integers)	$32\,(n \times m + f \times m)$	$32\,(n \times m + f \times m + f) + 48 \times f$
Coskun (binary sketch)	$n \times m$ (integers)	$f \times m$ (bits)	$32\,(n \times m) + f \times m$	$32\,(n \times m + f) + f \times m + 48 \times f$
TAM feature vector	–	$f \times n$ (integers)	$32\,(f \times n)$	$32\,(f \times n + f) + 48 \times f$

across the lifetime of the system. In contrast, per-flow storage scales only with the sketch length m, yielding a total per-flow cost of $f \times m$ integers for integer sketches. As a result, sketch-based approaches reduce per-flow memory from $O(n)$ to $O(m)$, enabling switches to track and correlate significantly more flows within the same SRAM budget. Among the sketch variants, the Coskun binary sketch offers the lowest per-flow storage overhead, as each sketch coordinate is stored as a single bit. This representation eliminates the need for per-flow integer sketch storage, at the expense of reduced fidelity in the compressed representation. Nevertheless, as shown in §4.3, binary sketches retain sufficient discriminatory power for effective correlation while enabling the largest scaling factor in terms of concurrent flows. The final column of Table 3.1 accounts for the additional metadata required by RevealNet 's heuristic optimizations discussed in §3.5.

3.4.3 Correlation

The final stage of RevealNet 's attack attribution pipeline correlates flow feature representations collected at different vantage points in order to determine whether they originate from the same source. Correlation is performed by computing a similarity or distance metric directly in the compressed feature space, avoiding the need to reconstruct full TAM vectors. The choice of similarity metric is dictated by the sketching method in use. For Coskun-style binary sketches, RevealNet employs Hamming distance, which counts the number of mismatched bits between two sketches. This metric is computationally inexpensive and well suited to compact binary representations. Flows exhibiting similar packet-timing patterns produce sketches with small Hamming distances, while unrelated flows tend to differ across many sketch dimensions. For Nasr-style sketches, RevealNet follows the original design and compares sketches using cosine similarity Cosine similarity captures the angular alignment between compressed vectors and is robust to variations in overall traffic volume, making it well suited to real-valued compressive-sensing sketches. Higher cosine similarity values indicate stronger correspondence between flows.

In RevealNet, sketch comparison is performed in the control plane after retrieving per-

flow sketches from switch registers. This design avoids placing computationally expensive similarity operations in the data plane, while still allowing switches to perform all sketch construction and state maintenance at line rate. By combining compact sketch representations with appropriate similarity metrics, RevealNet achieves scalable and accurate correlation even in environments with large numbers of concurrent flows.

3.5 Heuristic Optimizations for Attack Attribution

While the sketch-based correlation mechanisms described above form the core of RevealNet's attribution capabilities, cooperating networks typically observe an extremely large number of unrelated flows. Attempting to correlate the attacking flow against *all* outgoing flows at each correlator node would not only be computationally expensive but would also increase the likelihood of spurious matches, a challenge repeatedly documented in prior research [15, 42]. To mitigate unnecessary comparisons and reduce false-positives, RevealNet incorporates two lightweight heuristic optimizations, first proposed in the context of anonymity-flow detection [7], and adapts them to the programmable-switch setting.

These heuristics operate as a pre-filtering step before the sketch-based similarity computation. Their purpose is to rapidly eliminate flows that are obviously incompatible with the attacking flow due to clear temporal or volumetric discrepancies. Both heuristics are evaluated locally at each correlator node and are designed to work efficiently with the metadata maintained in the switch's data plane.

3.5.1 Creation time heuristic

The first heuristic is based on temporal alignment. A malicious flow observed at network N_i must correspond to an upstream flow that occurred within a reasonably bounded time window. If a flow at a cooperating network was created far earlier or far later than the attacking flow, it cannot be a matching candidate.

Let t_m denote the start time of the attacking flow as measured at the attacked network. We define a temporal acceptance window $[T_{\min}, T_{\max}]$, where these bounds are offsets relative to t_m. For each locally stored flow with creation timestamp t_f, we retain it only if:

$$T_{\min} \leq t_f - t_m \leq T_{\max}.$$

A flow outside this window is excluded from consideration. Since each switch maintains its flows in timestamp-sorted order, identifying the relevant subset requires only a small

number of comparisons and enables efficient filtering. This temporal heuristic typically eliminates a majority of unrelated flows early in the pipeline.

3.5.2 Packet count heuristic

Temporal proximity alone is insufficient to guarantee relevance: many flows may begin around the same time but exhibit vastly different traffic volumes. The second heuristic therefore filters flows based on their aggregate packet counts, which also serve as a rough behavioral signature for correlation tasks.

Suppose the attacking flow observes a total of p_m packets. We define acceptable packet-count bounds $P_{\min}$ and $P_{\max}$, which may be derived using simple multiplicative tolerances or percentile-based deviation rules. A candidate flow with packet count p_f is retained if:

$$P_{\min} \leq p_f \leq P_{\max}.$$

This comparison is performed directly using the data plane's per-flow packet count register. The packet count heuristic further prunes flows that are unlikely to exhibit similar communication patterns, shrinking the candidate set before the more expensive sketch-comparison stage is invoked.

3.5.3 Sequential application and complexity reduction.

RevealNet applies these two heuristics sequentially. Timestamp filtering produces a subset f_t of flows satisfying the temporal constraint; because flows are pre-sorted by creation time, the cost of identifying f_t is $O(\log(|\mathcal{F}|))$, where $|\mathcal{F}|$ denotes the total number of outgoing flows at the correlator node. Packet-count filtering is then applied over this reduced set, requiring $O(|f_t|)$ time and yielding the set f_{t+p}. Sketch-comparison is finally applied to this set at cost $O(|f_{t+p}| \times C)$, where C is the cost of a single similarity computation. Thus, the baseline $O(|\mathcal{F}| \times C)$ correlation cost is reduced to $O(\log(|\mathcal{F}|) + |f_t| + |f_{t+p}| \times C)$, a significant improvement when the heuristics aggressively prune the initial flow space.

Switch-level implementation To support these heuristics, RevealNet maintains auxiliary per-flow metadata in the data plane: each flow record includes a 48-bit creation timestamp and a 32-bit packet-count accumulator. This results in a storage cost of $48f + 32f$ bits, where f is the number of flows observed by the switch. As summarized in Table 3.1, this overhead is modest relative to sketch-storage requirements and fits comfortably within modern programmable-switch SRAM budgets. Both heuristics require only integer comparisons and register lookups, operations easily handled.

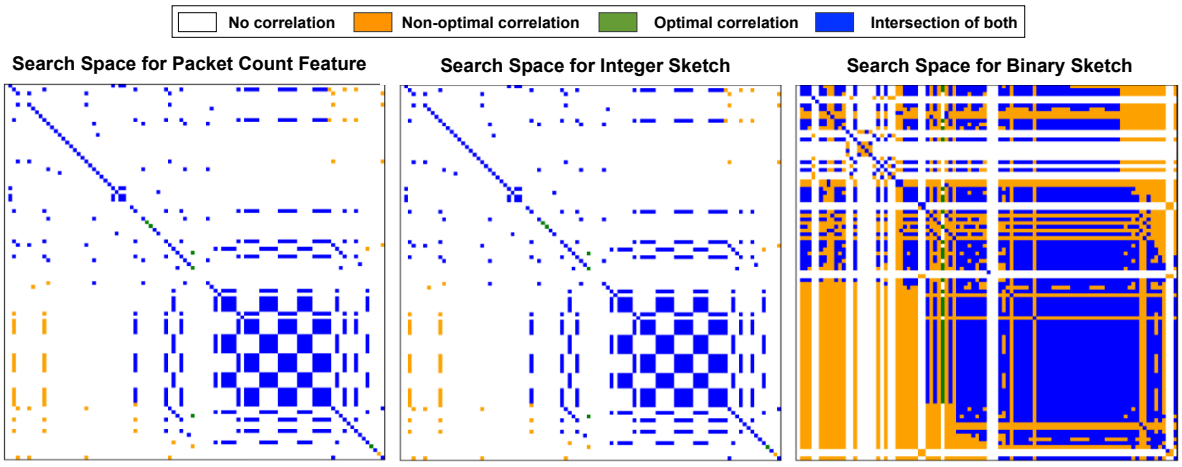


Figure 3.3: Illustration of RevealNet's heuristic impact on the Coskun Sketch, including Packet Count features (TAM), Integer Sketches, and Binary Sketches.

Impact of heuristics The effectiveness of the proposed heuristics is further illustrated in Figure 3.3, which visualizes the correlation search space for Coskun-style sketches under different feature representations. Each subplot shows pairwise correlation outcomes between candidate flows, color-coded according to the legend: white indicates pairs with no detected correlation; orange denotes correlations obtained without heuristics (non-optimal correlations); green corresponds to correlations obtained with heuristics enabled (optimal correlations); and blue represents the intersection of the two, i.e., flow pairs that are correlated both with and without heuristics. The figure is generated with both temporal and packet-count heuristics applied.

Across all three representations—packet-count (TAM) features, integer sketches, and binary sketches—the application of heuristics leads to a substantial reduction in false correlation candidates. The effect is even more pronounced for binary sketches (right plot). Due to their highly compressed representation, binary sketches exhibit a broad and dense set of correlations when heuristics are not applied, reflected by the extensive orange regions. Blue regions of Figure 3.3 shows that the combined use of temporal and packet-count heuristics dramatically suppresses this noise, collapsing the search space into a much smaller region in which true correlations dominate. As a result, the number of incorrect matches is significantly reduced, despite the aggressive compression imposed by binary sketches.

This observation is particularly important from a systems perspective. Binary sketches offer the lowest memory footprint and are therefore the most attractive option for deployment on memory-constrained programmable switches. The figure demonstrates that, when

coupled with lightweight heuristics, binary sketches can achieve better correlation quality comparable to binary sketches without heuristics.

3.5.4 Implementation

We built a prototype of RevealNet using `bmv2` [49], the reference P4 software switch. The data plane logic, including flow identification and sketching operations (for either sketch), was implemented in ∼500 lines of P4_16 code. In turn, RevealNet's control plane logic was written in ∼300 lines of Python code. This includes the installation of tables and rules supporting flow identification and sketching operations in the data plane, as well as fetching flows' feature vectors via reads to data plane registers for enabling the correlation backbone.

Chapter 4

Evaluation

4.1 Evaluation Goals and Metrics

We wish to evaluate the practicality of RevealNet along three key dimensions:

Effectiveness We evaluate RevealNet's attack attribution capability by measuring its correlation accuracy on malicious flows, using metrics aimed to capture the trade-off between successful correlations and incorrectly matched flows [41].

The *true positive rate* (TPR) measures the fraction of attacking flows that are correctly correlated by the system. Let f_m^s denote the number of malicious flows originated within a network with border switch s, and TP^s the number of those that are correctly matched to the malicious flows detected within a cooperating network under attack. Across all switches $\mathcal{S}$, $\text{TPR} = \sum_{s\in\mathcal{S}} TP^s \big/ \sum_s f_m^s$.

The *false positive rate* (FPR) captures incorrect correlations. Let f^s denote the total number of flows originated within a network with border switch s, and FP^s the number of such flows that are incorrectly matched to malicious flows f_m detected within an attacked network. Then $\text{FPR} = \sum_{s\in\mathcal{S}} FP^s \big/ \sum_s f_m f^s$, where the denominator reflects all potential false-positive pairs across all switches.

Efficiency We quantify the computational cost associated with flow correlation via the number of pairwise comparisons between detected malicious flows and the outgoing flows observed by cooperating networks. The computational effort across all switches $\mathcal{S}$ can be

Table 4.1: Summary of malicious/benign traffic datasets used in our experiments.

Dataset	Category	Description	Flows		Duration (s)
			Benign	Malicious	
Snojan	Botware	PPI malware downloading.	206 723	1 607	45.64
Dridex	Ransomware	Victim locations uploading.	125 424	3 202	54.75
Adload	Adware	Static resources for PPI adware.	125 417	602	54.80
Oracle	Web	TLS padding Oracle.	294 110	224	64.14
Penetho	Spyware	Wifi cracking APK spyware.	293 808	1 006	55.64
Bitcoinminer	Miner	Abnormal encrypted channels.	125 418	202	61.01

expressed as the sum of the pairwise comparisons for each switch: $\sum_{s \in \mathcal{S}} f_m \times f^s$, where f_m represents the number of malicious flows detected and f^s represents the number of outgoing flows observed by switch s.

Scalability We evaluate RevealNet' scalability by analyzing two key factors: a) the number of flows that can be concurrently stored and processed, and; b) the communication overhead required during attack attribution. Let f^s denote the number of outgoing flows observed by a cooperating network's switch, and let C^s represent the total communication cost (in bits) for transmitting these flows' features'. If each flow is represented by a feature vector of size m bits, then transmitting all flows f^s incurs a communication cost of $C^s = f^s \times m$.

4.2 Evaluation Methodology

We now describe the datasets used in our evaluation, followed by details on the parameterization of RevealNet's data structures and heuristics.

Datasets used for attack attribution We use six labelled network traffic datasets, released by Fu et al. [22], as a target of RevealNet' attack attribution capabilities. These datasets were compiled from a combination of Fu et al.'s own experimental data and traffic traces from the WIDE MAWI project in Tokyo, Japan. For exercising RevealNet's generalizability, we selected each dataset from six different categories of attacks collected by Fu et al. (see Table 4.1 for an overview of each dataset). Each of the datasets contains a different type of network attack along with background benign traffic. Fu et al.'s data records detail per-packet five-tuples in `.csv` files, along with packets' timestamps and labels (benign/malicious), allowing us to carve out individual flows identified by these 5-tuples.

Each dataset accounts for more than 100k flows, and the ratio of benign to malicious traffic is at least 39:1 (Dridex) and at most 1312:1 (Oracle).

For simplicity, we assume that the network IDS deployed within each RevealNet-enabled network acts as an oracle that can perfectly distinguish between benign and malicious flows. While this is within the realm of practicality for the datasets we considered [22], we recall that our goal is not to perform accurate malware classification, but rather to act on the alerts produced by IDSes (§2.4).

Emulating vantage points and network conditions Since the above datasets were collected at a single network vantage point and do not include raw packet traces that can be transparently replayed across some network topology (real or emulated) by special-purpose software such as `tcpreplay`, they cannot be directly used for correlation experiments across different networks, as required by RevealNet. To tackle this issue, and similarly to [15], we simulate the acquisition of two separate observations for each flow at different vantage points within RevealNet-enabled networks: a) at the border router of a cooperating network where hosts originate benign/malicious traffic, and; b) at the border router of a network which is targeted by some attack. We also assume that all flows in each of Fu et al. [22]'s datasets originate from a cooperating network and traverse (or target devices within) the attacked network. To facilitate this setup, we implemented a simulator that models WAN traffic being relayed via an intermediate proxy node. We produce a second set of datasets that represent traffic observations across the WAN, where packets experience an average latency increase of ∼200ms between any two vantage points. The simulator also supports the injection of perturbations, e.g., packet loss, allowing us to assess the robustness of flow correlation under degraded network conditions.

Parameterization of RevealNet's sketches Each dataset from Fu et al. [22] spans 45–65s of traffic. To explore the impact of temporal granularity in flow feature collection, we generated TAM time bins (t) of 0.1s, 0.5s, and 1s. We performed preliminary experiments using different sketch lengths ($m = 5, 10, 15$), keeping $m = 10$ as a baseline. A sketch length of 5 slightly improved TPR by up to +1.52% but at the cost of a substantial increase in FPR, reaching +114.84%. Conversely, using $m = 15$ provided no consistent TPR gains and resulted in mixed FPR outcomes (ranging from –20.4% to +92.65%), along with added storage overhead. Overall, for all datasets we considered, $m = 10$ strikes a favourable balance between accuracy and efficiency (see §4.3). Still, as previously studied [15], this parameter may need to be tuned for flows with different traits.

We follow the original methodology of each sketch to compute correlation scores. For

Coskun et al.'s sketch, we use Hamming distance and consider a match to be a true positive only when the distance between sketches is 0. For Nasr et al.'s sketch, we use cosine similarity, requiring a score of 1 for an exact match. We evaluate correlation using TAMs with both Hamming distance and cosine similarity, applying the same thresholds to define true positives. We adopt these thresholds to reflect high-confidence matches in attack attribution, where false associations can be especially harmful to benign users. Indeed, we experimented with relaxed correlation thresholds across various time bins t, but found that these looser criteria offered only marginal improvements in true positive rates while significantly increasing false positives. For instance, for Nasr et al.'s sketches' setups, a cosine similarity threshold of 0.9 yielded modest increases in TPR (up to 2.54%) but introduced significantly higher FPR (up to 373.04%).

Configuration of RevealNet's heuristics Network topology and flows' characteristics can significantly influence the choice of RevealNet's heuristics' configurations [7]. However, since the datasets we use to evaluate RevealNet (see Table 4.1) share similar traits on flow durations, we configure our heuristics to be consistent across all datasets. For the timing-based and packet count heuristics, we empirically found that a window of ± 2.5 seconds and a threshold of $\pm 5\%$ traffic volume filters out irrelevant candidate flows while retaining most valid matches. As described in §4.3, applying these heuristics to the `bitcoinminer` dataset helps reduce the number of comparisons per attacking flow by 3 orders of magnitude, without substantially sacrificing correlation accuracy.

4.3 Evaluation Results

We now describe the main results obtained during our evaluation of RevealNet. Throughout this section, we centre on the `bitcoinminer` dataset since it: a) has a representative benign-to-malicious flow ratio of 625:1, close to the median of all datasets; b) exhibits the second-largest temporal span, and; c) has a relatively smaller number of flows, facilitating faster experimentation. Results for other datasets, which exhibit similar trends, are are also discussed in brief.

Compact sketches obtain correlation accuracy equivalent to TAMs Figure 4.1 shows the TPR and FPR achieved by different parameterizations of the TAM and sketching approaches considered in our work, when correlating flows pertaining to the `bitcoinminer` dataset. The figure suggests that sketches attain a comparable correlation performance vs.

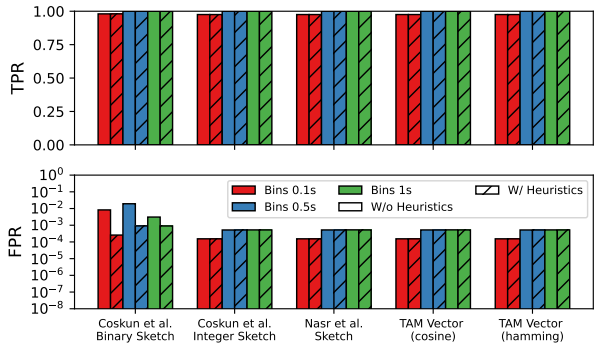


Figure 4.1: Bitcoinminer (unperturbed).

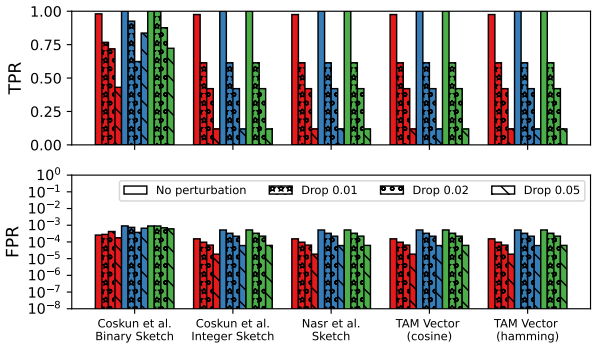


Figure 4.2: Bitcoinminer (w/perturbations).

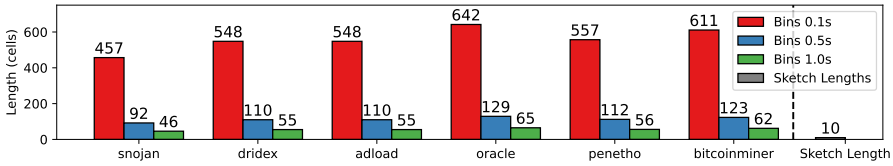
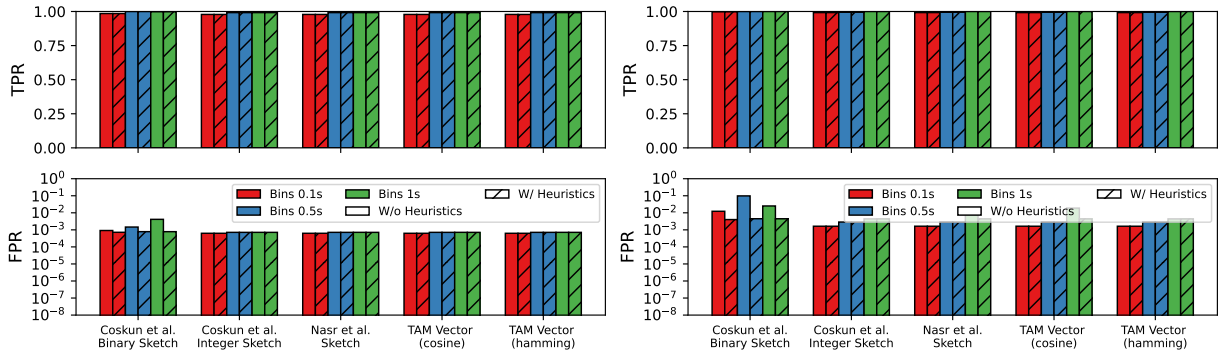


Figure 4.3: Length of TAM.

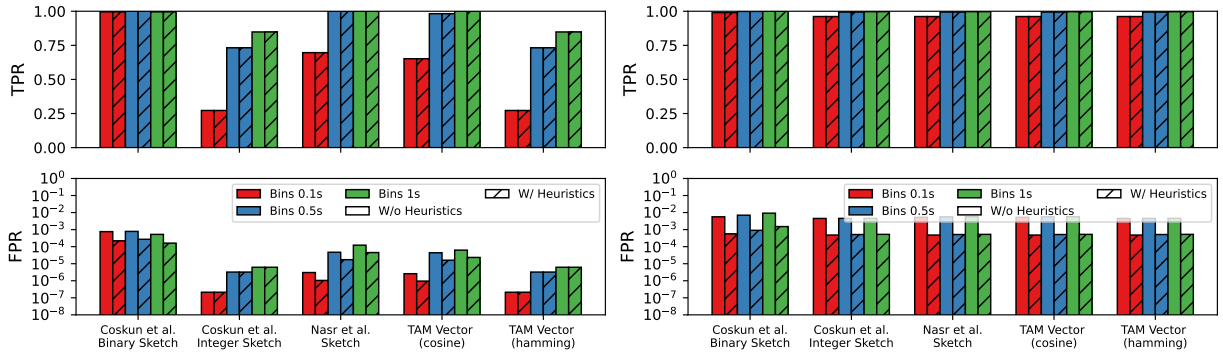
TAMs, making them highly attractive due to their smaller memory overhead. Focusing on the results obtained without the use of heuristics (solid bars), the integer sketching method from Coskun et al. [15] achieves equivalent accuracy (TPR: 0.9917, FPR: 3.98×10^{-4}) to TAM (in 0.1 seconds time bin and Hamming distance) while offering significant space savings—indeed, Figure 4.3 illustrates that TAM's memory footprint can be up to 60× larger than that of sketches under finer-grained binnings (e.g., $t = 0.1$). Figures 4.4a–4.4e show RevealNet's correlation effectiveness on additional datasets from Fu et al. [22], showing similar trends to those discussed on `bitcoinminer`. The TPR and FPR obtained across all datasets indicate that using the integer sketch achieves performance similar to TAM, while requiring 10% to 60% less memory (see Figure 4.3). This suggests that sketching can be effective for reducing memory usage. The results also show that the binary sketch achieves higher TPR than TAM in some cases, but with a higher FPR.

To assess whether the benefits of sketches are still applicable under perturbed network conditions, we performed additional experiments where we simulated the introduction of random packet drops at rates of 1%–5%. We now present our findings assuming that heuristics were deployed. As shown in Figure 4.2, both TPR and FPR declined under network perturbations (bars with markers), consistent with prior observations [15, 42]. For instance, the integer sketch (t =0.1s) maintained a TPR of 0.1188 and FPR of 4.7×10^{-5}, equivalent to that of TAM under the packet drop of 5%. Interestingly, the binary sketching variant yielded a significantly higher TPR (0.6634) compared to both integer sketches and TAM in the same conditions, albeit with a modest increase in FPR.



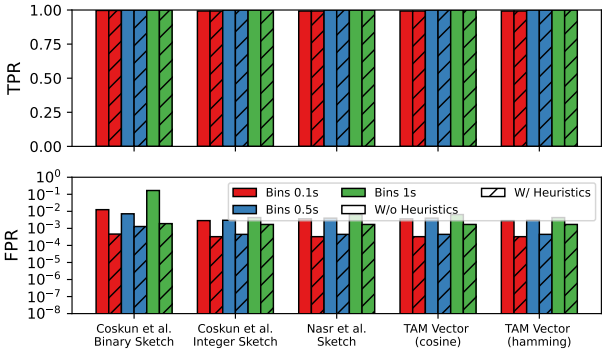
(a) adload

(b) dridex



(c) oracle

(d) penetho



(e) snojan

Figure 4.4: Correlation effectiveness of RevealNet across datasets.

Table 4.2: Number of feature vector comparisons required for each heuristic.

Dataset	None	Creation Time	Packet Count	Both
Dridex	128 626	19 409	2 403	824
Adload	126 019	17 386	1 168	258
Snojan	208 330	20 975	2 735	693
Oracle	294 334	41 580	95	84
Bitcoinminer	125 620	17 288	791	207
Penetho	294 814	28 040	1 735	454

Table 4.3: Total flows that can be stored in a P4 switch (for different methods).

Method/ Storage	Per flow (in Bytes)	Stored Flows (in 256 MB)
Nasr et al. (integer)	40	$\approx 6.4 \times 10^6$
Coskun et al. (integer)	40	$\approx 6.4 \times 10^6$
Coskun et al. (binary)	1.25	$\approx 204.8 \times 10^6$
TAM (0.1s bins)	2864	$\approx 1.05 \times 10^5$
TAM (0.5s bins)	576	$\approx 5.20 \times 10^5$
TAM (1s bins)	291	$\approx 1.03 \times 10^6$

Overall, the above results suggest that, even in noisy environments, sketches preserve the detection characteristics of more resource-intensive TAM configurations and can be relied upon for realizing RevealNet's correlation backbone. While not the main focus of our paper, we reckon that advanced traffic shaping techniques—beyond usual network perturbations—are challenging for accurate flow correlation, making this process significantly harder, if not infeasible [35, 44].

Heuristics reduce complexity and foster improved correlation Table 4.2 illustrates the impact of heuristics on reducing the flow comparisons performed by RevealNet. Without heuristics, comparisons range from 125k (`bitcoinminer`) to nearly 295k (`penetho` and `oracle`). With heuristics applied, this drops to 207, 454, and 84, respectively, thus reducing the computational effort involved in the correlation workload by at least three orders of magnitude. Beyond decreasing computational complexity, the heuristics also substantially lower false positives (see Figure 4.1 – bars with stripes). For instance, Coskun et al.'s binary sketch (based on packet counts tracked with $t = 0.1$) experiences a 96% decrease in FPR—from 0.008 to 0.0003—after heuristic filtering. This stems from eliminating benign or mismatched attacking flows that appear similar in sketch form but that differ significantly in creation time or total traffic volume.

Sketches allow for storing more flows concurrently Table 4.3 presents the approximated number of flows that can be stored in a Tofino v1 P4 switch equipped with ∼256 MB of SRAM [69], for the various feature extraction methods under analysis. All sketches are configured with a length of $m = 10$. For Coskun et al. and Nasr et al., the sketching process requires storing a projection matrix of size $n \times m$ (§3.4), which introduces a storage overhead of 24 400B, 4 920B, and 2 480B for TAMs based on 0.1s, 0.5s, and 1s bins (t), respectively. Although sketches require this fixed overhead, they dramatically improve storage capacity. For example, Coskun et al.'s binary sketch supports storing up to

Table 4.4: Communication overhead (in bits) for centralized and distributed correlation, evaluated per sketch (with heuristics) in the `bitcoinminer` dataset. The last column shows the overhead reduction under RevealNet's distributed setup.

Method	Centralized			Distributed (RevealNet)				OH Red. (%)
	$\mathbf{Sw_c \rightarrow CS}$	$\mathbf{Sw_a \rightarrow CS}$	**Total**	$\mathbf{S_a \rightarrow CM}$	$\mathbf{CM \rightarrow Sw_c}$	$\mathbf{Sw_c \rightarrow CM}$	**Total**	
Coskun et al. (binary)	1 193 390	2 020	1 195 410	2 020	38 380	736 896	777 296	35.0%
Coskun et al. (integer)	38 188 480	64 640	38 253 120	64 640	1 227 680	736 896	2 029 216	94.7%
Nasr et al. (integer)	38 188 480	64 640	38 253 120	64 640	1 227 680	736 896	2 029 216	94.7%
TAM	229 085 280	387 840	229 473 120	387 840	7 368 960	736 896	8 493 696	96.3%

204.8×10^6 flows, compared to just 1.05×10^5 with TAM at $t = 0.1$s granularity—the most memory-intensive setting in our study. Other sketching methods show similar scalability, reinforcing that sketch-based correlation is well-suited for memory-constrained P4 switches that must handle large flow volumes.

Distributed correlation saves bandwidth So far, our evaluation considers a single-switch setup. In practice, however, attack attribution spans multiple P4 switches located across different cooperating networks. In this distributed setting, correlation scales naturally in an "embarrassingly parallel" fashion: each switch handles its local traffic and performs correlation independently. We now gauge the communication overheads imposed by RevealNet, comparing them to traditional centralized attack attribution deployments.

Recall that RevealNet reverses the traditional data-sharing model of centralized systems, which require all probe nodes (Sw_c) to send full flow feature vectors to a central server (CS), resulting in high bandwidth overhead. Instead, RevealNet transmits only the feature vectors of *attacking flows*–collected at the attacked network's switch (Sw_a)–to a central correlation manager (CM), which then relays them to RevealNet-enabled switches (Sw_c) for localized correlation. To evaluate the communication overhead of centralized vs. distributed correlation, we simulate a topology with 20 RevealNet switches: 19 monitoring outgoing flows at cooperating networks (Sw_c), and one observing incoming flows at an attacked network (Sw_a). Assuming an even distribution of flows sourced from `bitcoinminer` (where we assume all flows to originate in cooperating networks and traverse the attacked network), each Sw_c sees 6 281 outgoing flows, while Sw_a sees a total of 119 339 incoming flows, out of which 202 are malicious. In a centralized setup, each Sw sends all observed flows' feature vectors to a CS, while the Sw_a sends its 202 feature vectors. In RevealNet, Sw_a sends the 202 feature vectors to the CM, which relays them to all Sw_c. Each Sw_c performs correlation locally and returns 202 matched flow tuples (192 bits each) to CM.

Table 4.4 shows a breakdown of the total communication involved in both scenarios. In our example setup with 20 switches/networks, RevealNet's distributed design reduces bandwidth usage by 35%–94.7%, depending on the used sketch.

Chapter 5

Conclusion

5.1 Concluding Remarks

In this thesis, we presented RevealNet, a practical and scalable framework for distributed attack attribution across multiple cooperating networks. RevealNet departs from traditional centralized attribution architectures by leveraging compact, sketch-based flow representations and the capabilities of programmable network devices to decentralize both feature extraction and correlation. By pushing correlation logic closer to the data plane, RevealNet reduces reliance on special-purpose middleboxes and minimizes the communication overheads typically incurred when exporting traffic features to external correlators.

A key contribution of this work is the adaptation of prominent traffic sketching techniques to the constraints of P4-programmable switches. Through careful design choices—such as integer-only arithmetic, online sketch updates, and fixed-size per-flow state—RevealNet demonstrates that sophisticated traffic correlation can be performed within the limited computational and memory budgets of modern switches. Our evaluation shows that RevealNet achieves correlation accuracy comparable to that of centralized approaches, while significantly improving scalability by enabling switches to track more flows and by reducing inter-network data exchanges. Overall, this work provides evidence that attack attribution can be effectively embedded within the network fabric itself. As programmable data planes continue to mature and proliferate in high-performance networks, approaches such as RevealNet open new opportunities for building scalable, low-latency security systems.

5.2 Limitations and Future Work

While the results presented in this thesis are promising, several limitations remain and motivate directions for future work. First, our prototype implementation of RevealNet was evaluated in a controlled environment using `bmv2`, the reference P4 software switch. Although `bmv2` provides functional fidelity to the P4 programming model, it does not fully capture the performance characteristics, resource constraints, and operational nuances of production-grade hardware targets such as Intel Tofino switches. Deploying and evaluating RevealNet on real programmable switch hardware remains an important next step to validate its practicality under real-world conditions.

A second important direction for future work concerns *privacy in collaborative attack attribution*. While RevealNet is designed to minimize data sharing by exchanging only compact flow-level representations rather than raw traffic traces, these summaries may still leak sensitive information about user behavior, traffic patterns, or network structure when shared across administrative domains [39]. Future work could explore the integration of privacy-enhancing techniques—such as noise injection, feature obfuscation, or cryptographic protocols for secure similarity computation [2, 17]—to further limit information disclosure while preserving correlation utility. Understanding the privacy–accuracy trade-offs introduced by such mechanisms is particularly relevant in large-scale, cross-domain deployments of collaborative security systems.

Beyond privacy, broader questions of *scalability and adoption incentives* also remain open. While RevealNet decentralizes correlation to avoid centralized bottlenecks, deploying such a system at Internet scale would require cooperation among heterogeneous networks with differing operational goals, trust relationships, and resource constraints. Future work could investigate incentive models and governance mechanisms that encourage participation by network operators, such as mutual benefit frameworks, regulatory support, or integration with existing inter-domain security initiatives [13, 16, 36].

Finally, RevealNet currently incorporates a limited set of flow sketching techniques and traffic features. There exists a broader design space of sketching methods, feature representations, and similarity metrics that could further improve correlation accuracy or efficiency. Exploring additional sketching schemes [31], incorporating richer traffic features [53], and dynamically adapting sketch parameters to network conditions represent promising directions for future work. Addressing these limitations will further strengthen the practicality of RevealNet and advance the deployment of decentralized attack attribution.

References

[1] AMD Corporation. Amd pensando 2nd generation ("elba"') data processing unit. https://www.amd.com/content/dam/amd/en/documents/pensando-technical-docs/product-briefs/pensando-elba-product-brief.pdf, 2024. Accessed: 2025-04-16.

[2] Onur Eren Arpaci, Raouf Boutaba, and Florian Kerschbaum. Over-threshold multiparty private set intersection for collaborative network intrusion detection. *arXiv preprint arXiv:2510.12045*, 2025.

[3] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *Proceedings of the Network and Distributed System Security Symposium*, 2021.

[4] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. The nuts and bolts of building flowlens. In *Proceedings of the Learning from Authoritative Security Experiment Results Workshop*, 2021.

[5] Debopam Bhattacherjee, Andrei Gurtov, and Tuomas Aura. Watch your step! detecting stepping stones in programmable networks. In *Proceedings of the IEEE International Conference on Communications*, pages 1–7, 2019.

[6] Santoshkumar Biradar, Parth Suresh Parsewar, Nilamadhab Mishra, Ankita Atul Galakatu, and Sarthak Santosh Gandewar. A survey on: Design, implementation, and evaluation of a secure and anonymous communication platform utilizing the tor network for enhanced privacy and data protection. In *Proceedings of the 4th IEEE International Conference on ICT in Business Industry & Government*, pages 1–6, 2024.

[7] Avrim Blum, Dawn Song, and Shobha Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings 7*, pages 258–277. Springer, 2004.

[8] Alessandro Brighente, Leonardo Cipolletta, Mauro Conti, and Ludovico Latini. Are flow correlation attacks effective on tor? a critical evaluation of state-of-the-art proposals. In *2025 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2025.

[9] Sambuddho Chakravarty, Marco V Barbera, Georgios Portokalidis, Michalis Polychronakis, and Angelos D Keromytis. On the effectiveness of traffic analysis against anonymity networks using flow records. In *Proceedings of the 15th International Conference on Passive and Active Network Measurement*, pages 247–257, 2014.

[10] Tisha Chawla, Shubh Mittal, Nate Mathews, and Matthew Wright. Espresso: Advanced end-to-end flow correlation attacks on tor. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, pages 219–220, 2024.

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PmLR, 2020.

[12] Xiang Chen, Chunming Wu, Xuan Liu, Qun Huang, Dong Zhang, Haifeng Zhou, Qiang Yang, and Muhammad Khurram Khan. Empowering network security with programmable switches: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 25(3):1653–1704, 2023.

[13] David Clark, Cecilia Testart, Matthew Luckie, and KC Claffy. A path forward: Improving internet routing security by enabling zones of trust. *Journal of Cybersecurity*, 10(1):tyae023, 2024.

[14] David D Clark and Susan Landau. Untangling attribution. *Harv. Nat'l Sec. J.*, 2:323, 2011.

[15] Baris Coskun and Nasir Memon. Online sketching of network flows for real-time stepping-stone detection. In *Proceedings of the Annual Computer Security Applications Conference*, pages 473–483, 2009.

[16] Xiaopeng Dai, Qianhong Wu, Bingyu Li, Bo Qin, Jialiang Fan, Fuyang Deng, and Mingzhe Zhai. R2e: A decentralized scheme for rewarding tor relays with cryptocurrencies. *IEEE Transactions on Dependable and Secure Computing*, 22(1):164–178, 2024.

[17] Inês Castro de Macedo. Privacy-preserving machine learning for network traffic analysis. Master's thesis, Universidade do Porto (Portugal), 2022.

[18] David L Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit, and Stuart Staniford. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*, pages 17–35, 2002.

[19] Roberto Doriguzzi-Corin, Luis Augusto Dias Knob, Luca Mendozzi, Domenico Siracusa, and Marco Savi. Introducing packet-level analysis in programmable data planes to advance network intrusion detection. *Computer Networks*, 239, 2024.

[20] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing china's blocking of unpublished tor bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, 2018.

[21] Ugo Fiore and Francesco Palmieri. A distance-based network activity correlation framework for defeating anonymization overlays. *Information Sciences*, 690:121559, 2025.

[22] Chuanpu Fu, Qi Li, and Ke Xu. Detecting Unknown Encrypted Malicious Traffic in Real Time via Flow Interaction Graph Analysis. In *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, USA, 2023.

[23] Ya Gao and Zhenling Wang. A Review of P4 Programmable Data Planes for Network Security. *Mobile Information Systems*, 2021.

[24] Joseph Hill, Mitchel Aloserij, and Paola Grosso. Tracking network flows with p4. In *Proceedings of the IEEE/ACM Innovating the Network for Data-Intensive Science*, pages 23–32, 2018.

[25] Nguyen Phong Hoang, Panagiotis Kintis, Manos Antonakakis, and Michalis Polychronakis. An empirical study of the i2p anonymity network and its censorship resistance. In *Proceedings of the ACM Internet Measurement Conference*, pages 379–392, 2018.

[26] Shiping Huang, Weina Niu, Ruoyu Wang, Tianlang Zhang, Kaipeng Li, Yanfeng Zhang, and Xiaosong Zhang. Flowcopcl: Enhanced flow correlation attacks on tor using patching and contrastive learning. *Tsinghua Science and Technology*, 31(2):745–759, 2026.

[27] Intel Corporation. Intel® tofino series: P4-programmable ethernet switch asics. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html, 2024. Accessed: 2025-04-16.

[28] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*, pages 337–348, 2013.

[29] Elie F. Kfoury, Jorge Crichigno, and Elias Bou-Harb. An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, 9:87094–87155, 2021.

[30] Ege Cem Kirci, Maria Apostolaki, Roland Meier, Ankit Singla, and Laurent Vanbever. Mass surveillance of VoIP calls in the data plane. In *Proceedings of the ACM Symposium on SDN Research*, pages 33–49, 2022.

[31] Shir Landau-Feibish, Zaoxing Liu, and Jennifer Rexford. Compact data structures for network telemetry. *ACM Computing Surveys*, 57(8):1–31, 2025.

[32] Ji Li, Chunxiang Gu, Xieli Zhang, Xi Chen, and Wenfen Liu. Attcorr: A novel deep learning model for flow correlation attacks on tor. In *Proceedings of the IEEE International Conference on Consumer Electronics and Computer Engineering*, pages 427–430, 2021.

[33] Zhen Ling, Junzhou Luo, Danni Xu, Ming Yang, and Xinwen Fu. Novel and practical sdn-based traceback technique for malicious traffic over anonymous networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1180–1188, 2019.

[34] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches. In *Proceedings of the 30th USENIX Security Symposium*, pages 3829–3846, 2021.

[35] Daniela Lopes, Jin-Dong Dong, Pedro Medeiros, Daniel Castro, Diogo Barradas, Bernardo Portela, Joao Vinagre, Bernardo Ferreira, Nicolas Christin, and Nuno Santos. Flow correlation attacks on tor onion service sessions with sliding subset sum. In *Proceedings of the Network and Distributed System Security Symposium*, 2024.

[36] Nguyen Cong Luong, Zeping Sui, Duc Van Le, Jie Cao, Bo Ma, Nguyen Duc Hai, Ruichen Zhang, Vu Van Quang, Dusit Niyato, and Shaohan Feng. Incentive mechanism design for resource management in satellite networks: A comprehensive survey. *IEEE Internet of Things Journal*, 13(3):3938–3964, 2025.

[37] Akshaya Mani, Tavish Vaidya, David Dworken, and Micah Sherr. An extensive evaluation of the internet's open proxies. In *Proceedings of the 34th Annual Computer Security Applications Conference*, page 252–265, 2018.

[38] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the tor network. In *Proceedings on Privacy Enhancing Technologies*, pages 63–76, 2008.

[39] Guozhu Meng, Yang Liu, Jie Zhang, Alexander Pokluda, and Raouf Boutaba. Collaborative security: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 48(1):1–42, 2015.

[40] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 183–195, 2005.

[41] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.

[42] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2053–2069, 2017.

[43] TJ OConnor, William Enck, W Michael Petullo, and Akash Verma. Pivotwall: Sdn-based information flow control. In *Proceedings of the Symposium on SDN Research*, pages 1–14, 2018.

[44] Se Eun Oh, Taiji Yang, Nate Mathews, James K Holland, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. Deepcoffea: Improved flow correlation attacks on tor via metric learning and amplification. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 1915–1932, 2022.

[45] Lennart Oldenburg, Marc Juarez, Enrique Argones Rúa, and Claudia Diaz. Mixmatch: Flow matching for mixnet traffic. *Proceedings on Privacy Enhancing Technologies*, 2024(2):276–294, 2024.

[46] L. Overlier and P. Syverson. Locating hidden servers. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[47] P4 Language Consortium. P416 language specification. https://p4.org/wp-content/uploads/sites/53/2024/10/P4-16-spec-v1.2.5.pdf, 2024. Specifies that P4 omits dynamic memory allocation and related constructs.

[48] P4ELTE Project. T4p4s: Retargetable p4 compiler and software switch. https://github.com/P4ELTE/t4p4s, 2024. Accessed: 2025-04-16.

[49] p4language Consortium. P4 Behavioral Model, https://github.com/p4lang/behavioral-model. Accessed: 2025-04-16.

[50] Francesco Palmieri. A distributed flow correlation attack to anonymizing overlay networks based on wavelet multi-resolution analysis. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2271–2284, 2019.

[51] Vera Rimmer, Theodor Schnitzler, Tom Van Goethem, Abel Rodríguez Romero, Wouter Joosen, and Katharina Kohls. Trace oddity: Methodologies for data-driven traffic analysis on tor. *Proceedings on Privacy Enhancing Technologies*, 2022(3):314–335, 2022.

[52] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. Subverting website fingerprinting defenses with robust traffic representation. In *Proceedings of the 32nd USENIX Security Symposium*, pages 607–624, 2023.

[53] Meng Shen, Jinhe Wu, Junyu Ai, Qi Li, Chenchen Ren, Ke Xu, and Liehuang Zhu. Swallow: A transfer-robust website fingerprinting attack via consistent feature learning. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 1574–1588, 2025.

[54] Gurjot Singh, Alim Dhanani, and Diogo Barradas. Revealnet: Distributed traffic correlation for attack attribution on programmable networks. In *2025 23rd International Symposium on Network Computing and Applications (NCA)*, pages 209–218, 2025.

[55] S. Staniford-Chen and L.T. Heberlein. Holding intruders accountable on the internet. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 39–49, 1995.

[56] Eric Wagner, Roman Matzutt, and Martin Henze. Seldom: An anonymity network with selective deanonymization. *arXiv preprint arXiv:2412.00990*, 2024.

[57] Xinyuan Wang, Douglas S Reeves, and S Felix Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *Proceedings of the 7th European Symposium on Research in Computer Security*, pages 244–263, 2002.

[58] Jiahao Wu, Heng Pan, Penglai Cui, Yiwen Huang, Jianer Zhou, Peng He, Yanbiao Li, Zhenyu Li, and Gaogang Xie. Patronum: In-network volumetric ddos detection and mitigation with programmable switches. In *Proceedings of the 29th European Symposium on Research in Computer Security*, pages 187–207, 2024.

[59] Bruno Missi Xavier, Rafael Silva Guimarães, Giovanni Comarela, and Magnos Martinello. Programmable switches for in-networking classification. In *Proceedings of the IEEE Conference on Computer Communications*, page 1–10, 2021.

[60] Guorui Xie, Qing Li, Yutao Dong, Guanglin Duan, Yong Jiang, and Jingpu Duan. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In *Proceedings of the IEEE Conference on Computer Communications*, pages 1938–1947, 2022.

[61] Jiarong Xing, Qiao Kang, and Ang Chen. Netwarden: Mitigating network covert channels while preserving performance. In *Proceedings of the 29th USENIX Security Symposium*, 2020.

[62] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, Ke Xu, Mingwei Xu, and Jianping Wu. Brain-on-Switch: Towards Advanced Intelligent Network Data Plane via NN-Driven Traffic Analysis at Line-Speed. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation*, pages 419–440, 2024.

[63] Kunikazu Yoda and Hiroaki Etoh. Finding a connection chain for tracing intruders. In *Proceedings of the 6th European Symposium on Research in Computer Security*, pages 191–205, 2000.

[64] Minlan Yu. Network telemetry: towards a top-down approach. *ACM SIGCOMM Computer Communication Review*, 49(1):11–17, 2019.

[65] Saif Zabarah, Omar Naman, Mohammad A Salahuddin, Raouf Boutaba, and Samer Al-Kiswany. Soteria: an approach for detecting multi-institution attacks. In *2023 26th Conference on innovation in clouds, internet and networks and workshops (ICIN)*, pages 113–120. IEEE, 2023.

[66] Menghao Zhang, Guanyu Li, Cheng Guo, Renyu Yang, Shicheng Wang, Han Bao, Xiao Li, Mingwei Xu, Tianyu Wo, and Chunming Hu. Superfe: A scalable and flexible feature extractor for ml-based traffic analysis applications. In *Proceedings of the Twentieth European Conference on Computer Systems*, page 818–834, 2025.

[67] Yin Zhang and Vern Paxson. Detecting stepping stones. In *Proceedings of the USENIX Security Symposium*, volume 171, page 184, 2000.

[68] Changgang Zheng, Xinpeng Hong, Damu Ding, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. In-network machine learning using programmable network devices: A survey. *IEEE Communications Surveys & Tutorials*, 26(2):1171–1200, 2023.

[69] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. An efficient design of intelligent network data plane. In *Proceedings of the 32nd USENIX Security Symposium*, pages 6203–6220, 2023.

[70] Huancheng Zhou and Guofei Gu. Cerberus: Enabling efficient and effective in-network monitoring on programmable switches. In *Proceedigns of the IEEE Symposium on Security and Privacy*, pages 4424–4439, 2024.

[71] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. Correlation-based traffic analysis attacks on anonymity networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(7):954–967, 2009.

[72] Yilin Zhu, Guang Cheng, Shunyu Zheng, and Hantao Mei. Toscorr: Transformer-based flow correlation attack on tor onion service. In *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 262–270. IEEE, 2024.