SketchFeature: High-Quality Per-Flow Feature Extractor Towards Security-Aware Data Plane

Sian Kim[†], Seyed Mohammad Mehdi Mirnajafizadeh[*], Bara Kim[‡], Rhongho Jang[*] and DaeHun Nyang[†]
[†]Ewha Womans University, [*]Wayne State University, [‡]Korea University

***Abstract*—Intelligent Network Data Plane (INDP) is emerging as a promising direction for in-network security due to the advancement of machine learning technologies and the importance of fast mitigation of attacks. However, the feature extraction function still poses various challenges due to multiple hardware constraints in the data plane, especially for the advanced per-flow 3rd-order features (e.g., inter-packet delay and packet size distributions) preferred by recent security applications. In this paper, we discover novel attack surfaces of state-of-the-art data plane feature extractors that had to accommodate the hardware constraints, allowing adversaries to evade the entire attack detection loop of in-network intrusion detection systems. To eliminate the attack surfaces fundamentally, we pursue an evolution of a probabilistic (sketch) approach to enable flawless 3rd-order feature extraction, highlighting High-resolution, All-flow, and Full-range (HAF) 3rd-order feature measurement capacity. To our best knowledge, the proposed scheme, namely SketchFeature, is the first sketch-based 3rd-order feature extractor fully deployable in the data plane. Through extensive analyses, we confirmed the robust performance of SketchFeature theoretically and experimentally. Furthermore, we ran various security use cases, namely covert channel, botnet, and DDoS detections, with SketchFeature as a feature extractor, and achieved near-optimal attack detection performance.**

I. Introduction

Artificial Intelligent (AI)-enhanced in-network defense [6], [50], [63], [49], [62] is emerging as a prominent trend for network security, supported by intensive research on General Network Intelligence (GNI) [36], [53], [4], [55], [13], [32], [64], [66], [37] and Intelligent Network Data Plane (INDP) [54], [63], [11], [30], [62], [52], [49]. Given the high-speed and large-scale nature of networks, thorough processing of raw data (i.e., packets) is believed to be unfeasible. Consequently, feature extraction in the data plane still remains challenging yet vital for AI-boosted in-network defense, by highlighting the essential needs for real-time processing capacities, constrained by hardware limitations such as pipelined architectures and limited computational resources.

To date, various features have been investigated to activate INDP for intrusion detection. Initially, 1st-order per-packet feature that is flow-agnostic and stateless, e.g., protocol, time to live, packet size, etc. [49], [62]. Research then expanded to include 2nd-order per-flow features that are flow-aware and stateful, e.g., flow size, mean, variance, minimum, maximum, etc. NetBeacon [63] demonstrated that 1st-order per-packet features encounter considerable limitations and must be combined with 2nd-order features to enhance accuracy in intrusion detection significantly. Furthermore, recent efforts have highlighted the effectiveness of 3rd-order features, specifically per-flow distribution information, in detecting application layer attacks, such as storage/timing covert channel, botnets, and website fingerprint [6], [50]. As such, In-Network Intrusion Detection Systems (IN-IDS) heavily rely on either 2nd-order or 3rd-order features, with the key difference stemming from the deployment locations of attack detection models. In alignment with recent INDP advancements, NetBeacon [63] focused on enabling a 2nd-order feature-based decision tree with the switch's data plane. On the other hand, 3rd-order feature-based systems, namely FlowLens [6] and NetWarden [50], deploy their detection logic in the switch's control plane (i.e., operation system). However, it is crucial to emphasize that irrespective of the location of attack detection logic, the feature extraction must occur within the switch data plane to enable line-rate investigation of all flows and traffic.

Although higher-order features provide superior attack detection capacities, in this paper, we reveal that precisely measuring flow characteristics for advanced 3rd-order features is complex and a simplistic approach can inadvertently provide new opportunities for adversaries. Specifically, measuring per-flow 3rd-order vector features significantly increases both time and space complexity compared to 2nd-order per-flow features. Given the hardware constraints of the switch data plane, current IN-IDS had to employ various workarounds for feature extraction to manage the full control loop, including per-flow feature extraction, attack flow detection, and access control list (ACL) deployment. However, our analysis indicates that feature extraction functions have become the weakest point of the detection systems due to their workaround designs, enabling adversaries to evade the entire attack detection loop.

For instance, FlowLens [6] and NetWarden [50] both targeted covert channel issues with per-flow distribution information. To simplify data plane operations, both systems employ a mean of quantizing a continuous distribution feature into a fewer number of discrete bins. The differences lie in FlowLens [6] conducts finer-grained quantization but limits its measurement to only the top-K bins, chosen from prior knowledge using priori-known attack data. Conversely, NetWarden [50] captures the entire distribution range but employs very coarse quantization, resulting in a very low-resolution perception of the flow features. These strategies result in a phenomenon where only selected flows are escalated to the switch's control plane for further detection, inherently precluding a comprehensive all-flow analysis in such flow escalation systems. We prefer to call their data plane mea-

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.241071
www.ndss-symposium.org

surement functions as *Symptom Detector* rather than feature extractor, since 3rd-order features are partially measured and used for attack flow identifications in such flow escalation systems. We highlight that both symptom detector designs assumed a static attacker, which allows an advanced adversary to evade the attack detection either by shifting the distribution patterns of attack flows or by overwhelming the system with crafted dummy flows (see section II-B for details). Besides, a similar flow selective processing behavior is observed in NetBeacon [63] that fully integrates ML models in the data plane. In particular, its feature extractor is active for predicted large flows only, which implies that only a subset of flows and packets will proceed to the detection model with moderate accuracy of flow size predictions from prior knowledge.

In this paper, we emphasize the critical need for **H**igh-resolution, **A**ll-flow, and **F**ull-range (**HAF**) 3rd-order feature extraction and introduce SketchFeature designed to fully support HAF within the programmable data plane. SketchFeature improves traffic visibility, thereby robustifying the control loop of IN-IDS by addressing vulnerabilities in feature extraction processes. Moreover, SketchFeature not only delivers exceptional 3rd-order feature accuracy but also facilitates 2nd-order feature quality through a post-hoc aggregation, thereby, has the potential to advance diverse INDP-based security applications [6], [50], [10]. SketchFeature[1] has been successfully deployed in a commercial switch with Tofino fabric [2]. Our contributions are as follows:

(1) Extending Sketch Capacity for 3rd-order Features. We tackle the 3rd-order feature measurement challenge using a sketch data structure with memory random sharing, traditionally limited to per-flow 2nd-order features. To extend this capability, we introduce a novel concept of *sketch virtualization* that allows the traditional sketch data structure to preserve all-flow measurement capacity in more complex tasks.

(2) Addressing Technical Challenge for Sketch Evolution. We identify the technical challenges in advancing traditional sketch capacity for higher-order feature measurement, which is completely neglected by state-of-the-art schemes when pursuing a similar goal. We call it phantom decoding, which refers to negative queries of non-existent features when a flow is encoded from multiple feature dimensions 3rd-order feature. In this paper, we make an initial effort to model the issue formally and theoretically and propose a feasible use of membership testing, enhancing noise reduction with provable error bounds with probabilistic guarantees.

(3) Theoretical and Experimental Validation. We show that the proposed SketchFeature is feasible for HAF 3rd-order feature measurement with exceptional memory efficiency and effective suppression of phantom decoding noise, without compromising inherent sketch accuracy. We provide detailed experimental results evaluating SketchFeature's ability to extract 3rd-order features such as inter-packet delay (IPD) and packet size (PS) distributions, along with 2nd-order features, such as minimum, maximum, mean, standard deviation, and entropy per flow, to show the feasibility of SketchFeature.

(4) Advancing Security Applications. We evaluate our system's security effectiveness through various use cases, namely covert channel, botnet, and DDoS attack detection varying datasets [31], [42], [8], [43]. The results demonstrate the high-quality and robust feature extraction capability of SketchFeature and near-optimal attack detection performance, where the optimal is based on exact feature measurement.

[1]The source code is at https://github.com/ISRL-EWHA/SketchFeature

Organization. The rest of the paper is organized as follows. In section II, we motivate our works by proposing two practical attacks on the existing feature extractors. In section III, we explain the SketchFeature's idea and primitive design, followed by a theoretical analysis in section IV. In section V, we investigate the feature extraction quality of SketchFeature over various settings and security applications, with extended discussions in section VI. Finally, we discuss related works and conclude the paper, in sections VII and VIII, respectively.

II. Background and Motivation

A. Background

3rd-order Features for Security. Due to the ability of finer-grained measurement involving multi-dimensional information per flow, 3rd-order features have shown to be crucial in advanced security applications [50], [6], [19], [20], [63], such as per-flow inter-packet delay (IPD) distribution and packet size (PS) distribution. Furthermore, the richer and finer information contained in 3rd-order features can be aggregated to derive lower-order features, such as min, max, mean, standard deviation, and magnitude, which are also preferred in various attack detection scenarios [38], [63].

Data Plane Constraints. Early detection of attacks within the network data plane is crucial for mitigating link flooding [65], [51] as it allows immediate actions to minimize attack impacts. Various efforts have focused on enhancing data-plane detection capabilities [6], [50], [63], [49], [62], [34], [59], [57], [14], [29], [17], [28]. However, with line-rate packet processing as the top mission, the first constraint of ASIC-based programmable switches is limited support for arithmetic logic. While addition, subtraction, and bit operations are available, relatively complex operations are absent, such as multiplication, division, and floating point operations. Second, due to physical constraints, ASIC maintains only 12 stages (hardware partitions) on its pipeline, each embeds scarce stateful SRAM and TCAM memory and arithmetic logic unit (ALU) resources. Notably, a table (array) data structure defined in the stateful memory can be accessed only once per packet and for one-entry read-write operation. With logical and physical constraints, ASIC shows a natural limit of recording rich flow information for advanced and robust attack detections.

B. Motivation

System Model (Data Plane Workarounds). To accommodate the data plane needs, several workaround solutions have been proposed, namely *selective flow feature extraction with symptom detection*. As illustrated in Fig. 1, the existing In-network IDS (IN-IDS) falls into two categories: in-network detection and intelligent data plane. ❶ *In-network Detection.* FlowLens [6] and NetWarden [50] fall into the first category to perform per-flow detection in the programmable switch's control plane. However, unlike the data plane, a programmable switch's control plane is not capable of processing fine-grained features for all flows because of the hardware nature of using

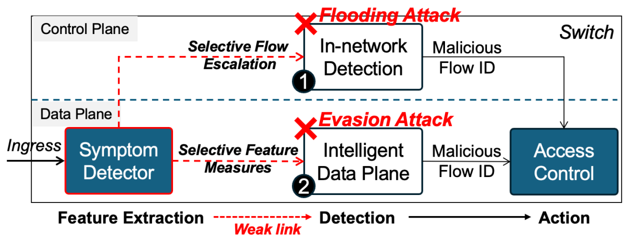


Fig. 1: System and threat models: through targeting the symptom detector, the attacker can bypass the flow selection process, thereby evading the attack detection mechanism in both the data and control plane.

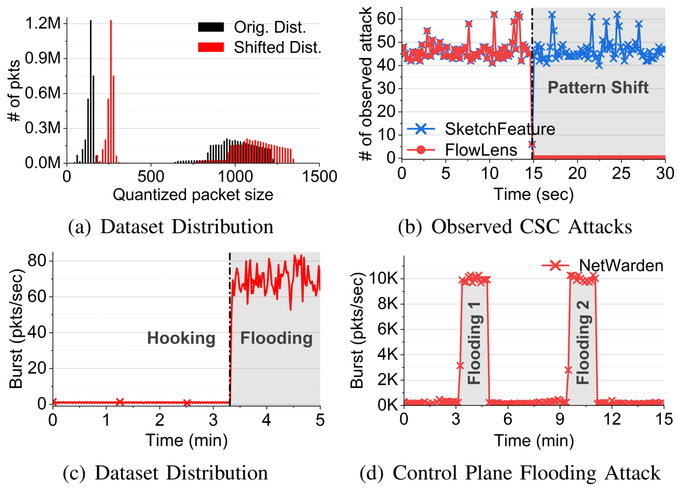


Fig. 2: Attack evasion demonstration for two state-of-the-art 3rd-order feature collections: as shown, both selective and low-resolution symptom detectors are vulnerable to attack evasion in the data plane and flooding of the control plane.

a general CPU with DRAM. Therefore, a per-flow "feature extractor" in the data plane *escalates only suspicious flows* to the control plane for detection with either a predefined threshold [50] or learning-based model [6]. ❷ *Intelligent Data Plane.* NetBeacon [63] is in line with a recent trend to deploy a machine learning model in the data plane [54]. A per-flow feature extractor is placed in front of the model, to pre-process a 2nd-order (stateful) feature for learning-based attack detection. Nevertheless, due to the huge amount of stateful memory required for all-flow measurement, NetBeacon utilizes a stateless (per-packet) 1st-order feature for a first-aid action, namely the prediction of large flow packets. By doing so, NetBeacon's feature extractor and detection model *serve large flows only* leveraging a flow size prediction model.

Threat Model. Our threat model sheds light on the detective feature extractors that supply flows *selectively* to the detection phase of the control loop, as shown in Fig. 1. We assume an adversary that targets the naive *flow selection logic* with strong prior knowledge (e.g., historical traffic patterns [58], [50] and flow size [63]). We call it "symptom detector" since the flow selection is performed based on either low-resolution or selective symptoms of a flow. The attacker's goal is to evade the entire control loop by attacking the vulnerable link.

Attack Surface 1. Selective Symptom Detection. FlowLens [6]'s data plane is a hash table that originally aimed at per-flow packet size distribution (3rd-order). However, covering the full spectrum of the packet size distribution (e.g., 1500 bins per table entry for packet size range 0∼1500 Bytes) is not feasible at the data plane. To workaround, FlowLens takes a two-step design to relax the complexity. Its "feature extractor" first quantizes the packet size range in a relatively fine-grained resolution. Despite this effort, covering the full spectrum of the quantized feature is a big burden for memory. For example, a quantization interval 16 still requires each hash table entry to record 94 bins with 32-bit counters per bin, equivalent to 376 Bytes per flow. Our analysis with a 5-second CAIDA trace indicates that 6 MB memory space allows only 6 bins quantization (low resolution) for per-low packet size distribution measurement, and with a finer-grained 94-bin setting, FlowLens can handle 12.6% of flows only. Therefore, FlowLens's second design is to selectively measure flows that fall into historical top-K bins and detect attacks with the top-K symptoms only.

Data-plane Evasion Attack. Fig. 2 (a) shows the attacker's logic. The black bars depict the packet size distribution (PSD) of the known covert storage channel (CSC) traffic Facet [31]. With top-10 bins, FlowLens claimed to identify covert channels accurately. However, it has been studied that network traffic patterns vary even for the same flow [29]. Also, the packet size is under an attacker's control. In our experiment, we simply added a small amount of data to each packet of the original attack trace, which is equivalent to only 7.4% bandwidth overhead; however, it triggered significant enough distribution shifts, as shown in Fig. 2 (a) (red bars). As a result, the selective symptom detector could not observe any adversarial flows with the pattern shifting, resulting in the evasion of the entire control loop even before detection, as shown in Fig. 2 (b). It is worth mentioning that the state-of-the-art data plane detection system relies on the essential packet header fields (i.e., top-K 1st-order features) to predict the eventual flow size and perform detection for large flows only [63]. Therefore, the primitive adversarial behaviors that tamper per-packet information (e.g., packet size) remain effective for misleading prior knowledge-based decision-making.

Intuition. One may consider that attacking with a specific packet length can be a strong indicator of pattern shifts. However, it is still invisible to FlowLens due to selective feature measurement based on prior knowledge of Top-K. Therefore, it is desired that a feature extractor can measure a full spectrum (full-range) of fine-grained (high-resolution) distribution features for entire traffic (all-flows), to eliminate blind spots for IN-IDS.

Attack Surface 2. Low-resolution Symptom Detection. NetWarden [50] leverages per-flow inter-packet delay (IPD) distribution to mitigate covert timing channel (CTC) [50], [21], [35], [12], [42], where attackers aim to hide secret information varying IPDs. Different from FlowLens [6] that monitors selective symptoms, NetWarden's control plane detector is aimed at full-range IPD distribution. However, since it is unrealistic to send all packets to the programmable control plane, a coarse IPD distribution (very large quantization interval) is measured at the data plane to escalate only suspicious flows to the control plane with a pre-defined threshold (i.e., symptom detector).

Control-plane Flooding Attack. For CTC detection, maintaining a *low* false positive rate (benign as an attack) is as critical as *low* false negative rate (attack as a benign) due to a high level of unobservability [7]. We stress that instead of attacking the detection algorithm that resides in the control plane, an adversary can fool the coarse symptom detector by injecting a large number of dummy flows. Consequentially, the coarse measurement with a static threshold will trigger the redirection of massive dummy flow packets toward the control plane leading to resource exhaustion. As shown in Fig. 2 (c), we craft dummy attack flows, which are slow (i.e., $>$ 1-second IPD) at the beginning and hooking stage, then burst after 200 packets were sent (i.e., symptom threshold). As shown in Fig. 2 (d), the dummy flooding traffic can increase the packet communicating to the control plane one thousand times. Since the amount of flows is an attacker's option, NetWarden's control plane can be flooded easily.

Intuitions. NetWarden [50] may use a rate limiter or detector to counter a control plane flooding attack. However, a rate limiter at the control plane cannot prevent flooding of the data-control-plane channel. Moreover, a rate limiter applied in the data plane requires the support of scarce matching-action resources and an additional mechanism for flooding detection, making attack surface mitigation complex. Therefore, it is desired that a feature extractor can process features in the data plane only, without redirecting packets to the control plane, to mitigate the control-plane threats fundamentally.

C. Design Goals

Through the security analyses, we can observe that the various workarounds in the data plane made adversarial attacks easier for evading the entire control loop of IN-IDS. Particularly, the selective and low-resolution symptom detectors based on prior knowledge are defective when adversaries slightly tamper the traffic patterns. These insights motivate us to design a better feature extraction function, namely SketchFeature, for a robust 3rd-order feature measurement. The design goals of the proposed scheme lie in 1) leveraging a probabilistic (sketch) approach to enable 3rd-order feature measurement, 2) identifying fundamental challenges for the sketch evolution pursuing higher-dimensional and higher-order feature encoding, 3) exploring viable solutions to tackle the challenges with theoretical supports, and 4) realizing high-resolution, all-flow, full-range (HAF) feature extraction without workarounds to eliminate attack surfaces fundamentally.

III. SketchFeature Design

We begin with a formal definition of the problem. Then, we describe the encoding and decoding algorithms of SketchFeature, followed by design logistics.

A. Formal Problem Definition

Let each flow f_i within a dataset $\mathcal{F} = \{f_1, f_2, \dots, f_p\}$, where p denotes the number of distinct flows. With q representing the number of quantization levels, $a_{f_{ij}}$ indicates the number of packets of flow f_i in the j-th quantized bin (QL_j). Then, each flow f_i is characterized by an distribution vector $a_{f_i} = (a_{f_{i1}}, a_{f_{i2}}, \dots, a_{f_{iq}})$, as shown in Fig 3. A functional

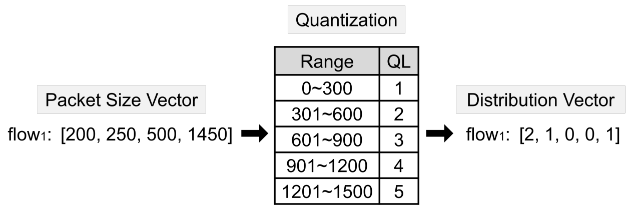


Fig. 3: Complexity reduction of per-flow distribution (3rd-order) feature through quantization.

sketch, SketchFeature, denoted as $\mathcal{SF}$, is used to provide estimated values for specific queries $Q(\mathcal{SF}, a_{f_{ij}})$, where

$$\hat{a_{f_{ij}}} \leftarrow Q(\mathcal{SF}, a_{f_{ij}}),$$

which estimates the packet count in QL_j of flow f_i. This produces the estimated distribution vector $\hat{a_{f_i}} = (\hat{a_{f_{i1}}}, \hat{a_{f_{i2}}}, \dots, \hat{a_{f_{iq}}})$ for flow f_i when aggregated across all quantized bins. In the case of *phantom decoding*, when querying a flow that has never been encoded in QL_j, SketchFeature has a high probability of providing the following response,

$$0 \leftarrow Q(\mathcal{SF}, a_{f_{ij}}).$$

B. Building Blocks of SketchFeature

Quantizing 3rd-order Feature. Quantization is a powerful tool for per-flow distribution (3rd-order) feature measurement [50], [6]; converting continuous feature values into discrete values for complexity reduction. Arithmetic quantization, which divides a continuous range of values uniformly into multiple intervals (or bins), is frequently used when dealing with features like inter-packet delay (IPD) and packet size (PS) distributions. As shown in Fig. 3, with a quantization level (QL, hereafter) of 300, the packet size within the range [0,300] can be grouped into the first bin, values of [301, 600] become the second bin, and so on. The interval setting (i.e., the number of bins) is the main trigger of a trade-off between granularity (resolution) and complexity (distinct discrete values). As state-of-the-art works [50], [6], SketchFeature also leverages quantization for complexity reduction, but pursue *higher resolution* of *full range* distribution features for all-flow (i.e., HAF feature measurement) to accelerate in-network intrusion detection.

Data Structure and Workflow. Fig. 4 illustrates the data structure and the simplified workflow of SketchFeature. It includes three primitive components including a quantization function (QNT), virtual sketch, and membership test function (Bloom Filter). SketchFeature's sketch data structure has d independent array ($d = 3$ in the illustrated example), each consisting of w 32-bit counters. For encoding, each feature (packet) value of a flow is firstly quantized into the corresponding bin, and then encoded into both sketch and membership test functions, where the former tracks the frequency of the identical feature and the latter records the appearance of the feature. For decoding, all flows' distribution vectors can be retrieved from the sketch, by querying the appearance of all bins from the membership test function. The unique design of SketchFeature is the virtual sketch that allows feature-aware encoding, which is the key to enabling complex feature extraction even in the highly resource-constrained data plane of programmable switches.

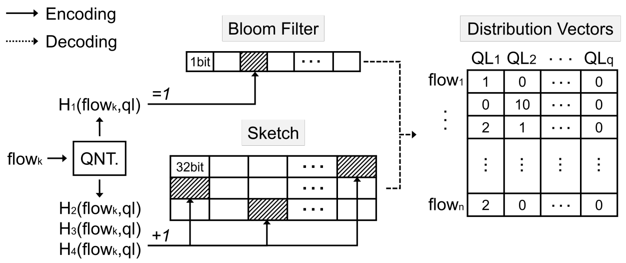


Fig. 4: SketchFeature: data structure and workflow.

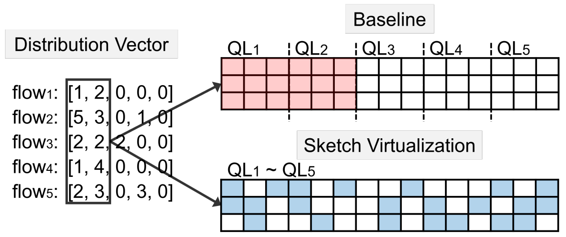


Fig. 5: Sketch partition (baseline) vs. sketch virtualization.

Algorithm 1: Encoding and Decoding

```
Input: flowID $f$, feature value $v$, Bloom filer $B$, Sketch $S$, number
       of layers of Bloom filter $d_b$ and sketch $d_s$, width of layers
       in Bloom filter $w_b$ and sketch $w_s$
Function Encoding($f, v$):
    $l \leftarrow$ get_quantization_level($v$)
    for $i \leftarrow 1$ to $d_b$ do
        $idx \leftarrow$ hash_i($f$, $l$) % $w_b$
        $B_i[idx] \leftarrow 1$
    end
    for $i \leftarrow 1$ to $d_s$ do
        // Sketch virtualization.
        $idx \leftarrow$ hash'_i($f$, $l$) % $w_s$
        $S_i[idx] \leftarrow S_i[idx] + 1$
    end
end
Function Decoding($f, l$):
    for $i \leftarrow 1$ to $d_b$ do
        // Tackling phantom decoding issue.
        $idx \leftarrow$ hash_i($f$, $l$) % $w_b$
        if $B_i[idx]$ is 0 then
            return 0
        end
    end
    $a \leftarrow$ INT_MAX
    for $i \leftarrow 1$ to $d_s$ do
        $idx \leftarrow$ hash'_i($f$, $l$) % $w_s$
        $a \leftarrow$ min($a$, $S_i[idx]$)
    end
    return $a$
end
```

Encoding and Decoding. Algorithm 1 describes the encoding and decoding operation of SketchFeature. For simplicity, the distinct hash functions for each quantization level sketch are implemented by using the quantization level (l) as an input to the hash function. During encoding, when a flow with flow ID f arrives, the feature value v (e.g., packet size, IPD, etc.) is quantized to determine the corresponding quantization level. Then, it sets the bit in the Bloom filter to 1, indicating that flow f has been encoded at l-th quantized bin sketch. Lastly, the counters in all layers of the sketch are incremented to update that the value falls into the l-th quantized bin sketch has arrived for flow f. As shown, sketch virtualization is realized through distinct hash functions, using a single sketch without partitioning it. Decoding is the process of estimating the packet count of flow f that belongs to the quantization level l. Similar to encoding, the Bloom filter is checked using the hash function with l as an input. If any bit across all layers is 0, the value in the sketch is phantom, so 0 is returned. Otherwise, since SketchFeature is based on the Count-Min sketch, it returns the minimal counter value across all layers.

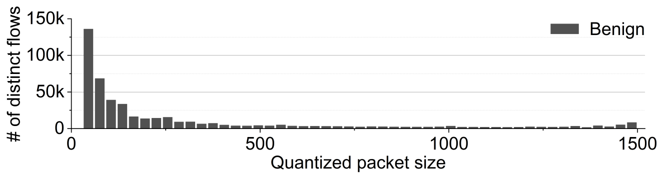
C. Design Logistics of SketchFeature

In the following, we first provide a step-by-step explanation of the virtual sketch function of SketchFeature. Then, we unveil a key technical challenge during the evolution of the Count-Min sketch from 2nd-order (scalar) to 3rd-order (vector) feature extractions. Lastly, we discuss our solution, namely the membership test, that can be fully deployed in the data plane. The ultimate goal of SketchFeatureis **High**-resolution, **A**ll-flow, and **F**ull-range (**HAF**) 3rd-order feature extraction.

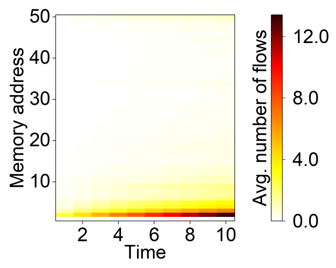
Sketch Partitioning (Baseline). For a better understanding, we start with a strawman (baseline) approach that utilizes a conventional sketch for distribution feature encoding. We note that the baseline sketch is not introduced by this work but is inspired by and extended and generalized from NetWarden's [50] data plane approach, where the original version detects suspicious candidates with a coarse binary partition of a distribution feature (i.e., 2 bins only) for simple outlier detection. The baseline was the only sketch that measures 3rd-order (distribution) vector features in the data plane, and possibly for fine-grained HAF feature extraction.

As illustrated in Fig. 5, the baseline approach quantizes packet size range (i.e., 0∼1500 bytes) into 5 bins (i.e., $QL_1 \sim QL_5$), and then uses five memory-independent Count-Min sketches [15] to perform per-flow counting at each sketch distinguishing bins (QLs). The baseline approach is generally valid but suffers from a memory inefficiency issue stemming from the hard partition of sketches. As shown in Fig. 6, the imbalance of network traffic leads to a biased load to the sketches and causes significant memory inefficiency. When the distribution vector is skewed, all flows will be encoded into the sketches designated for bins QL_1 and QL_2, whereas sketches assigned for bins QL_3, QL_4, and QL_5 remain unused and wasted. Additionally, as more flows are encoded in bins QL_1 and QL_2, they experience greater overestimation due to hash collisions (i.e., sketch error). We note that with a fine-grained quantization, more data will be encoded into smaller sketches, saturating the sketch memory (i.e., error-bound guarantee) and leading to higher noise of feature values in the crowd range. Eventually, a "single-point failure" will change the whole shape of the distribution feature when decoding.

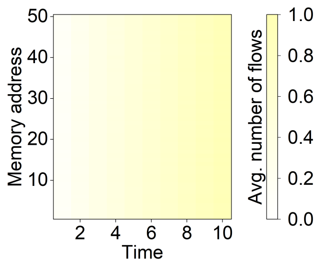
Sketch Virtualization (Our Approach). To improve memory efficiency, we introduce a strategy called *sketch virtualization*. This technique allows distinct sketches (for different bins) to virtually share a single memory space without partitioning it. To realize sketch virtualization, SketchFeature employs different hash functions for each bin (QL_i), enabling the feature value to be encoded into the virtual sketch corresponding to its range. As shown in Fig. 5, SketchFeature uses the



(a) Packet size distribution of CAIDA Trace (2.59 million packets) [1]



(b) Baseline sketch



(c) Sketch virtualization

Fig. 6: Memory efficiency comparison: sketch partitioning (baseline) vs. sketch virtualization (SketchFeature). 6 MB memory space was given to each scheme for per-flow packet size distribution measurement using a 5-second CAIDA trace.

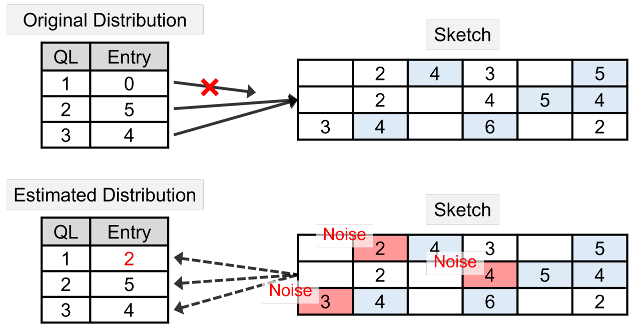


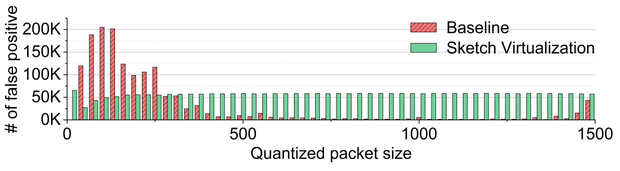
Fig. 7: Concept of phantom decoding.

entire memory space for all bins. This is possible because of a characteristic of a hash function, which is designed to distribute its output as uniformly as possible, thereby enhancing memory utilization and reducing sketch errors caused by hash collisions. Notably, in resource-constrained environments such as ASIC-based hardware, creating numerous partitioned sketches is infeasible, thus the baseline sketch cannot be deployed in the data plane in an HAF manner. However, virtual sketches differentiated by hash functions can be implemented without additional overhead. To confirm, Fig. 6 (b) and (c) show the average number of flows encoded in each register (counter) when the feature is quantized into 50 bins (i.e., the number of QLs). Unlike SketchFeature, the baseline sketch uses partitioned individual sketches; thereby each memory address corresponds to a unique bin. Referring to the dataset in Fig. 6 (a), the baseline sketch shows collisions concentrated in specific quantized bin sketches according to the dataset's distribution. This issue becomes more pronounced over time, and bins that rarely encode flows result in significant memory waste. Conversely, SketchFeature virtually divides a single sketch and utilizes the whole memory space, promoting an even distribution of the dataset across the entire memory space.

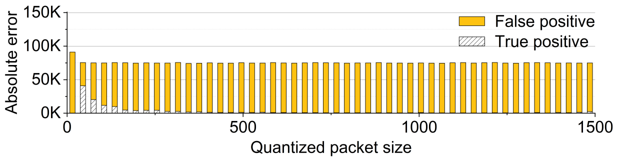
Understanding Phantom Decoding Issue. Although multiple efforts were made to extend the sketch's capacity from 2nd-order to 3rd-order feature measurement, the most critical technical challenge for sketch theory is not fully discussed and often neglected by existing works [25], [50]. In this work, we make initial efforts to systematically study the sketch challenge for the 3rd-order feature measurement, namely *Phantom decoding*. Worth mentioning that the phantom feature decoding issue does not exist when using the sketch in a conventional way for 2nd-order feature extraction. However, when working with a 3rd-order feature, a single flow feature has to be distinguished within the flow by different bins for a distribution representation. Here, the conventional sketch blindly decodes the corresponding feature value, because it is impossible to determine which bin the flow was encoded with at the moment of decoding, which we refer to the phantom decoding issue. As shown in Fig. 7, the first bin (QL_1) in the original distribution vector of the flow has no value, and therefore, it is not encoded into the sketch. However, this absence of encoding is unknown during decoding, leading to a response with a random value to the query for this non-existent value. This phantom decoding issue results in amplified decoding values that are merely noise to the distribution vector. We note that traditional sketches are primarily designed to respond to positive queries of encoded flows. However, in the context of decoding 3rd-order features, simply confirming the presence of flows is insufficient to prevent negative queries. It also becomes essential to verify the appearance of specific bins for suppressing phantom decoding. For example, HistSketch [25] recognized the issue, however, detoured the issue on a data center with an unlimited resource to record all flows' bin information to solve the issue, neither in the data plane nor within the sketch.

Membership Test for Suppressing Phantom Decoding. We note that network flows are mostly found to be sparse in bin space (i.e., feature distribution), which in turn, inevitably brings the phantom feature decoding effect. Furthermore, given a fixed memory, increasing the number of features (i.e., the number of quantized bin sketches) to achieve higher resolution can exacerbate the issue. To overcome the phantom decoding challenge and achieve higher quality of the estimated distribution, SketchFeature uses Bloom filter [9] as a membership indicator to determine which features have been encoded per flow. Similar to sketch virtualization, the Bloom filter of SketchFeature is virtually divided for different bins using distinct hash functions. When encoding a flow, the bit in the Bloom filter corresponding to the bin is set to 1. Therefore, unlike HistSketch [25] relying on data center resources to record encoded bins, SketchFeature can address the phantom decoding issue in the data plane. Due to the nature of the Bloom filter, false negatives cannot occur, but false positives are possible. However, the probability of false positives can be theoretically analyzed and controlled (See section IV).

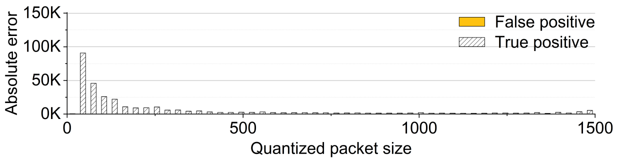
Fig. 8 (a) shows the negative impact of phantom decoding in the baseline sketch and sketch virtualization. As shown, the probability of phantom decoding is proportional to the saturation level of the sketch, as the chance of falsely decoding a value increases when there are fewer empty counters. Consequently, it can be observed that the baseline sketch suffers from a massive number of phantom decoding at the heavily saturated



(a) False positive distribution (phantom decoding issue)



(b) Sketch virtualization without membership test

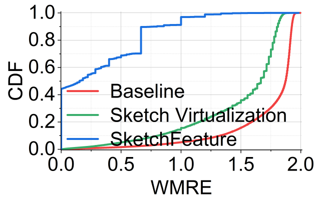


(c) Sketch virtualization with membership test (SketchFeature)

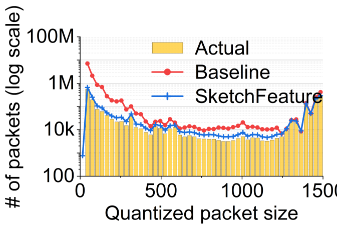
Fig. 8: Step-by-step analyses: (a) compares the phantom decoding error (false positive) between sketch partitioning (baseline) and sketch virtualization (SketchFeature). (b) shows both true and false positives with sketch virtualization design only, and (c) shows that phantom decoding errors can be resolved by combining our membership test design.

bins. On the other hand, the memory utilization of the sketch virtualization approach is almost uniform across all bins, resulting in a nearly constant rate of phantom decoding, even for the rarely used bins, hurting feature measurement accuracy significantly. Fig. 8 (b) shows the cumulative error for each quantized bin when decoding the sketch virtualization method without a membership test. As shown, the error caused by phantom decoding (false positive) is much greater than errors caused by actually encoded values (true positive). Our solution to the problem is to use the Bloom filter for the bin record tracking, and the operations have been shown in Algorithm 1. Here, we allocated 2 MB (out of 6 MB total memory) to the Bloom filter. As shown in Fig. 8 (c), although the error due to true positives increased slightly due to the reduced sketch size, the false positive errors were reduced to a negligible level. Consequently, the overall error decreased, confirming that the membership test effectively enhances accuracy.

All Put Together. To verify SketchFeature's HAF feature quality, we measured how closely the estimated packet size distribution matches the original distribution on a per-flow basis. Fig. 9 (a) shows Weighted Mean Relative Error (WMRE), a metric for measuring the similarity between the two distributions, where a value closer to 0 indicates less error. As shown, sketch virtualization demonstrates better accuracy than the sketch partition design (baseline) due to its memory efficiency. Later, by resolving the phantom decoding issue with our membership test function, SketchFeature measures the per-flow distribution that is closest to the original among the methods. Fig. 9 (b) compares an estimated packet size distribution of SketchFeature with ground truth in log-scale. As shown, in bin QL_1, where no packets are distributed, the baseline sketch



(a) Per-flow distribution quality



(b) Traffic distribution quality

Fig. 9: Accuracy comparison of sketch partitioning (baseline), sketch virtualization, and SketchFeature (sketch virtualization + membership test). 6 MB of memory space was given to measure the 3rd-order distribution feature quantized in 50 bins.

does not suffer from the phantom decoding issue. Thus, it can perfectly decode 0 from the sketch. However, in bin QL_2, where packets are mostly distributed, NetWarden's estimated packet count is 12.58 times higher than the ground truth. On the other hand, SketchFeature's estimated number of packets ranges from 1.02 to 1.58 times higher than the ground truth, which is much lower than the overall average estimated error of the baseline (i.e., 3.68 times higher than the ground truth), which infers SketchFeature can perform the measurement that is closer to the original distribution. Moreover, we can observe that the accuracy of the baseline sketch is heavily influenced by the feature distribution, whereas SketchFeature effectively transforms the distribution to be almost uniform benefiting from sketch virtualization, thereby reducing overall error significantly. Therefore, SketchFeature achieves HAF feature measurement by providing 1) higher resolution for distribution features benefited from sketch virtualization delivered memory efficiency, 2) all flow encoding capacity with sketch scalability, and 3) full-range feature measure with negligible error, due to the effectiveness of membership test for resolving the phantom decoding issue.

IV. Theoretical Analysis

In this section, we analyze the phantom decoding probability and error bounds of the baseline sketch (*Baseline*) and SketchFeature. To do so, we first summarize the necessary notation in Table I. Then, we compare the performance of the baseline sketch and SketchFeature through theoretical proofs and evaluate the results obtained by substituting real datasets.

A. Probability of Phantom Decoding

Both *baseline* and *sketch virtualization* sort inputs into q quantized bins, which inevitably incurs phantom decoding to result in poor measurement results. In this section, the phantom decoding probabilities for both schemes are presented.

Lemma 1 (Phantom decoding probability of *sketch virtualization*)**.** *Given sketch virtualization consisting of a Count-Min sketch with w counters in d layers, the phantom decoding probability when inserting n distinct elements (or flow in our context) is as follows:*

$$P_{phantom} = \left(1 - e^{-n/w}\right)^{d}.$$

Proof. Considering the decoding process of Count-Min sketch [15], the scenario where a non-existent flow decodes a phantom feature value occurs when all of d counters for the flow in every layer are non-empty, which is *phantom decoding* that we have defined. Thus, after encoding all elements, the probability that all of the corresponding d counters are non-empty for a non-existent flow represents the phantom decoding probability. This can be calculated similarly to the false positive probability in Bloom filter [9].

Encoding n distinct elements into *sketch virtualization*, which comprises d layers with w counters each, the probability that a specific counter in any layer is not empty can be expressed as follows:

$$P_{\text{non-empty}} = 1 - \left(1 - \frac{1}{w}\right)^n,$$

When a non-existent element is queried, the probability of encountering a phantom decoding, where all d layers are not empty, can be calculated as:

$$P_{\text{phantom}} = \left(1 - \left(1 - \frac{1}{w}\right)^n\right)^d,$$

Assuming that w is sufficiently large, the above expression can be approximated as follows:

$$P_{\text{phantom}} = \lim_{w\to\infty}\left(1 - \left(1 - \frac{1}{w}\right)^n\right)^d = \left(1 - e^{-n/w}\right)^d. \quad \square$$

The baseline sketch strictly separates memory space into q independent memory spaces running each sketch for q quantized bins, and no crossover of a packet is allowed across bins. Therefore, we can regard it as a system having q parallel and isolated sketches.

Lemma 2 (Phantom decoding probability of *baseline*)**.** *If the l-th quantized bin sketch of the baseline sketch having q bins consists of d layers, each with w/q counter, then the phantom decoding probability for partitioned individual sketches is as follows:*

$$P'_{\textit{phantom}} = \left(1 - e^{-q\cdot n_l/w}\right)^d.$$

Proof. The baseline sketch differs from SketchFeature in that it utilizes hard partitioning to divide the sketch space into $1/q$ quantized bins, each independently used for quantized bin-level operations. In the same memory setting, each sketch with a width of w/q, and the l-th quantized bin sketch of *baseline* contains n_l elements. By leveraging the phantom decoding probability of Lemma 1, the phantom decoding probability of the l-th quantized bin sketch in *baseline* can be computed. $\square$

Theorem 1. *If the number of elements encoded in the l-th quantized bin sketch of the baseline sketch exceeds $1/q$ of the entire dataset, then the phantom decoding probability of baseline is greater than or equal to that of sketch virtualization. In other words,*

$$\textit{If } \frac{1}{q} \le \frac{n_l}{n}, \textit{ then } P'_{\textit{phantom}} \ge P_{\textit{phantom}}.$$

Proof. When comparing Lemma 2 with Lemma 1, if $n \le q\cdot n_l$, the phantom decoding probability of *baseline* is greater than or equal to that of *sketch virtualization*. Therefore, when a quantized bin has more than $1/q$ elements of the entire dataset, the phantom decoding probability in the bin of *baseline* is greater than that of *sketch virtualization*. $\square$

TABLE I: Notions

Params	Description
n	The number of all distinct flows
n_i	The number of distinct tuples in the i-th quantized bin
q	The number of quantized bins in total
k	The average number of packets per flow
b	The average number of bins occupied by a flow
d	The number of layers of the baseline sketch and SketchFeature
w	The number of counters in each layer of SketchFeature
$a_{f_{ij}}$	The actual count of the i-th flow in the j-th bin ($i \le n$, $j \le q$)
A_j	The number of counts in the j-th quantized bin, $A_j = \sum_i a_{f_{ij}}$
A	The total number of counts in all bins, $A = \sum_j A_j = nk$

Theorem 1 leads to a strong demand for membership tests, which eliminates the adverse effect of negative queries by limiting phantom decoding probability in every bin.

B. Error Bound Analysis of Sketch Virtualization

In this section, we compare the error bounds of the baseline and SketchFeature based on the error bound proof of Count-Min sketch. The proof assumes that decoding is performed only for the elements that have been encoded, which is justified by the membership test algorithm using Bloom filter [9] can suppress the phantom decoding probability to a negligible extent. For a fair comparison, only the sketch error excluding errors caused by phantom decoding issues is taken into account in both schemes. While Count-Min sketch is designed to collect 2nd-order features, the baseline sketch and SketchFeature utilize it as a primitive for the 3rd-order sketch with different encoding and decoding methods. To analyze the baseline sketch and SketchFeature using the error bound analysis of the Count-Min sketch, we define the notation as shown in Table I. Also, we set the width of SketchFeature as $w = \lceil e/\epsilon \rceil$ and the number of layers as $d = \lceil \ln(1/\delta) \rceil$ for the error bound parameter ϵ and the probability guarantee parameter δ.

Theorem 2 (Error Bound of SketchFeature)**.** *The decoded feature value $\hat{a}_{f_{ij}}$ of the i-th flow in the j-th bin has the following guarantees: $a_{f_{ij}} \le \hat{a}_{f_{ij}}$; and with probability at least $1-\delta$,*

$$\hat{a}_{f_{ij}} \le a_{f_{ij}} + \epsilon \cdot \sum_j A_j.$$

Proof. In a sketch extracting 2nd-order features of network traffic, the key to distinguish a different element is the flow ID (e.g., source IP, 5 tuples). However, SketchFeature utilizes a tuple composed of the flow ID and quantization level (i.e., quantized bin number) as the key to determine the encoding counter among w counters in the sketch. Therefore, the total number of distinct elements (or flows), denoted as n, is defined not by the count of distinct flow IDs but by the count of distinct tuples consisting of flow ID and quantization level. Here, we note that Count-Min's error bound is limited not by the total number of distinct elements (or flows) but by the total number of data (or packets). Assuming no phantom decoding issue, the error bound of SketchFeature is equivalent to that of a Count-Min sketch encoding and querying $n \times q$ distinct elements. $\square$

Theorem 3 (Error Bound of Baseline)**.** *When the decoded feature value $\hat{a}_{f_{ij}}$ belongs to the j-th quantized bin sketch, it has the following guarantees: $a_{f_{ij}} \le \hat{a}_{f_{ij}}$; and with probability at least $1-\delta$,*

$$\hat{a}_{f_{ij}} \le a_{f_{ij}} + q \cdot \epsilon \cdot \sum_i a_{f_{ij}},$$

where q is the total number of quantized bins, and $\sum_i a_{f_{ij}} = A_j$ is the number of packets encoded in the j-th quantized bin.

Proof. The baseline sketch differs from SketchFeature in that it employs hard partitioning, creating q individual quantized bin sketches, and encodes elements into the respective sketches. Consequently, only A_j packets belonging to n_j flows are encoded in the j-th quantized bin sketch. Assuming the baseline sketch has the same memory as SketchFeature, the width of *baseline*'s quantized bins is approximately w/q. As a result, the error bound of the j-th quantized bin sketch in the baseline sketch can be considered equivalent to that of a Count-Min sketch with a width of w/q, encoded results of A_j packets belonging to n_j flows are in the j-th quantized bin. $\square$

From the perspective of the feature value distribution, Theorem 2 and Theorem 3 imply that SketchFeature's estimation has the same error bound for all elements across bins, while the baseline sketch's error bounds vary depending on the distribution of the dataset over q bins. That is, the baseline sketch's measurement accuracy estimation is unstable giving fluctuating error bounds ($q \cdot \epsilon \cdot A_j$) for a bin j, whereas SketchFeature can be seen as adding relatively uniform errors ($\epsilon \cdot \sum A_j$) to all elements across all quantized bins.

1) Total Error Bound Comparison: In the previous section, we examined the error bounds of the decoded feature values for both baseline and SketchFeature for a specific element (or flow). In this section, we will compare the total sum of error bounds for all elements, considering the distribution of the dataset. To utilize the error bounds derived from Theorems 2 and 3, we assume that for sufficiently large d and sufficiently small δ, all elements satisfy the defined error bounds. Before introducing the main theorem, a theorem is presented to find a range of inner products of two probability distributions, flow cardinality probability distribution and packet size probability distribution over quantized bins.

Theorem 4. *Let $k = \sum A_i/n$, and $b = \sum n_i/n$, where A_i is the number of packets encoded in i-th bin by SketchFeature, and n is the total number of distinct flows. We assume $q \ge 2$ and $k/b \ge (q+1)/(q-1)$. Let $p = (p_1, \dots, p_q)$ be the probability distribution of the number of distinct elements (or flows) over q quantized bins, where $p_l = n_l/nb$ is a fraction of distinct flows in the l-th quantized bin over the total number of distinct flow-bin pairs. That is, p is the flow cardinality probability distribution. Also, let $\hat{p} = (\hat{p}_1, \dots, \hat{p}_q)$ be the probability distribution of the number of packets over q bins, where $\hat{p}_l = A_l/nk$ is the fraction of packets encoded in the l-th quantized bin over the total number of packets. Then, the inner product of the two probability distributions p and $\hat{p}$ satisfies*

$$\frac{b}{k(q-1)} \le (p_1, p_2, \dots, p_q) \cdot (\hat{p}_1, \hat{p}_2, \dots, \hat{p}_q) \le 1$$

Proof. The upper bound $p \cdot \hat{p} \le 1$ follows from the Cauchy–Schwarz inequality. Below we prove the lower bound $p \cdot \hat{p} \ge \frac{b}{k(q-1)}$.

Since $A_l \ge n_l$, $l = 1, \dots, q$, we have

$$\hat{p}_l \ge \frac{b}{k} p_l, \quad l = 1, \dots, q.$$

Therefore, $p \cdot \hat{p} \ge m$, where

$$m = \min_{(x,y) \in C} x \cdot y.$$

Here, C is the set of all $(x, y) \in \mathbb{R}^q \times \mathbb{R}^q$ such that $x = (x_1, \dots, x_q)$ and $y = (y_1, \dots, y_q)$ are probability vectors satisfying

$$x_l \ge \frac{b}{k} y_l, \quad l = 1, \dots, q.$$

Note that $\min\limits_{(x,y) \in C} x \cdot y$ exists, since $x \cdot y$ is continuous in (x, y) on a compact set C. Now, it suffices to show $m = \frac{b}{k(q-1)}$ to complete the proof.

For every $((x_1, \dots, x_q), (y_1, \dots, y_q)) \in C$, there is a permutation $(\sigma_1, \dots, \sigma_q)$ such that $y_{\sigma_1} \le \cdots \le y_{\sigma_q}$. Since $((x_{\sigma_1}, \dots, x_{\sigma_q}), (y_{\sigma_1}, \dots, y_{\sigma_q})) \in C$ and $(x_1, \dots, x_q) \cdot (y_1, \dots, y_q) = (x_{\sigma_1}, \dots, x_{\sigma_q}) \cdot (y_{\sigma_1}, \dots, y_{\sigma_q})$, we have

$$m = \min_{(x,y) \in C_1} x \cdot y,$$

where $C_1 = \{(x, y) \in C : y_1 \le \cdots \le y_q\}$.

If $(x, y) \in C_1$, then $x \cdot y \ge \tilde{x} \cdot y$, where

$$\tilde{x}_l = \begin{cases} 1 - \sum_{j=2}^q \frac{b}{k} y_j, & \text{if } l = 1 \\ \frac{b}{k} y_l, & \text{if } 2 \le l \le q. \end{cases}$$

This is verified as follows:

$$\begin{aligned} & x \cdot y - \tilde{x} \cdot y \\ &= \sum_{l=2}^q (x_l - \tilde{x}_l) \cdot y_l \\ &= \left(\left(1 - \sum_{l=2}^q x_l \right) - \left(1 - \sum_{l=2}^q \tilde{x}_l \right) \right) y_1 + \sum_{l=2}^q (x_l - \tilde{x}_l) \cdot y_l \\ &= \sum_{l=2}^q (x_l - \tilde{x}_l) \cdot (y_l - y_1) \\ &\ge 0, \end{aligned}$$

since $x_l - \tilde{x}_l \ge 0$ and $y_l - y_1 \ge 0$ for $2 \le l \le q$. Therefore, $x \cdot y \ge \tilde{x} \cdot y$. Furthermore, $(\tilde{x}, y) \in C_1$, since $\tilde{x}_1 \ge x_1 \ge \frac{b}{k} y_1$ and $\tilde{x}_l = \frac{b}{k} y_l$ for $2 \le l \le q$. Hence,

$$m = \min_{(x,y) \in C_2} x \cdot y,$$

where $C_2 = \{(x, y) \in C_1 : x_l = \frac{b}{k} y_l \text{ for } 2 \le l \le q\}$.

If $(x, y) \in C_2$, then $x \cdot y \ge \tilde{x} \cdot \tilde{y}$, where $\tilde{x} = (\tilde{x}_1, \dots, \tilde{x}_q)$ and $\tilde{y} = (\tilde{y}_1, \dots, \tilde{y}_q)$ are given by

$$\tilde{x}_l = \begin{cases} x_1, & \text{if } l = 1, \\ \frac{1}{q-1} \sum_{j=2}^q x_j, & \text{if } 2 \le l \le q, \end{cases}$$

$$\tilde{y}_l = \begin{cases} y_1, & \text{if } l = 1, \\ \frac{1}{q-1} \sum_{j=2}^q y_j, & \text{if } 2 \le l \le q. \end{cases}$$

Therefore,

$$m = \min_{(x,y)\in C_3} x \cdot y,$$

where $C_3 = \{(x, y) \in C_2 : x_2 = \cdots = x_q\}$. Note that C_3 is the set of all $(x, y) \in \mathbb{R}^q \times \mathbb{R}^q$ such that

$$0 \le y_1 \le y_2 = \cdots = y_q, \quad \sum_{l=1}^{q} y_l = 1,$$

$$x_2 = \cdots = x_q = \frac{b}{k} y_2, \quad x_1 = 1 - \frac{(q-1)b}{k} y_2.$$

Therefore, $(x, y) \in C_3$ if and only if there is $t \in \mathbb{R}$ such that

$$x = \left(1 - \frac{b(1-t)}{k}, \frac{b}{k}\frac{1-t}{q-1}, \ldots, \frac{b}{k}\frac{1-t}{q-1}\right), \tag{1}$$

$$y = \left(t, \frac{1-t}{q-1}, \ldots, \frac{1-t}{q-1}\right), \tag{2}$$

$$0 \le t \le \frac{1-t}{q-1}.$$

Note that $0 \le t \le \frac{1-t}{q-1}$ if and only if $0 \le t \le \frac{1}{q}$. For x and y given by (1) and (2), we have

$$x \cdot y = \frac{bqt^2 + ((k-b)(q-1) - 2b)\,t + b}{k(q-1)}, \tag{3}$$

which is increasing in t on $[0, \frac{1}{q}]$. Therefore, m is given by (3) evaluated at $t = 0$. Thus, $m = \frac{b}{k(q-1)}$. $\square$

Now, we are ready to calculate the range of error bounds.

Theorem 5. *Let's denote the sum of error bounds for all elements of SketchFeature and baseline sketch as σ_{sf} and $\sigma_{baseline}$, respectively. Then, the following inequality holds for a quantized distribution with q bins:*

$$\frac{1}{q} \cdot \sigma_{baseline} \le \sigma_{sf} \le \frac{k(q-1)}{qb} \cdot \sigma_{baseline}.$$

where $k = \sum A_i/n$, and $b = \sum n_i/n$, where $\sum A_i$ is the total number of packets encoded in SketchFeature, and n is the total number of distinct flows, assuming $q \ge 2$ and $k \ge b$.

Proof. For SketchFeature, due to memory virtualization for multiple sketches, every element has the same error bound. According to Theorem 2 and the fact that only nb flow and bin pairs are to be decoded, it has a total error bound as follows:

$$\sigma_{sf} = \sum_{i=1}^{n}\sum_{j=1}^{q}(a_i - \hat{a}_i) \le nb \cdot \epsilon \cdot \sum A, \tag{4}$$

For the baseline sketch, depending on the distribution of the packet features (PS, IPD), the error bounds vary across q quantized bin sketches. The error bound for the l-th quantized bin sketch is $q \cdot \epsilon \cdot \sum A_l$ as proved in Theorem 3. Given that the number of distinct elements encoded in the sketch is n_l, the sum of error bounds can be expressed as follows:

$$\sigma_{baseline} \le q \cdot \epsilon \cdot (n_1 \cdot \sum A_1 + n_2 \cdot \sum A_2 + \ldots + n_q \cdot \sum A_q), \tag{5}$$

Since $n_1 + n_2 + \ldots + n_q = nb$, we can substitute $n_l = p_l \cdot nb$ for a flow's cardinality probability distribution p_l where $p_1 + p_2 + \ldots + p_q = 1$. p_l is a fraction of distinct flows in the l-th quantized bin over the total sum of the number of distinct flows over bins. Additionally, since $\sum A_1 + \sum A_2 + \ldots + \sum A_q = \sum A$, we can similarly substitute $\sum A_l = \hat{p}_l \cdot \sum A$ for a packet size probability distribution $\hat{p}_l$, where where $\hat{p}_1 + \hat{p}_2 + \ldots + \hat{p}_q = 1$. Using Eq. 4 and replacing n_l and $\sum A_l$ with $p_l \cdot nb$ and $\hat{p}_l \cdot \sum A$ in Eq. 5, respectively, we obtain

$$\begin{aligned}\sigma_{baseline} &\le q \cdot nb \cdot \epsilon \cdot \sum A \cdot (p_1 \cdot \hat{p}_1 + p_2 \cdot \hat{p}_2 + \ldots + p_q \cdot \hat{p}_q)\\ &= q \cdot (p_1, p_2, \ldots, p_q) \cdot (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_q) \cdot \sigma_{sf},\end{aligned} \tag{6}$$

By directly applying Theorem 4,

$$\frac{b}{k(q-1)} \cdot q \cdot \sigma_{sf} \le \sigma_{baseline} \le q \cdot \sigma_{sf}.$$

By rearranging the above for σ_{sf}, we finally obtain

$$\frac{1}{q} \cdot \sigma_{baseline} \le \sigma_{sf} \le \frac{k(q-1)}{qb} \cdot \sigma_{baseline} \quad \square$$

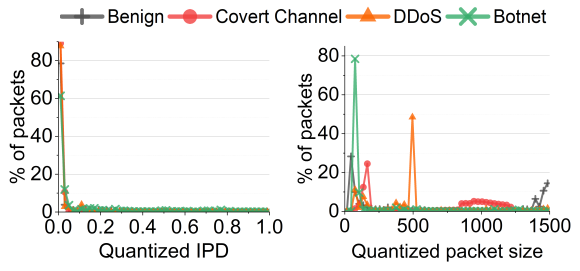
Implications. Observing Eq. 6, $\sigma_{baseline}$ is handicapped by q factors and advantaged by the inner product. q handicap is obvious considering the case that every packet is inserted into one bin. Also, the advantageous case for the baseline sketch happens when the inner product $P \cdot Q$ has $1/q$ when either packets or flows are uniformly distributed to q bins, since $(p_1 = 1/q, p_2 = 1/q, \ldots, p_q = 1/q) \cdot (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_q) = 1/q \cdot \sum \hat{p}_i = 1/q$. By Eq. 6,

$$\sigma_{baseline} = q \cdot \frac{1}{q} \cdot \sigma_{sf} = \sigma_{sf},$$

which means the baseline sketch's total error is the same as SketchFeature's. This observation supports our idea of *sketch virtualization*, where any given input distribution is converted into a uniform distribution by *sketch virtualization*'s memory address space randomization. This address randomization of *sketch virtualization* is equivalent to the case that uniformly distributed cardinality and packet size distributions are fed to the baseline sketch. This signifies the memory-efficient scenario of the baseline sketch, where it evenly distributes n/q data points across q sketches. However, in the case of a skewed distribution, the worst-case scenario occurs when all n elements are inserted into a single quantized bin. In this case, the baseline sketch achieves the same total error bound as SketchFeature encoding n elements with $1/q$ of the memory. The upper bound of the total error occurs only when the baseline sketch is fed by data with a very peculiar distribution (one bin with a large number of flows and the others with uniformly distributed flows) shown in Theorem 4. This implies that SketchFeature can be improved further if we can distribute traffic in a specific form, which is not practical. Considering that the 3rd-order traffic distribution is highly skewed in reality, the total error of SketchFeature distributes somewhere around $1/q$ of the baseline sketch.

V. Evaluation

In this section, we analyze SketchFeature's feature extraction performance varying memory and workload. Next, we conduct an end-to-end system evaluation, comparing the SketchFeature-based system with baseline, perfect, and state-of-the-art systems, leveraging diverse attack detection tasks.



(a) IPD packet distribution (b) PS packet distribution

Fig. 10: The percentage of packets within each bin when quantizing 5-second benign and malicious traces into 50 bins.

A. Metrics

Weighted Mean Relative Error (WMRE). WMRE is used to evaluate distribution feature quality. It is represented by $\frac{\sum_{i=1}^{n}|a_i-\hat{a_i}|}{\sum_{i=1}^{n}(\frac{a_i+\hat{a_i}}{2})}$, where n is the number of distinct bins, and a_i and $\hat{a_i}$ are the actual and estimated value for each bin, respectively.

Area Under the ROC Curve (AUC). We used AUC to evaluate the model's performance, considering all possible classification thresholds for a robust comparison. This metric quantifies the performance of attack detection models by comparing the True Positive Rate (TPR) and False Positive Rate (FPR) at various classification thresholds (ROC curve).

F1 Score. We used the F1 score to evaluate the accuracy of the binary classification model. It is defined as the harmonic mean of precision (the proportion of true positive results in all positive predictions) and recall (the proportion of true positive results in all actual positives) to denote the trade-off between precision and recall.

Accuracy. Accuracy refers to the measure of the correctness of a model's predictions compared to the actual values. It is calculated as the ratio of the number of correct predictions to the total number of predictions made by the model.

False Negative and Positive Rate (FNR and FPR). We used FNR and FPR to show the target flow prediction performance of a model under the imbalanced dataset. FNR measures the proportion of actual positive instances (attacks) that are incorrectly classified as negative (benign) by the model. FPR measures the proportion of actual negative (benign) instances that are incorrectly classified as positive (attack) by the model.

B. Feature Quality: Per-flow Distribution

To verify the design effectiveness, we compare the feature accuracy of SketchFeature with the baseline sketch varying measurement tasks, memory space, and workload.

Dataset. We used four different datasets, including CAIDA (benign) [1], Facet covert storage channel (CSC) [31], CIC (DDoS) [43], and botnet (Botnet) [8]. The covert timing channel (CTC) trace was excluded from the feature quality analysis as it represents a synthetic dataset [50]. For a realistic scenario, each attack trace was mixed separately with CAIDA benign traffic to distinguish security use cases. For feature integrity, the original packet order with raw timestamps is preserved during the dataset mixture.

Settings. SketchFeature records per-flow 3rd-order (vector) features, including inter-packet delay (IPD) and packet size (PS) distribution, which significantly increases the number of possible distinct elements for encoding. This amplification is proportional to the feature granularity (i.e., the number of quantization bins). For instance, in our 3rd-order feature quality experiments, we quantize each per-flow feature into 50 bins, which increases the number of distinct (encodable) elements by up to 50 times compared to 2nd-order (scalar) feature measurement tasks. Fig. 10 illustrates quantized IPD and PS distributions of our datasets. Here, we quantize IPD features up to 1 second and PS features from 1 to 1500 bytes, within 50 bins. For a fair comparison, we allocate equal memory space to all schemes. Within the given memory budget, SketchFeature reserves 2 MB specifically for the Bloom filter-based membership test for suppressing phantom decoding.

Accuracy Varying Memory. Fig. 11 compares the feature quality of baseline and SketchFeature with 5-second traces by varying memory settings. With different traces (i.e., Benign, CSC, DDoS, and Botnet), Fig. 11(a)-(d) illustrate IPD feature accuracy and Fig. 11 (e)-(h) depict PS feature accuracy. As shown, SketchFeature consistently achieves a significantly lower WMRE across all tasks compared to the baseline. Notably, in attack trace measurements, SketchFeature achieves near-zero error as memory increases, and in benign trace measurements, we observe a rapid error reduction with additional memory allocation. These results validate both our experimental (section III-C) and theoretical analyses (section IV-B), demonstrating the effectiveness of SketchFeature's design at scale. Meanwhile, SketchFeature maintains high performance even under extremely low-memory conditions, with 1 MB allocated to Sketch and 2 MB to the Bloom filter within a 3 MB total memory budget. This efficiency is contributed by SketchFeature's sketch virtualization design balancing bins across the entire memory space and suppressed phantom decoding with the membership test function. In contrast, the baseline employs static memory partitioning for bins, which limits memory efficiency due to highly skewed bin distributions.

Accuracy Varying Workload. Next, we fixed the total memory space to 6 MB and varied the workload up to 60 seconds of trace for a pressure test. Fig. 12 (a)-(h) show the WMRE over time by varying features and traces. With the increased pressure, the sketch memory will be saturated. However, we observe SketchFeature achieves non-compromised feature accuracy for attack flow measurement and consistently outperforms baseline, thanks to sketch virtualization and phantom-free decoding designs. We noticed that SketchFeature shows rapid growth of errors for the benign trace measurement due to the domination of mice flows, which are sensitive to small noises. However, we found that mice flow impact on false positives is negligible for ML-based applications (see section V-C and Table II for details).

C. End-to-end System Evaluation: Security Use Cases

Next, we provide the end-to-end system evaluation and compare SketchFeature with the existing data plane solutions that leverage 3rd-order [6], [50] and 2nd-order [63] features varying security use cases. To evaluate the robustness and efficacy of SketchFeature, we also compare SketchFeature with perfect measurement and the baseline sketch (see section III),

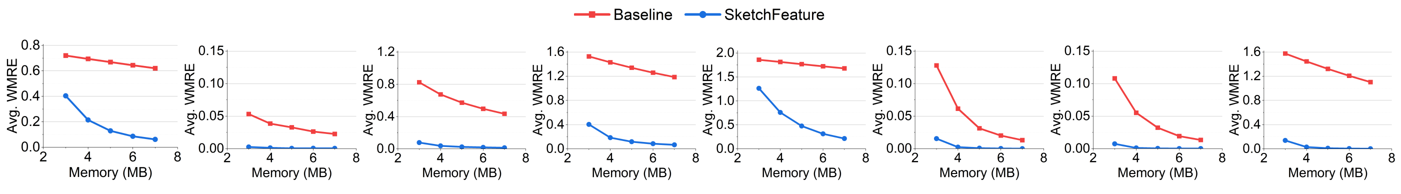

(a) Benign (IPD) (b) CSC (IPD) (c) DDoS (IPD) (d) Botnet (IPD) (e) Benign (PS) (f) CSC (PS) (g) DDoS (PS) (h) Botnet (PS)

Fig. 11: Comparison of 3rd-order (vector) feature quality. Average WMRE of distribution features (IPD and PS) varying memory.

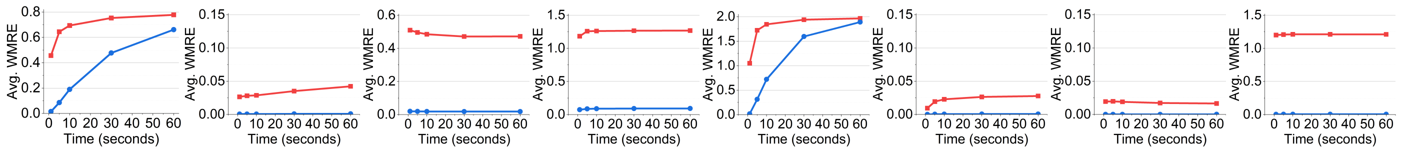

(a) Benign (IPD) (b) CSC (IPD) (c) DDoS (IPD) (d) Botnet (IPD) (e) Benign (PS) (f) CSC (PS) (g) DDoS (PS) (h) Botnet (PS)

Fig. 12: Comparison of 3rd-order (vector) feature quality. Average WMRE of distribution features (IPD and PS) varying workload.

which represents a feasible solution for HAF 3rd-order feature extraction in the data plane.

Dataset. We varied security use cases with four attack traces, including Covert Storage Channel (*CSC*), Covert Timing Channel (*CTC*), Distributed Denial of Service (*DDoS*), and botnet (*Botnet*) traffic. For each attack type, we used 80% of attack flows for training and 20% for testing, with each set mixed separately with 5 seconds of CAIDA [1] as background traffic. For CSC, we selected all 1 K attack flows from traffic encoded by Facet [31], a censorship resistance tool over Skype. For CTC, we followed [42] to craft a dataset containing 6 K attack flows for encoding 4 bits of data by varying IPD across different ranges and combinations. For DDoS, we selected 10 K attack flows from two categories of DDoS, namely, reflection (e.g., NTP, DNS) and flooding (e.g., UDP, LDAP, SNMP) from CIC-DDoS [43]. For botnet, we selected 8.6 K attack flows from ISCX botnet dataset [8], containing malicious flows of Neris and Virut botnets.

Settings. We adhere to the originally proposed feature extraction design tailored to detect specific types of attacks for each scheme. In CTC detection, we quantize the IPD feature values for up to 5 seconds in 1000 bins. In CSC, DDoS, and botnet detections, we quantize the PS feature values up to 1500 Bytes in 94 bins [6]. For a fair comparison, we allocated equal resources in the switch's data plane and control plane for all schemes. In the data plane, we allocated 6 MB of data plane memory for all schemes for feature extraction. In the control plane, we used a 1-D Convolutional Neural Network (CNN) model with three layers of 1-D convolutional layers (followed by max-pooling layers) followed by three layers of Multi-Layer Perceptron (MLP) with Rectified Linear Unit (ReLU) activation function and dropout rate of 0.2 to prevent overfitting. To ensure robustness, the collected feature was tested and averaged over 10 independently trained models. To prevent temporal data snooping [5], we eliminated time dependency between the training and test datasets.

Table II compares performance of SketchFeature with the other systems across security use cases presented in their respective original studies.

Compared to Baseline and Perfect Measures. The baseline serves as a feasible data plane solution for HAF feature extraction using hard partitioning of the sketch, yet its accuracy is heavily influenced by traffic distribution. As shown in Fig. 10, **CSC** attacks exhibit a distinct pattern compared to benign traffic, resulting in baseline performance very close to perfect measurement demonstrated in Table II. In contrast, in **CTC**, **DDoS**, and **Botnet** attack scenarios, the baseline approach performs poorly due to poor feature quality for both attack and benign traffic, resulting in extremely poor F1 scores (<0.20) and AUC values (<0.60) overall. Promisingly, SketchFeature achieves comparable performance with perfect measure in terms of AUC, accuracy, FPR, and FNR. However, for the DDoS and botnet use cases, we observed approximately 13% reduction in F1 scores, dropping to 0.746 and 0.715, respectively, due to the unavoidable sketch noise. Despite this, SketchFeature still outperforms all existing approaches.

Compared to NetBeacon [63]. In **CSC** detection, NetBeacon measures top-K bins only among a full range of the PS distribution (3rd-order) feature, with strong assumption of prior knowledge. We follow the configuration outlined in the original work: top-6 quantized bins, including [96, 112), [112, 128), [128, 144), [144, 160), [160, 176) and [176, 192). While its mechanism for predicting long and short flows enables full-flow coverage and good accuracy, our findings indicate that incorporating the full range of 3rd-order features can yield even better performance. In **DDoS** detection, NetBeacon mainly works with 2nd-order features (i.e., per-flow minimum IPD and minimum PS features), and alternates to 1st-order features (i.e., per-packet time-to-live, packet size, and protocol fields) in the case of resource shortage. The results verify the superiority of 3rd-order HAF features achieved by SketchFeature, with significantly improved AUC by 25.6% and F1 score by 11%.

Compared to NetWarden [50]. For NetWarden [50], we followed the original work to perform feature extraction in the control plane and further utilized the KS test [41] for **CTC** attack detection. As shown in Table II, NetWarden achieved near-optimal performance. The reasons are twofold: first, it assumed perfect suspicious flow (symptom) detection in the data plane with prior knowledge (i.e., attack IPD > 1 ms); second, it assumed unlimited resources in the control plane for all packet measurements of the data plane elevated flows, for extracting perfect per-flow IPD distribution features. Besides the relaxed assumptions, such an approach is vulnerable under a control

TABLE II: End-to-end system evaluation.

Schemes	AUC	Acc.	F1	FPR	FNR
Covert Storage Channel (CSC)					
Perfect Measure	0.999	0.999	0.981	0.0	0.020
Baseline	0.986	0.999	0.978	0.0	0.027
NetBeacon [63]	0.995	0.981	0.983	0.018	0.018
FlowLens [6]	0.502	0.998	0.132	0.0	0.928
SketchFeature	**0.998**	**0.999**	**0.981**	**0.0**	**0.019**
Covert Timing Channel (CTC)					
Perfect Measure	1.0	1.0	1.0	0.0	0.0
Baseline	0.526	0.230	0.090	0.798	0.148
NetWarden [50]	0.999	0.999	0.999	0.0	0.0
FlowLens [6]	0.501	0.956	0.029	0.0	0.985
SketchFeature	**0.999**	**0.996**	**0.965**	**0.003**	**0.0**
Distributed Denial of Service (DDoS)					
Perfect Measure	0.997	0.997	0.877	0.001	0.125
Baseline	0.561	0.991	0.197	0.0	0.875
NetBeacon [63]	0.738	0.995	0.636	0.0	0.521
FlowLens [6]	0.504	0.991	0.126	0.0	0.932
SketchFeature	**0.994**	**0.994**	**0.746**	**0.004**	**0.130**
Botnet					
Perfect Measure	0.922	0.997	0.845	0.001	0.142
Baseline	0.494	0.979	0.007	0.012	0.978
FlowLens [6]	0.498	0.992	0.106	0.0	0.942
SketchFeature	**0.914**	**0.995**	**0.715**	**0.003**	**0.172**

plane flooding attack without a complex countermeasure (see section II). In comparison, the proposed SketchFeature, with data-plane-based HAF feature measurement, achieved near-optimal results without prior knowledge of the attack and exposing a clear attack surface to adversaries.

Compared to FlowLens [6]. The hash table-based FlowLens [6] is configured to measure the full range of features, with 1000 quantized bins for **CTC** attack and 94 bins for **CSC**, **DDoS**, and **Botnet** attack scenarios. As shown in Table II, FlowLens demonstrates poor performance for all tasks, especially in botnet detection. On average, it covered only 12.6% of total flows, leading to the omission of more than 90% of attack flows, a high FNR of 0.942, and a low AUC of 0.498. Exact per-flow measurement using a hash table contributed to the irreconcilable trade-off between flow and feature scalability, which cannot but scarify partial flows or features granularity for fixed memory space. On the contrary, SketchFeature breaks through the trade-off by a sketch design and achieves HAF feature extraction.

VI. Discussions

Adversary with Pre-knowledge of SketchFeature. In our threat model, adversaries are not allowed to access the switch's internal states, which means the attacker has to guess the seed value of the hash function to exploit the hash collision of the targeted flow, which is not trivial. Poisoning the entire sketch with an imbalanced feature distribution is infeasible, given the uniform and random counter-sharing nature of SketchFeature, thanks to sketch virtualization design. Moreover, the HAF feature measurement design allows SketchFeature to observe all pattern shifts at a fine-grained level, benefiting advanced ML technologies that address concept drift issues.

INDP Implementation. On Tofino-1 switch [2], SketchFeature consumes only 7 stages out of a total of 12 stages. We note that the up-to-date Tofino-2 switch equips 20 stages with much more memory and computation resources [3], which will allow SketchFeature to coexist with more complex and computation-intensive applications, such as in-data-plane ML. However, we note that current in-data-plane ML schemes [63], [54] support up to 2nd-order features, and are not yet compatible with 3rd-order features, but we anticipate advanced ML solutions soon with the current trend. Lastly, we emphasize that SketchFeature's feature quality is independent of where ML functions are deployed: data plane or control plane.

VII. Related Work

Sketch is an essential tool for traffic measurement, and it has proven its efficiency in operating on compact small memory resources [46], [26], [22], [18], [27]. Especially with the memory constraint in ASIC hardware switches [10], [16], they have become popular for traffic measurement with the main focus on extracting lower-dimensional 1st-order or 2nd-order [45], [39], [44], [60], [56], [23], [29], [40] features. Recently, a new phase of In-Network Intrusion Detection Systems (IN-IDS) has emerged, aiming to eliminate the feature transfer delay between the data plane and the control plane by extracting features and leveraging an AI module to detect attacks simultaneously [48], [52], [30], [62], [11], [47], [49]. However, lower-dimensional features showed limitations in attack detection [63], highlighting the need for richer 3rd-order vector features. FlowLens [6] realized IN-IDS by extracting 3rd-order vector features using a table with top-K quantized bins to store flows. Another 3rd-order symptom detector, NetWarden [50], utilizes sketches to enable full-range measurements for all flows, but it suffers from low resolution. There is also a body of works [24], [61], [33] proposed to collect partial distribution-relation features, such as tail quantile over a data stream, which differs from the focus of HAF feature collection in the data plane by SketchFeature design. The latest work, HistSketch [25], aimed to realize per-flow item distribution by extending FlowLens's exact measurement, supported by remote data centers with unlimited resources. This differs from SketchFeature, the first 3rd-order feature collection sketch designed for the switches' data plane.

VIII. Conclusion

In this paper, we propose SketchFeature, a novel approach to collect 3rd-order features, including per-flow Packet Size (PS) distribution and Inter-Packet Delay (IPD). These features have been essential data for the recent success of AI for malicious traffic detection. However, the collection of such an approach is challenging due to the memory constraint in the data plane. In this manner, we have devised two attack mechanisms that can bypass the collection and measurement of such a feature, successfully evading the detection logic of traffic. Then, we proposed a sketch virtualization technique to collect High-resolution, All-flow, and Full-range (HAF) per-flow 3rd-order feature extraction. Finally, through our extensive theoretical and empirical analysis, we have shown the high quality and scalable feature collection for effective attack detection.

Acknowledgment

The authors thank the anonymous shepherd and reviewers for their valuable feedback. This work is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) No. RS-2023-00222385 and No. RS-2023-NR077143, and the National Science Foundation (NSF) under grant IIS-2211897. DaeHun Nyang and Rhongho Jang are the corresponding authors.

References

[1] "The cooperative association for internet data analysis, equinix chicago data center," [13:00-14:00, Apr 19 2018, from Sao Paulo to New York]. [Online]. Available: https://www.caida.org

[2] "Edgecore WEDGE 100BF-32X." [Online]. Available: https://www.edge-core.com/product/dcs800/

[3] "Intel Tofino-2." [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html

[4] S. Abbasloo, C. Yen, and H. J. Chao, "Classic meets modern: a pragmatic learning-based congestion control for the internet," in *Proc. of ACM SIGCOMM*, 2020.

[5] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *Proc. of USENIX Security*, 2022.

[6] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *Proc. of ISOC NDSS*, 2021.

[7] D. Barradas, N. Santos, and L. E. T. Rodrigues, "Effective detection of multimedia protocol tunneling using machine learning," in *Proc. of USENIX Security*, 2018.

[8] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *Proc. of IEEE CNS*, 2014.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[11] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pforest: In-network inference with random forests," *arXiv preprint arXiv:1909.05680*, 2019.

[12] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert timing channels: design and detection," in *Proc. of ACM CCS*, 2004.

[13] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. of ACM SIGCOMM*, 2018.

[14] X. Chen, S. L. Feibish, M. Braverman, and J. Rexford, "Beaucoup: Answering many network traffic queries, one memory update at a time," in *Proc. of ACM SIGCOMM*, 2020.

[15] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[16] P. Cui, H. Pan, Z. Li, J. Wu, S. Zhang, X. Yang, H. Guan, and G. Xie, "Netfc: Enabling accurate floating-point arithmetic on programmable switches," in *Proc. of IEEE ICNP*, 2021.

[17] D. Dao, R. Jang, C. Jung, D. Mohaisen, and D. Nyang, "Minimizing noise in hyperloglog-based spread estimation of multiple flows," in *Proc. of IEEE/IFIP DSN*, 2022.

[18] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

[19] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proc. of ACM CCS*, 2021.

[20] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," in *Proc. of ISOC NDSS*, 2023.

[21] S. Gianvecchio and H. Wang, "Detecting covert timing channels: an entropy-based approach," in *Proc. of ACM CCS*, 2007.

[22] A. Goyal and H. D. III, "Approximate scalable bounded space sketch for large data NLP," in *Proc. of EMNLP*, 2011.

[23] L. Gu, Y. Tian, W. Chen, Z. Wei, C. Wang, and X. Zhang, "Per-flow network measurement with distributed sketch," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 411–426, 2024.

[24] J. Guo, Y. Hong, Y. Wu, Y. Liu, T. Yang, and B. Cui, "Sketchpolymer: Estimate per-item tail quantile using one sketch," in *Proc. of ACM SIGKDD*, 2023.

[25] J. He, J. Zhu, and Q. Huang, "Histsketch: A compact data structure for accurate per-key distribution monitoring," in *Proc. of IEEE ICDE*, 2023.

[26] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proc. of ACM SIGCOMM*, 2017.

[27] R. Jang, S. Moon, Y. Noh, A. Mohaisen, and D. Nyang, "Instameasure: Instant per-flow detection using large in-dram working set of active flows," in *Proc. of IEEE ICDCS*, 2019.

[28] C. Jung, S. Kim, R. Jang, D. Mohaisen, and D. Nyang, "A scalable and dynamic acl system for in-network defense," in *Proc. of ACM CCS*, 2022.

[29] S. Kim, C. Jung, R. Jang, D. Mohaisen, and D. Nyang, "A robust counting sketch for data plane intrusion detection," in *Proc. of ISOC NDSS*, 2023.

[30] J.-H. Lee and K. Singh, "Switchtree: In-network computing and traffic analyses with random forests," *Neural Computing and Applications*, pp. 1–12, 2020.

[31] S. Li, M. Schliep, and N. Hopper, "Facet: Streaming over videoconferencing for censorship circumvention," in *Proc. of Workshop on Privacy in the Electronic Society*, 2014.

[32] C. Liu, M. Xu, Y. Yang, and N. Geng, "DRL-OR: deep reinforcement learning-based online routing for multi-type service requirements," in *Proc. of IEEE INFOCOM*, 2021.

[33] J. Liu, H. Dai, R. Xia, M. Li, R. B. Basat, R. Li, and G. Chen, "Duet: A generic framework for finding special quadratic elements in data streams," in *Proc. of ACM WWW*, 2022.

[34] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches," in *Proc. of USENIX Security*, 2021.

[35] X. Luo, E. W. Chan, and R. K. Chang, "Tcp covert timing channels: Design and detection," in *Proc. of IEEE DSN*, 2008.

[36] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. of ACM SIGCOMM*, 2017.

[37] S. M. M. Mirnajafizadeh, A. R. Sethuram, D. Mohaisen, D. Nyang, and R. Jang, "Enhancing network attack detection with distributed and in-network data collection system," in *Proc. of USENIX Security*, 2024.

[38] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *Proc. of ISOC NDSS*, 2018.

[39] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "SketchLib: Enabling efficient sketch-based monitoring on programmable switches," in *Proc. of USENIX NSDI*, 2022.

[40] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Stenkiste, "Sketchovsky: Enabling ensembles of sketches on programmable switches," in *Proc. of USENIX NSDI*, 2023.

[41] P. Peng, P. Ning, and D. S. Reeves, "On the secrecy of timing-based active watermarking trace-back techniques," in *Proc. of IEEE S&P*, 2006.

[42] S. H. Sellke, C.-C. Wang, S. Bagchi, and N. Shroff, "Tcp/ip timing channels: Theory to implementation," in *Proc. of IEEE INFOCOM*, 2009.

[43] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *Proc. of IEEE ICCST*, 2019.

[44] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "Fcm-sketch: generic network measurements with data plane support," in *Proc. of ACM CoNext*, 2020.

[45] L. Tang, Q. Huang, and P. P. Lee, "A fast and compact invertible sketch for network-wide heavy flow detection," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2350–2363, 2020.

[46] M. Tirmazi, R. B. Basat, J. Gao, and M. Yu, "Cheetah: Accelerating database queries with switch pruning," in *Proc. of ACM SIGMOD*, 2020.

[47] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *Proc. of IEEE INFOCOM*, 2021.

[48] G. Xie, Q. Li, C. Cui, P. Zhu, D. Zhao, W. Shi, Z. Qi, Y. Jiang, and X. Xiao, "Soter: Deep learning enhanced in-network attack detection based on programmable switches," in *Proc. of IEEE SRDS*, 2022.

[49] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *Proc. of IEEE INFOCOM*, 2022.

[50] J. Xing, Q. Kang, and A. Chen, "NetWarden: Mitigating network covert channels while preserving performance," in *Proc. of USENIX Security*, 2020.

[51] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," in *Proc. of USENIX Security*, 2021.

[52] Z. Xiong and N. Zilberman, "Do switches dream of machine learning?: Toward in-network classification," in *Proc. of ACM Workshop on Hot Topics in Networks*, 2019.

[53] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. A. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *Proc. of USENIX NSDI*, 2020.

[54] J. Yan, H. Xu, Z. Liu, Q. Li, K. Xu, M. Xu, and J. Wu, "Brain-on-switch: Towards advanced intelligent network data plane via nn-driven traffic analysis at line-speed," in *Proc. of USENIX NSDI*, 2024.

[55] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng, "ACC: automatic ECN tuning for high-speed datacenter networks," in *Proc. of ACM SIGCOMM*, 2021.

[56] K. Yang, S. Long, Q. Shi, Y. Li, Z. Liu, Y. Wu, T. Yang, and Z. Jia, "Sketchint: Empowering INT with towersketch for per-flow per-switch measurement," *IEEE Trans. Parallel Distributed Syst.*, vol. 34, no. 11, pp. 2876–2894, 2023.

[57] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: adaptive and fast network-wide measurements," in *Proc. of ACM SIGCOMM*, 2018.

[58] C. Zhang, Z. Cai, W. Chen, X. Luo, and J. Yin, "Flow level detection and filtering of low-rate ddos," *Comput. Networks*, vol. 56, no. 15, pp. 3417–3431, 2012.

[59] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *Proc. of ISOC NDSS*, 2020.

[60] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "Cocosketch: High-performance sketch-based measurement over arbitrary partial key query," in *Proc. of ACM SIGCOMM*, 2021.

[61] F. Zhao, S. Maiyya, R. Wiener, D. Agrawal, and A. E. Abbadi, "Kll±approximate quantile sketches over dynamic datasets," in *Proc. of VLDB Endowment*, 2021.

[62] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proc. of ACM SIGCOMM*, 2021.

[63] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *Proc. of USENIX Security*, 2023.

[64] G. Zhou, G. Chen, F. Lin, T. Xu, D. Wei, J. Wu, L. Chen, Y. Lu, A. Qu, H. Shao, and H. Jiang, "Primus: Fast and robust centralized routing for large-scale data center networks," in *Proc. of IEEE INFOCOM*, 2021.

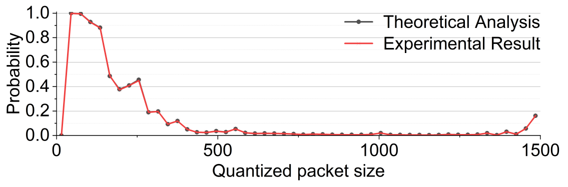
[65] H. Zhou, S. Hong, Y. Liu, X. Luo, W. Li, and G. Gu, "Mew: Enabling large-scale and dynamic link-flooding defenses on programmable switches," in *Proc. of IEEE S&P*, 2023.

[66] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," in *Proc. of ACM SIGCOMM*, 2021.

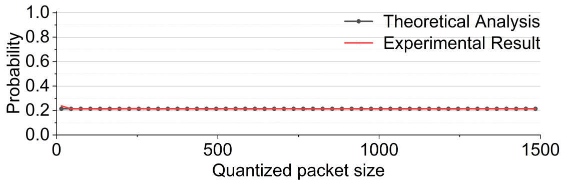
Appendix

A. Evaluation Based on the Proof

Sketch virtualization utilizes the given memory space as a single sketch, distinguishing positions solely by hashes, regardless of which quantized bin the data belongs to. Consequently, any never-encoded element querying it would yield the same phantom decoding probability. On the other hand, the baseline sketch divides the given memory space into q sketches, each encoding quantized bins separately. As a result, the phantom decoding probability increases with the presence of more data in individual quantized bin sketches. Fig. 13 illustrates the phantom decoding probability in each quantized bin sketch when inserting benign datasets into the baseline sketch and *sketch virtualization*, based on both theorem and implementation approaches. As evident from the figure, both baseline sketch and *sketch virtualization* exhibit implementation experiment results similar to what was proven in the theorem. Furthermore, the feature distribution of the dataset has a similar trend to the phantom decoding probability of the baseline sketch, confirming that the probability of phantom decoding is higher in regions where the dataset is more abundant. In the case of *sketch virtualization*, since it shares a single sketch across all quantized bin sketches, memory usage is more balanced, leading to generally lower phantom decoding probability. Furthermore, *sketch virtualization* does not have partitions, resulting in phantom decoding occurring at similar probabilities across all quantized bin sketches regardless of dataset distribution.



(a) Baseline Sketch



(b) Sketch Virtualization

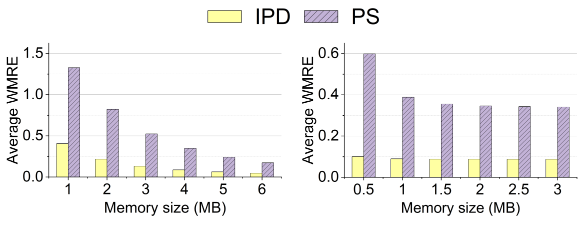
Fig. 13: Phantom decoding probability when encoding the packet size distribution of 2.6 M benign traces using the baseline sketch and *sketch virtualization* with 50 quantized bins in 6 MB of memory. The theorem result is obtained by substituting the parameters of the dataset into the theorem, while the implementation result is obtained by inserting the dataset into the implemented sketches.

B. Analysis Varying Parameters

Next, we evaluate SketchFeature by varying memory, configuration, and feature granularity (quantization bins).

Varying Sketch Memory. Fig. 14 (a) illustrates the variation in WMRE when varying the size of the sketch and fixing the Bloom filter at 2 MB. Due to the sufficient size of the Bloom filter, phantom decoding rarely occurs, allowing us to observe the variations in the error introduced by the sketch. As memory increases, the probability of multiple flows colliding in a single counter decreases, enabling both IPD and PS distributions to be measured more accurately. The PS distribution has a higher WMRE than the IPD distribution because the PS distribution of the benign dataset contains more distinct flow and feature pairs than the IPD distribution. In other words, when measuring the PS distribution, more counters in the sketch remain non-empty by encoding more distinct QL values per flow.

Varying Bloom-Filter Memory. To analyze the impact of Bloom filter size on accuracy, we kept the sketch size constant at 4 MB and increased the Bloom filter size in steps of 0.5 MB. As shown in Fig. 14 (b), when measuring IPD, the issue of phantom decoding is not significant due to the smaller number of distinct flow and feature pairs. This results in minimal changes in WMRE, even with increased memory size. However, for PS distribution, the variation in WMRE is relatively significant. It can be observed that using a Bloom filter of approximately 2 MB or larger effectively mitigates accuracy degradation caused by the phantom decoding issue.



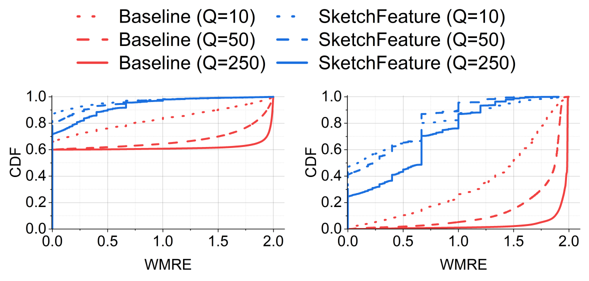
(a) Varying sketch size (b) Varying bloom filter size

Fig. 14: The average WMRE of the flow-wise feature distribution measured by SketchFeature with 50 quantized bins, while varying the size of Bloom filter and the sketch.

Varying Quantized Bins. Fig. 15 shows the accuracy difference between the baseline sketch and SketchFeature by varying the number of quantized bins Q. Since the baseline sketch divides the memory space for several independent sketches with varying bins, increasing the number of bins reduces the memory size allocated to each sketch. Consequently, the baseline sketch's accuracy tends to degrade more as the number of quantized bins (Q) increases in both IPD and PS distribution measurements. In SketchFeature, increasing the number of quantized bins is equivalent to increasing the number of flow and feature pairs, resulting in more counters in the sketch during encoding. For example, when Q is 3, a flow uses up to 3 different sketches, whereas when Q is 10, it needs to use 10 different sketches. Consequently, flow-wise hash collisions occur more frequently with more quantized bins. However, due to the randomized memory virtualization scheme, SketchFeature exhibits relatively small performance degradation compared to *baseline* even with 250 quantized bins.

C. 2rd-order Feature: Aggregated (Scalar) Features

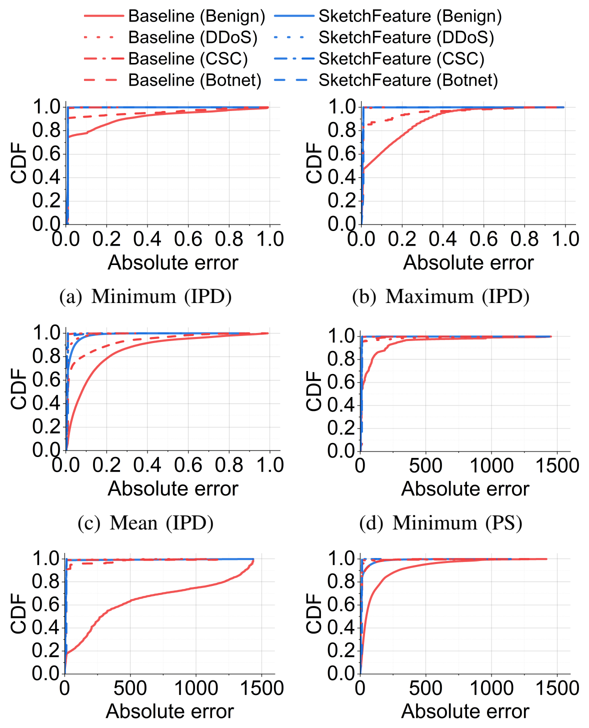
We further decoded the 3rd-order feature of flows to obtain 2nd-order features, such as minimum, maximum, and mean. As shown in Fig. 16 (a)-(f), SketchFeature shows the best accuracy, achieving almost zero absolute error across all metrics and varying security applications (see blue lines at the top). When extracting minimum and maximum values,



(a) Inter-packet delay (IPD) (b) Packet size (PS)

Fig. 15: The Cumulative Distribution Function of WMRE for flow-wise feature distribution measured while varying the number of quantized bins from 10 to 250, with 2 MB Bloom filter and 4 MB sketch.

SketchFeature benefits from the improved memory efficiency of sketch virtualization and suppressed phantom decoding via membership test. More specifically, with the relaxed collision, since the Bloom filter can almost accurately determine which bin a flow has been encoded into, SketchFeature can extract minimum and maximum values with much higher accuracy than the baseline sketch. The accuracy for minimum IPD of the baseline sketch is comparable to that of SketchFeature as in Fig. 16 (a). This is because the IPD distribution is highly skewed towards the first bin QL_1, as shown in Fig. 10 (a). In the PS distribution, where such favorable conditions to the baseline sketch are not given, SketchFeature outperforms the baseline sketch. Overall, SketchFeature shows stable and robust performance for all cases, whereas the baseline is sensitive to traffic characteristics.



(a) Minimum (IPD) (b) Maximum (IPD)

(c) Mean (IPD) (d) Minimum (PS)

(e) Maximum (PS) (f) Mean (PS)

Fig. 16: Comparing 2nd-order (scalar) feature quality. Absolute errors of statistics aggregated from 3rd-order measures.

D. P4 Implementation and Hardware Resource Utilization

We implemented a prototype of SketchFeature in the data plane of Tofino-1 fabric-based programmable switch [2]. The code snippet can be found in Appendix D. The hardware resource utilization after data plane deployment is demonstrated in Table III. We configured a total of 3 MB SRAM memory space for SketchFeature, where 1.5 MB for the Bloom filter and 1.5 MB for sketch, showing 35.54% SRAM memory usage. Also, only 0.35% of TCAM memory was used for range matching for feature quantization. Besides, 11.11% of the hash unit was used for sketch memory randomization. Lastly, 12.50% of the ALU and 1.82% of the VLIW were utilized for logic construction. Notably, the resource usage presented in the table is the average across 7 stages, not 12 stages. Thus, 5 stages remain fully available, allowing for the deployment of additional programs.

The data plane module was implemented using the P4 language [10] in Barefoot SDE 9.0.0. The implemented code snippet of SketchFeature is shown in Code 1. The function `table_get_QL` maps the current packet size of an arriving packet to one of the 50 quantized bins (lines 1∼15). When the table is applied, the quantization level is stored in user-defined metadata for inter-stage data communication through the `get_QL` function (lines 34 and 18). Subsequently, the hash value obtained by hashing the flow ID, which is the five-tuple, and the quantization level is used to update the Bloom filter and sketch (lines 37∼43). Due to the limited hardware environment of the data plane, where we cannot use different hash functions across layers, we resolved this issue through hash slicing.

TABLE III: Hardware resource utilization of SketchFeature.

Resources	SRAM	TCAM	Hash Unit	ALU	VLIW
Usage	35.54%	0.60%	19.05%	21.43%	3.12%

```
1  table table_get_QL {
2      key = {
3          hdr.ipv4.total_len : range;
4      }
5      actions = {
6          get_QL;
7      }
8      const entries = {
9          0 .. 30 : get_QL(16w1);
10         31 .. 60 : get_QL(16w2);
11         ...
12         1471 .. 1500 : get_QL(16w50);
13     }
14     size = 50;
15 }
16
17 action get_QL(bit<16> ql) {
18     meta.ql = ql;
19 }
20
21 Register<bit<1>, bit<BF_BIT>>(BF_SIZE) bloom_filter_l1;
22 Register<bit<1>, bit<BF_BIT>>(BF_SIZE) bloom_filter_l2;
23 Register<bit<1>, bit<BF_BIT>>(BF_SIZE) bloom_filter_l3;
24 Register<bit<32>, bit<S_BIT>>(S_SIZE) sketch_l1;
25 Register<bit<32>, bit<S_BIT>>(S_SIZE) sketch_l2;
26 Register<bit<32>, bit<S_BIT>>(S_SIZE) sketch_l3;
27
28 RegisterAction<bit<32>, bit<S_BIT>, bit<32>>(sketch_l1) update_l1 = {
29     void apply (inout bit<32> value, out bit<32> result) {
30         value = value + 32w1;
31     }
32 };
33
34 table_get_QL.apply();
35 bit<32> hash_val = hash_function.get({meta.flow_id, meta.ql});
36
37 bloom_filter_l1.write(hash_val[BF_BIT-1:0],1w1);
38 bloom_filter_l2.write(hash_val[BF_BIT+3:4],1w1);
39 bloom_filter_l3.write(hash_val[BF_BIT+7:8],1w1);
40
41 update_l1.execute(hash_val[S_BIT+1:2]);
42 update_l2.execute(hash_val[S_BIT+5:6]);
43 update_l3.execute(hash_val[S_BIT+9:10]);
```

Code 1: Data Plane Implementation