Duet: Towards Efficient and Accurate Sketch-Based Measurement on DPUs

Yang Wu*¶, Songlin Chen*¶, Fuliang Li*✉, Man Hou‡, Jiaxing Shen§, Xingwei Wang*
*School of Computer Science and Engineering, Northeastern University, Shenyang, 110819,
‡Zhongguancun Laboratory, Beijing, China, §Lingnan University, Hong Kong
Email: wuyang@mails.neu.edu.cn, 2490245@stu.neu.edu.cn, lifuliang@cse.neu.edu.cn,
houman@zgclab.edu.cn, jiaxingshen@LN.edu.hk, wangxw@mail.neu.edu.cn

***Abstract*—Sketch-based measurement has become a fundamental primitive for monitoring network traffic and supporting performance-critical data center applications. However, existing approaches for deploying sketches on DPUs either underutilize DPU hardware capabilities or suffer from degraded accuracy due to limited Arm-side processing capacity and multi-core contention. In particular, most designs directly port host-side sketch implementations to the DPU Arm subsystem, introducing excessive forwarding overhead and hardware pressure. In this paper, we present Duet, an efficient and accurate sketch-based measurement on DPUs. Duet adopts a layered measurement architecture that dynamically schedules flows between Arm-based sketches and the DPU hardware forwarding pipeline according to traffic characteristics. Heavy flows are offloaded to hardware tables to leverage line-rate processing and relieve Arm-side contention, while light flows are retained in software sketches to avoid excessive hardware table updates. To further address cross-layer aggregation and multi-core concurrency challenges, Duet incorporates a zeroing mechanism and lock-free optimizations to ensure measurement consistency and accuracy. We implement Duet on an NVIDIA BlueField-2 DPU and conduct extensive experiments on a real testbed. Experimental results demonstrate that, compared with traditional sketch-based measurement approaches, Duet improves throughput by a factor of approximately $7.1\times$, packet rate by a factor of approximately $28.9\times$, and measurement accuracy by approximately 50.1%.**

***Index Terms*—DPU, Sketch, Software Offloading, Network Measurement, Hardware–Software Co-Design.**

I. INTRODUCTION

Traditional host-based measurement approaches can no longer keep pace with the stringent requirements of modern data centers in terms of accuracy, performance, and resource efficiency. For example, in multi-tenant environments, sketches [1]–[4] are typically executed on the host CPU using system-call-based approaches, tracing-based mechanisms, or DPDK-based frameworks. However, these solutions face inherent trade-offs: system-call and tracing-based approaches often suffer from limited accuracy and insufficient throughput [5]–[7], while DPDK-based solutions achieve higher performance at the cost of substantial CPU resource consumption. With the evolution of network hardware [8]–[10], major vendors have begun deploying Data Processing Units (DPUs) in data centers, offloading network functions such as sketch-based measurement from the host CPU to dedicated hardware.

¶ Yang Wu and Songlin Chen contributed equally to this work.

Offloading sketches to DPUs offers several compelling advantages: First, DPUs operate directly on the network data path and can process packets at line rate without introducing additional host interrupts or context switches, fundamentally eliminating interference with host applications [11]. Second, DPUs provide relatively isolated and controllable compute and memory resources, enabling sketch performance to remain stable and predictable even under high load [12]. Finally, sketches deployed on DPUs exhibit strong scalability and reusability, allowing them to be uniformly deployed across different hosts and multi-tenant environments without requiring intrusive modifications to the host software stack [13].

However, existing measurement frameworks fail to fully exploit the strengths of both sketches and DPU hardware.

(1) Fail to leverage DPU's high performance. A DPU typically consists of an Arm-based subsystem responsible for general-purpose computation and a hardware forwarding pipeline that executes packet processing at high speed based on installed rules. Most existing sketch-based approaches directly inherit host-side designs and require packets to be diverted from the forwarding pipeline to the Arm subsystem for measurement. This additional processing step introduces extra overhead and reduces overall throughput. Moreover, as the number of hardware table entries grows, the memory footprint and computational cost associated with maintaining connection state on the DPU increase, leading to frequent context switches on the Arm cores, which further degrades forwarding performance.

(2) Fail to preserve sketch's accuracy benefits. Existing approaches primarily implement sketch logic on the Arm subsystem, whose single-core performance and instruction throughput are significantly lower than those of host-side x86 processors. When packet arrival rates exceed the processing capacity of the Arm cores, packet drops become inevitable, directly compromising measurement accuracy. In addition, sketches are often updated concurrently by multiple Arm cores, which can introduce inconsistencies such as stale reads and redundant updates. Enforcing consistency through locking or coherence mechanisms further slows down packet processing, exacerbating packet loss and indirectly reducing measurement accuracy.

(3) lack deep co-design between sketches and DPU. Most

measurement frameworks treat sketch logic and DPU hardware as loosely coupled components: sketches are executed passively on the Arm subsystem, while the forwarding pipeline independently handles high-speed packet processing. This separation prevents sketches from being aware of real-time data-plane states and limits their ability to offload measurement logic into hardware to reduce Arm-side load. As a result, these designs neither alleviate multi-core consistency issues nor fully capitalize on the performance potential of the DPU.

Design Overview. To address the fundamental tension between performance and accuracy when deploying sketch-based measurement on DPU, we present Duet, a hardware–software co-designed measurement framework that tightly integrates sketches with DPU hardware. Duet adopts a layered measurement architecture that decomposes measurement logic across the Arm subsystem and the DPU hardware data plane.

By default, the majority of flows are measured at the Arm layer using sketches. When the traffic volume of a flow persistently grows beyond a predefined threshold, its corresponding flow entry is dynamically offloaded to the DPU's hardware forwarding table. After offloading, subsequent packets of this flow bypass the Arm subsystem and are measured and forwarded directly in hardware, avoiding sustained pressure from high-volume traffic on the Arm execution path.

Key Insight. The key insight behind Duet is that network traffic exhibits substantial heterogeneity along two dimensions: packet volume and flow table cardinality. Meanwhile, the Arm subsystem and the DPU hardware forwarding pipeline possess complementary strengths when handling these two types of workloads. Specifically, heavy flows that generate a high packet volume but correspond to a relatively small number of flow entries are better suited for offloading into the DPU hardware tables, where they can fully leverage the high-throughput hardware data path and reduce contention on the software execution path.

Guided by this insight, Duet performs fine-grained layering and dynamic scheduling of measurement tasks to balance the computational load on the Arm subsystem with the utilization of DPU hardware resources. To further address cross-layer aggregation and multi-core concurrency introduced by the layered design, Duet incorporates a zeroing mechanism and lock-free multi-core optimizations, reducing statistical errors and improving the consistency and accuracy of measurement results. Overall, Duet embodies a tightly integrated measurement–forwarding co-design paradigm, offering a new approach to achieving both high performance and high accuracy for sketch-based measurement on DPU platforms.

Overall, these results indicate that deep sketch–DPU co-design is both necessary and effective, which motivates the contributions of this paper as follows:

1) **Characterization.** We conduct an initial characterization of sketch-based measurement on DPU platforms and identify hardware flow table entries as a key performance-limiting factor. Our study reveals that increasing hardware table occupancy amplifies the computational and memory pressure on the Arm subsystem, leading to degraded sketch performance and reduced measurement accuracy.
2) **Design.** We propose Duet, a hardware–software co-designed measurement framework that tightly integrates sketches with DPU hardware. Duet dynamically schedules flows between sketches on the Arm subsystem and hardware flow tables based on traffic volume and packet distribution, effectively balancing Arm-side computation and DPU hardware resource utilization. To address cross-layer consistency and multi-core concurrency, Duet further introduces a zeroing mechanism and lock-free multi-core optimizations to reduce measurement errors.
3) **Evaluation.** We implement Duet on a DPU-based platform and conduct extensive experiments to demonstrate that Duet achieves higher throughput, improved measurement accuracy, and better scalability compared to existing approaches.

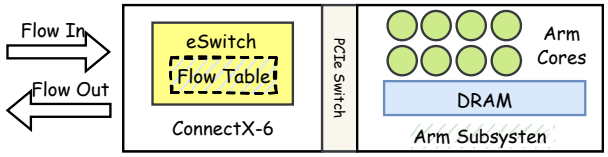

Fig. 1: Architecture of the NVIDIA BlueField-2 DPU.

II. BACKGROUND AND MOTIVATION

A. Sketch-based Measurement on DPUs

Sketch. A class of probabilistic data structures designed for large-scale data stream measurement and are widely used in network traffic statistics, heavy-hitter detection, and anomaly identification. Their primary goal is to enable approximate measurement of traffic characteristics under memory constraints and high-speed data streams, while providing efficient and scalable operation with bounded error. Compared to traditional per-flow measurement, sketches trade a small amount of accuracy for substantially reduced storage and computational overhead, making them well suited for line-rate processing in high-speed networks. However, existing sketch-based measurement systems lack a unified design that is deeply aligned with DPU architectures, and thus fail to fully exploit the DPU's potential for tightly coupled computation and forwarding.

DPUs. We use the NVIDIA BlueField-2 (BF2) DPU as our target platform (Figure. 1), as it represents a widely deployed class of DPU architectures in modern data centers. BF2 adopts a highly integrated system-on-a-chip (SoC) design, aiming to offload and isolate network and I/O infrastructure functions that are traditionally handled by the host CPU.

Arm Subsystem. The Arm subsystem serves as the general-purpose computation and control-plane foundation of the BlueField-2 DPU. It typically consists of multiple Arm processor cores along with associated memory and peripherals, and is capable of running a full Linux operating system independently. The Arm subsystem is responsible for managing and controlling infrastructure services such as networking, storage,

Item\EntryNum	0	200K	400K	600K	800K	1M
CS	1.9K	4.3K	6.5K	8.8K	11K	13K
IN	1.2K	2.7K	4.2K	5.8K	7.2K	8.6K

TABLE I: Arm Subsystem Performance

EntryNum\CoreNum	1	2	3	4	5	6
500K	77K	30K	29K	27K	22K	17K
1M	63K	44K	25K	21K	14K	10K

TABLE II: Multi Core Flow Table Offloading Rate

and security.In a typical DPU design, the Arm cores primarily handle control-plane tasks—including protocol processing, resource management, and policy decision making—while delegating high-throughput and low-latency data-plane operations to dedicated hardware acceleration units.

Flow Table. DOCA Flow is a core library in NVIDIA's DOCA framework that abstracts and programs the DPU's internal hardware data path. It provides a unified API to developers while mapping these abstractions to the underlying hardware flow tables and forwarding logic of ConnectX-series network controllers. DOCA Flow serves as the key mechanism bridging the Arm control plane and the hardware data plane, and adopts a classic match–action programming model. The core abstractions in this model include pipes, match rules, monitor (mon) actions, modify (mdf) operations, and forward (fwd) directives.

As illustrated in Figure 1, DOCA Flow can be used to implement line-rate traffic measurement directly in the hardware data path. When a packet enters a pipe, it is matched against flow rules; upon a successful match, counting operations are performed in the monitor stage. The packet header can then be modified via the mdf stage, and finally forwarded according to the fwd action. This entire processing pipeline is executed at line rate within the hardware data path, without per-packet involvement from the Arm subsystem, thereby significantly improving overall processing performance.

Measurement on DPUs. Although DOCA Flow enables high-performance packet processing in the hardware data path, practical traffic measurement inevitably requires the involvement of the Arm control plane. Due to limited hardware resources, the data plane cannot comprehensively measure all flows; instead, the control plane must dynamically manage and update hardware flow tables to select and maintain the set of flows to be measured. In addition, Sketch is typically implemented directly within the Arm subsystem without relying on the DPU's hardware tables. The general-purpose processing capability of the Arm subsystem enables the realization of complex operations and computations required by Sketch. Furthermore, the Arm subsystem commonly employs multiple cores for parallel packet processing. Therefore, Sketch implementations running on the Arm subsystem must explicitly consider multi-core characteristics in order to fully unleash the performance potential of the DPU. Existing Sketch frameworks fail to effectively integrate with the DPU's hardware flow tables and its multi-core architectural characteristics.

B. Existing Limitations and Motivations

Our further experimental observations show that these control-plane operations impose significant overhead on the Arm subsystem and become a performance bottleneck under high traffic load. Sketches are deployed on the Arm subsystem of the DPU, and their measurement performance is therefore inherently constrained by the processing capability of the Arm cores.

Setup: We evaluate the runtime overhead incurred on the Arm subsystem when offloading flow rules to the DPU hardware flow tables via DOCA Flow. After the program starts, multiple worker threads are created, each pinned to a dedicated Arm core. These threads concurrently install flow table entries corresponding to distinct five-tuples into the hardware data plane, with the Counter action enabled for every entry.

Result: We use vmstat to monitor the number of context switches and interrupts, thereby quantifying the system overhead introduced by control-plane operations. The results are summarized in Table I. As the number of hardware flow table entries increases, the number of context switches (CS) on the Arm subsystem rises from approximately 2,000 to 14,000, while the number of interrupts (IN) increases from around 1,100 to 9,000. These results indicate that, as more rules are offloaded to the hardware data plane, the Arm subsystem spends a growing amount of time on flow table management and scheduling-related control operations. Consequently, its ability to concurrently execute measurement tasks such as Sketch is significantly impaired.

We further evaluate the entries offloading rate per core under different numbers of Arm cores, with the results shown in Table II. When two Arm cores are used, the offloading rate reaches its peak at approximately 110k rules per second. However, as more cores are added, the offloading rate unexpectedly decreases. This phenomenon suggests that increasing the number of Arm cores introduces substantial synchronization and inter-core communication overhead, which ultimately degrades the efficiency of hardware flow table management. These findings highlight the necessity of designing more efficient multi-core coordination mechanisms to mitigate such overheads.

Motivation. Motivation: The above experiments reveal that the major bottleneck of sketch-based measurement on DPUs does not come from the existence of hardware flow-table entries themselves, but from frequent and indiscriminate flow-table management on the Arm subsystem. When a large number of flows are continuously inserted into or updated in the hardware tables, the Arm cores spend substantial CPU cycles on control-plane operations such as rule installation, synchronization, and scheduling. This overhead reduces the processing capacity available for software sketches and eventually degrades both throughput and measurement accuracy. However, avoiding hardware tables entirely is also inefficient, because high-volume flows continuously consume Arm-side processing resources if they remain in software sketches. This

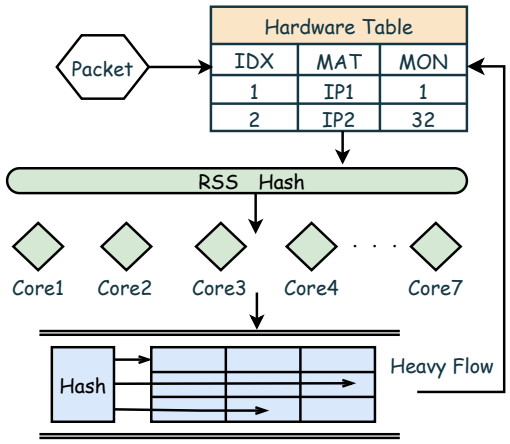

Fig. 2: Overview of Data Structure

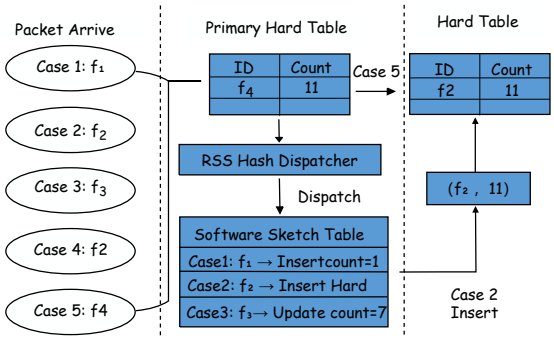

Fig. 3: Five Cases in the Sketch-Hardware Measurement Framework.

creates a trade-off: blindly offloading all flows introduces excessive control-plane overhead, while keeping all flows in software overloads the Arm subsystem under high traffic rates.

Duet is motivated by the observation that real-world traffic is typically heavy-tailed: a small number of flows contribute a large fraction of packets, while most flows are short-lived or low-volume. Therefore, hardware offloading should be selective rather than exhaustive. Duet uses software sketches on the Arm subsystem to monitor incoming flows and identify flows whose packet volume persistently exceeds a predefined threshold. Only these stable heavy flows are migrated to the DPU hardware flow table, where subsequent packets can be counted and forwarded at line rate. In contrast, light flows remain in the software sketch, avoiding unnecessary hardware rule insertions.

In this way, Duet reconciles the tension between control-plane overhead and data-plane acceleration. By limiting hardware-table updates to a small set of high-benefit heavy flows, Duet reduces Arm-side measurement load while avoiding the excessive rule-management cost caused by offloading all flows.

III. Design

In this section, we first propose the basic Duet, then introduce the optimization methods and the optimized algorithm.

A. Basic Version

1) Overview of Data Structure: The data structure of the basic Duet is shown in Fig. 2. As illustrated, this data structure serves as the core carrier for hierarchical flow table offloading, adopting a hierarchical architecture of "hardware high-speed layer + software layer". It includes three core modules: hardware table, RSS hash distribution component, and software Sketch, supporting differentiated processing and dynamic switching of heavy and light flows.

The difference between the hardware table and the software table lies in that the hardware table stores matching rules and states of heavy flows to achieve line-speed flow matching [14], forwarding, and counting — when a packet hits the hardware table, it is directly processed and output; as the carrier for light flow processing in the software layer, the sketch is responsible for matching and counting maintenance of local light flows; it utilizes multi-core parallel capabilities to improve the overall processing throughput of light flows. In short, this design mainly includes the following two key ideas:

- **Intelligent Separation of Heavy and Light Flows:** As the "distribution hub" when the hardware table is not hit, RSS directionally distributes unmatched packets to different compute cores through hash operations, providing scheduling support for multi-core parallel processing of light flows and avoiding single-core performance bottlenecks. Heavy flows and light flows are distinguished based on a predefined threshold — heavy flows are stored in the hardware table, and light flows are stored in the software Sketch. This solves the contradiction between limited hardware resources and unbalanced traffic distribution.
- **Hierarchical Accelerated Processing:** heavy flows are offloaded to the DPU eSwitch hardware flow table, leveraging hardware acceleration to achieve forwarding and counting with ≤ 1 microsecond latency; light flows are processed in parallel by Arm multi-cores through software Sketch, balancing performance and flexibility.

2) Case Study: To intuitively understand these concepts, we now analyze five insertion cases in Fig. 3. To intuitively present the workflow and the logic for processing heavy and light flows of the basic Duet, the following analysis is conducted based on 4 typical data packet processing cases (Note: In the cases, the five-tuple (source IP, destination IP, source port, destination port, protocol number) of a flow is used as the unique identifier (ID), the number of data packets is used as the counting metric, and the predefined heavy flow judgment threshold is 10):

Case 1: First Access of a New Light Flow, After a data packet with flow ID $f1$ enters the system, it is first submitted to the hardware flow table for matching query. The system determines that its five-tuple entry is not stored, indicating that the flow is an unidentified new flow. Subsequently, the data

packet is forwarded to the RSS hash distribution component, which directs it to a specific computing core through hash operation. The core first queries the count of $f1$ in the local software Sketch table and finds that the count is 0 (indicating the first access of the flow). It only performs the insertion operation in the Sketch table, updating the corresponding counter value of $f1$ to 1, thus completing the initial storage of the new light flow.

Case 2: Light Flow Grows into a Heavy Flow Triggering Hardware Migration, After a data packet with flow ID $f2$ accesses the system, the hardware flow table fails to match it. The packet is then distributed to a designated computing core via RSS hash. The core queries the software Sketch table and finds that the cumulative count of $f2$ is 10, which is equal to the predefined heavy flow threshold. Thus, the flow is determined to have grown into a heavy flow. At this point, the system terminates subsequent operations of the software Sketch table, inserts the five-tuple matching rules and count status of $f2$ into the hardware flow table, completing the migration of the heavy flow from software to hardware. Subsequent processing of this flow will be directly undertaken by the hardware flow table.

Case 3: Light Flow Continues Accumulating Without Reaching the Threshold, After a data packet with flow ID $f3$ accesses the system, the hardware flow table fails to match it. The packet is distributed to a designated computing core via RSS hash. The core queries the software Sketch table and finds that the cumulative count of $f3$ is 6, which is less than the heavy flow threshold of 10. Therefore, the flow is still determined to be a light flow. The system performs the update operation in the Sketch table, increments the corresponding counter value of $f3$ by 1 (updating it to 7), and completes the count accumulation of the light flow.

Case 4: Subsequent Processing of a Migrated Heavy Flow, When a data packet with flow ID $f2$ accesses the system again (having been migrated to the hardware flow table previously), the system first performs a matching query through the hardware flow table and finds that its five-tuple entry is stored. At this point, there is no need to go through RSS hash distribution or software Sketch table query. The system directly increments the count by 1 at the corresponding counter position in the hardware flow table, realizing line-rate counting and fast forwarding of the heavy flow.

Case 5: Aging Removal of Stale Heavy Flow in Hardware Table for Clearing Obsolete Data Flows, Flow ID $f4$ has been migrated to the hardware flow table in advance with a cumulative count of 11, and the hardware flow table embeds a real-time aging detection mechanism that continuously monitors the packet arrival timestamp of each hardware table entry. When the mechanism detects that the latest packet arrival timestamp of $f4$ exceeds the predefined aging timeout threshold, the flow is determined to be a stale heavy flow with no incoming packets for a long time. At this point, the system triggers the aging removal operation of hardware table entries, and completely deletes the five-tuple matching rules, count status and other related entries of $f4$ from the hardware flow table, realizing the cleaning function of obsolete old data flows. If subsequent data packets of $f4$ enter the system again, the hardware flow table will fail to match the entry of $f4$, and the flow will be reprocessed according to the processing procedure of a new unknown flow (Case 1).

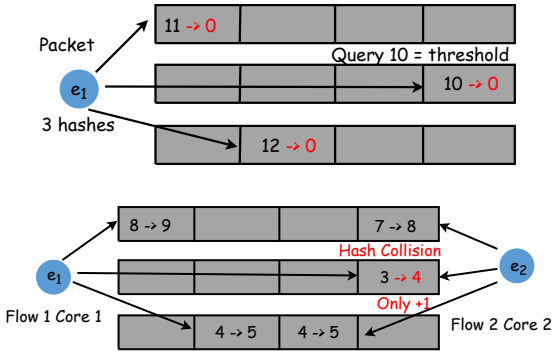

Fig. 4: Optimization Mechanisms.

B. Optimization Mechanisms

The inherent hash collision of the Sketch algorithm will affect measurement accuracy. Although multi-core parallel processing can improve performance, it may introduce lock overhead and data race. How to improve performance by adopting a multi-threaded parallel architecture while reducing collisions through optimized Sketch design to achieve "co-existence of high performance and high accuracy" is a key technical bottleneck that Duet needs to break through. This addresses the challenge of balancing Sketch performance and accuracy on DPU [15]–[17].

1) Counter Reset: The introduction of the resetting operation not only enhances the capability of the Sketch structure to mitigate hash collisions, but also effectively eliminates the abnormal overestimation of light-flow counts caused by hash collisions between heavy and light flows. In conventional Sketch designs, when a light flow and a heavy flow are mapped to the same counter, the measurement of the light flow can be severely contaminated by the heavy flow, resulting in significant overcounting. By introducing a counter reset mechanism, the accumulated errors introduced by hash collisions can be promptly eliminated. Counters associated with newly identified heavy flows are reset to zero and restart counting, which significantly reduces the interference of heavy flows on light-flow measurements and improves the accuracy and stability of traffic measurement. Its core principle can be further illustrated with the diagram below:

The reset process is triggered when the software Sketch detects that the count of a flow reaches a predefined threshold (as marked in the diagram: Query 10 = threshold, i.e., the threshold is set to 10), identifies the flow as a heavy flow, and the heavy flow has been successfully offloaded to the hardware flow table.

Extract the 5-tuple information of the heavy flow, then calculate its index positions in all counter arrays using the 3

built-in hash functions ($h1$,$h2$,$h3$) of the Sketch. Next, subtract the cumulative count of this heavy flow in the Sketch from the counter values corresponding to these indices (e.g., 11, 10, 12 in the diagram), and finally reset these counters to 0 in a targeted manner. This eliminates the interference of the heavy flow on the counting of other flows due to hash collisions.

2) Multi-threaded Lock-free Parallel Processing: On this basis, we design a multi-threaded lock-free parallel processing architecture [18], which can further improve the processing performance of the software Sketch layer. This architecture effectively addresses the issue of repeated count accumulation caused by locking mechanisms during concurrent insertion of light flows by multiple threads, significantly enhancing the accuracy of light flow statistics; meanwhile, our method proactively evicts heavy flows to the hardware table without compromising the statistical accuracy of heavy flows in any way. The core design is as follows:

We assign an independent network queue to each Arm core of the DPU via Worker threads, while maintaining a single global Sketch instance shared across all cores. By binding each flow to a fixed core through RSS hashing (as shown in the figure, Flow 1 and Flow 2 are bound to different cores), the same flow is always processed by the same core, which fundamentally eliminates cross-core data competition on the shared Sketch without requiring any locking mechanism. The update of Sketch counters is implemented using atomic operations: when hash collisions occur between data packets from different cores (e.g., Core 1 and Core 2 marked with "Hash Collision" in red in the figure), the counter only performs a single atomic increment by 1 (i.e., $3 \rightarrow 4$ marked with "Only +1" in the figure). Taking Flow 2 as an example, its query result is 4 instead of 5, which ensures data consistency under a lock-free premise and effectively avoids count deviation issues. The system adopts a batch packet reception and transmission mechanism to further improve processing efficiency; when the number of data packets on any core (e.g., Core 1) exceeds the preset threshold, the heavy flows on that core are automatically scheduled for priority processing.

Analysis. Consider a $d \times w$ Sketch structure, where L denotes the probability that an update is lost due to concurrent insertions from multiple cores, N represents the total packet count, f_x indicates the true frequency of flow x, and each row employs an independent hash function $h_i(\cdot)$.

Following the Count-Min Sketch analysis, the counter value consists of the true frequency and collision noise introduced by other flows mapped to the same counter. The expected counter value without concurrency is $\mathbb{E}[C_i(x)] = f_x + \frac{N-f_x}{w}$.

where the expected collision noise is bounded by $\mathbb{E}[\Delta_x] = \frac{N-f_x}{w}$. To model concurrent overwrites, define $Z \sim Bernoulli(1-L)$. where $Z = 1$ indicates that an insertion succeeds and $Z = 0$ indicates that the update is lost due to concurrent accesses. We assume each insertion succeeds independently with probability $(1-L)$.

Under this assumption, the counter value becomes $C_i(x) = \sum_{j=1}^{f_x} Z_j + \sum_{y \neq x} \sum_{k=1}^{f_y} Z_{y,k} \cdot \mathbf{1}[h_i(y) = h_i(x)]$.

The first term represents successful updates from flow x. The second term represents successful updates from other flows mapped to the same counter.

The expected counter value becomes $\mathbb{E}_L[C_i(x)] = (1-L)\left(f_x + \frac{N-f_x}{w}\right)$. For analytical simplicity, we approximate the Sketch estimation by analyzing a single row independently. The expected estimation becomes $\mathbb{E}_L[\hat{f}_x] = (1-L)\left(f_x + \frac{N-f_x}{w}\right)$.

The estimation bias introduced by concurrent overwrites is $Bias_x = \mathbb{E}_L[\hat{f}_x - f_x]$, which yields $Bias_x = -Lf_x + (1-L)\frac{N-f_x}{w}$.

Since heavy flows are filtered by the hardware flow table, the Arm-side Sketch mainly stores light flows. For light flows, the underestimation term Lf_x remains limited. Therefore, concurrent overwrites have a relatively small impact on estimation accuracy in the Arm subsystem.

C. Duet Algorithm

1) Algorithm Details: Based on the discussed optimizations, we obtain the insertion operation of the optimized Duet in Algorithm 1. As mentioned earlier, Algorithm 1 is the core of the optimized Duet, processing all incoming network packets and implementing the full-process logic of "hardware table line-speed processing of heavy flows, software Sketch parallel processing of light flows, heavy flow threshold determination, counter targeted reset to eliminate hash collision, and lock-free atomic update to ensure multi-core parallelism".

First, we define the entry point of the DPU packet processing pipeline, compatible with both programmable switches and DPUs. For each incoming packet with flow ID f, we first match against the hardware flow table P (lines 2-4). Upon a successful match, the hardware counter is incremented directly, ensuring error-free counting for heavy flows. For packets missing the hardware flow table (light flows), they are dispatched to DPU Arm cores via RSS hashing (lines 6-7). The minimum value among the CMSketch counters is selected as the approximate flow count min_cnt, mitigating overestimation errors from hash collisions (lines 8-14). If min_cnt has not reached threshold TOP, atomic lock-free updates (sync_fetch_and_add) are applied to the CMSketch counters (lines 17-19). Otherwise, the flow is promoted to the hardware flow table P with its count initialized to TOP, and the corresponding CMSketch entries are cleared (lines 22-24).

The core goal of the query algorithm is to obtain the cumulative packet count of the target flow, following the principle of "hardware table priority, software table fallback". The algorithm achieves near-exact accuracy for heavy flow counting and high-precision approximation for light flow counting.

First, the hardware flow table is queried with high priority. If $idx_p \neq$ NULL, flow f has been offloaded to hardware and its exact count is retrieved directly (lines 1-4).

Then, perform the software CMSketch query (fallback solution). For packets that miss the hardware flow table, the high-precision approximate count of light flows is provided through

Algorithm 1 Duet Insertion Operation

Require: Hash functions $h_1, ..., h_k$ (for CMSketch), rss_h (for RSS hash), primary table P (hardware table), CMSketch counters $CM[1..k][1..m]$, threshold TOP

Ensure: Updated primary table P and CMSketch CM for packet with flow ID f

1: **Pipeline Entry:**
2: **if** $f \in P$ **then**
3: $P.Cnt[idx\_p] \mathrel{+}= 1$
4: **exit pipeline**
5: **end if**
6: $idx\_rss = rss\_h(f)$
7: $min\_cnt = +\infty$
8: **for** $i = 1$ to k **do**
9: $idx\_cm = h_i(f)$
10: $cnt_i = CM[i][idx\_cm]$
11: **if** $cnt_i < min\_cnt$ **then**
12: $min\_cnt = cnt_i$
13: $min\_stage = i$
14: **end if**
15: **end for**
16: **if** $min\_cnt + 1 < TOP$ **then**
17: **for** $i = 1$ to k **do**
18: $idx\_cm = h_i(f)$
19: **sync_fetch_and_add**$(CM[i][idx\_cm],\ 1)$
20: **end for**
21: **else**
22: Insert f into P with $P.Cnt[idx\_p] \leftarrow TOP$
23: Clear CM entries for flow f
24: **end if**
25: **exit pipeline**

CMSketch, and the minimum value is assigned to final_cnt. The optimization measures in Algorithm 1 (including targeted counter reset and lock-free parallelism) reduce the errors induced by heavy flows, thereby improving the accuracy of the Sketch(lines 5-13).

IV. EVALUATION

In this section, we conduct a comprehensive evaluation of the proposed Duet. All experiments are repeated, and the final results are presented as average values. Focusing on the core design goals of Duet, we verify its performance, accuracy, scalability, and resource efficiency through three key experiments:

1) Performance advantages of Duet in *throughput* and *packet processing rate* compared to traditional Sketches (CMSketch [19], CountSketch [20], ElasticSketch [21]);
2) Accuracy advantages of Duet in traffic measurement compared to traditional Sketches;
3) Performance stability and convergence characteristics of Duet under high-speed traffic scenarios.

Algorithm 2 Duet Query Operation

Require: Hash functions $h_1, ..., h_k$ (for CMSketch), primary table P (hardware table), CMSketch counters $CM[1..k][1..m]$

Ensure: Flow ID f, primary table P, CMSketch CM

1: **Query:**
2: **if** $idx\_p \neq$ NULL **then**
3: $final\_cnt = P.Cnt[idx\_p]$
4: **else**
5: $final\_cnt = +\infty$
6: **for** $i = 1$ to k **do**
7: $cnt_i = CM[i][idx\_cm]$
8: **if** $cnt_i < final\_cnt$ **then**
9: $final\_cnt = cnt_i$
10: **end if**
11: **end for**
12: **end if**
13: **return** $final\_cnt$

A. Implementation

A Duet prototype was implemented on the NVIDIA BlueField-2 DPU [22], [23], based on DOCA SDK [24] and DPDK 22.04 [25]. Its core logic, developed in C++, includes 5-tuple extraction, hash calculation, counter update, heavy flow detection, and a precision evaluation module (supporting ARE, AAE, RMSE, WMRE). The hardware-software co-design adopts a "lightweight software counting $\rightarrow$ hardware heavy flow offloading" process, with multi-core parallel processing (independent queues per Arm core) to avoid inter-core data races. User-defined resource constraints are supported, and new instances can be deployed on unallocated SoC cores without affecting existing tasks.

A standard dual-server testbed ensured experimental fairness and reproducibility: two servers are directly connected via a 100 Gbps Mellanox ConnectX-6 card; the DPU server (8 Arm cores, 32 GB DRAM) run Duet. The software environment included Ubuntu 20.04 LTS (Linux 5.15), DOCA 1.5, DPDK 22.04, PktGen 22.07.2 [26], and Python Scapy (for custom workloads). Flow identification used 5-tuple hashes [27], and precision metrics were calculated by comparing DPU measurement data with load generator ground truth via PCIe.

B. Baseline

Four typical Sketches are selected for comparison with Duet. All Sketches are configured with the same memory range (300 $\sim$ 600 KB) to ensure fair comparison. Detailed parameters are as follows:

- Duet (Proposed in this Paper): Uses 3 arrays ($n = 3$), 3 hash functions, 32-bit counters, and an heavy flow threshold set to 0.05% of total packets. It supports dynamic offloading to hardware flow tables.
- CMSketch (Count-Min Sketch): Uses 3 arrays ($n = 3$), 3 hash functions, 32-bit counters, and adopts a minimum-value estimation strategy.

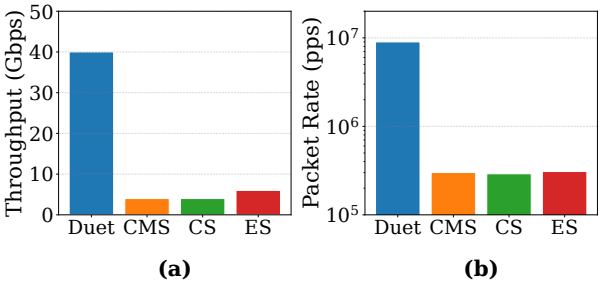

Fig. 5: Throughput Performance of Sketches.

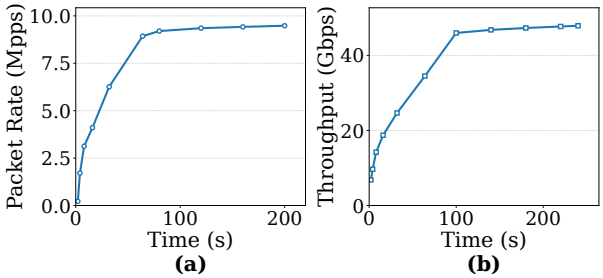

Fig. 6: Convergence Analysis of Sketches.

- CountSketch: Uses 3 arrays ($n = 3$), 3 hash functions, 32-bit counters, and adopts a random increment/decrement update strategy.
- ElasticSketch: Allocates memory to heavyweight and lightweight components at a $1 : 10$ ratio. The heavyweight component uses exact counting, while the lightweight component uses approximate counting.

Both schemes used the same DPU CPU core type to avoid architecture-induced deviations, with comparison dimensions including four precision metrics (ARE, AAE [1], RMSE, WMRE [28], [29]), throughput, packet processing rate, and convergence rate.

C. Result

(EXP1: Performance). To systematically evaluate the bandwidth capability of Duet, we ran pktgen on the host to continuously generate 1024-byte packets, with the traffic distribution following the characteristics of the CAIDA [30] real-world trace dataset. CMS, CS, and ES were all deployed on the Arm subsystem of the DPU. For fairness, each sketch instance was allocated eight Arm cores; each core independently executed one sketch instance without sharing state, and the final measurement results were aggregated. All other configuration parameters were kept identical across schemes.

As shown in Figure 5 (a), the processing bandwidth of CMS, CS, and ES remains below 10 Gbps, whereas Duet achieves approximately 40 Gbps, corresponding to a performance improvement of about $7\times$. Furthermore, under the same experimental configuration, we reduced the packet size to 64 bytes to evaluate the processing capability under small-packet workloads. As illustrated in Figure. 5 (b), the packet processing rates of CMS, CS, and ES are approximately 0.3 Mpps, while Duet achieves approximately 9 Mpps, representing a $29\times$ improvement over ES.

Overall, the results indicate that traditional sketches are generally performance-limited when deployed on the DPU. This limitation mainly stems from two factors. First, the computational capability of the DPU's Arm cores is significantly weaker than that of general-purpose server CPUs. Second, in multi-core environments, shared data structures introduce additional synchronization overhead due to consistency maintenance and locking mechanisms, further constraining system throughput.

ES uses a voting-based eviction mechanism to separate large and light flows, which mitigates hotspot conflicts in the sketch. Duet follows this design and further exploits the DPU hardware flow table. In contrast, other sketch schemes rely only on the Arm subsystem and cannot effectively utilize DPU hardware acceleration, limiting the platform's performance potential.

(EXP2: Accuracy). To systematically evaluate the measurement accuracy of Duet, we ran pktgen on the host to generate high-speed traffic [31] and constructed two types of workloads: (i) real-world network traces and (ii) synthetic traffic. Except for the traffic type, all other experimental configurations keep identical to those used in the performance evaluation to ensure comparability.

First, the results on the real-world dataset are shown in Figure. 7. Across four evaluation metrics, Duet consistently achieves higher measurement accuracy. Due to the limited computational resources of the DPU's Arm subsystem, under high-speed traffic conditions the processing rate of a sketch may fail to keep pace with the packet arrival rate, leading to packet loss or queue congestion and causing the measured accuracy to deviate from its theoretical upper bound. ES achieves the best ARE performance when memory is limited. However, when memory exceeds 500 KB, its accuracy becomes lower than that of Duet. As memory increases, Duet gradually improves its accuracy for large flows. This improvement enhances the acceleration effect of DPU hardware on traffic measurement, leading to higher measurement accuracy. In addition, 600 KB is a small amount of memory for a DPU. Therefore, sacrificing a small amount of memory space to obtain better accuracy is an acceptable trade-off.

Second, the results on the synthetic dataset are shown in Figure. 8. In this workload, heavy flows packet account for approximately 80% of the total traffic, while their absolute number remains relatively small. moreover, under this traffic distribution, the relative impact of packet loss and congestion on per-flow measurement results is reduced, leading to a noticeable improvement in accuracy. Nevertheless, even in this more favorable setting, Duet retains its performance advantage.

Overall, Duet demonstrates stable and superior accuracy across different traffic distributions. This advantage stems primarily from two aspects. First, Duet offloads heavy flows to the DPU hardware flow table for forwarding and counting; Second, Duet enhances the sketch running on the Arm

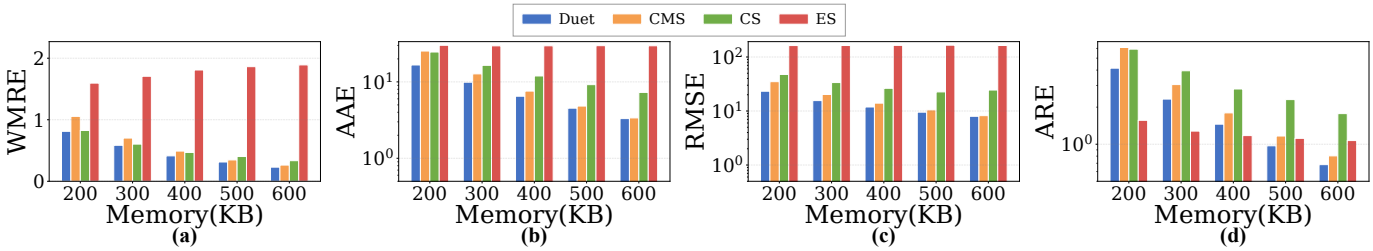

Fig. 7: Accuracy Metrics Comparison of Sketches for Real-world Workload (Memory: 300 $\sim$ 600 KB).

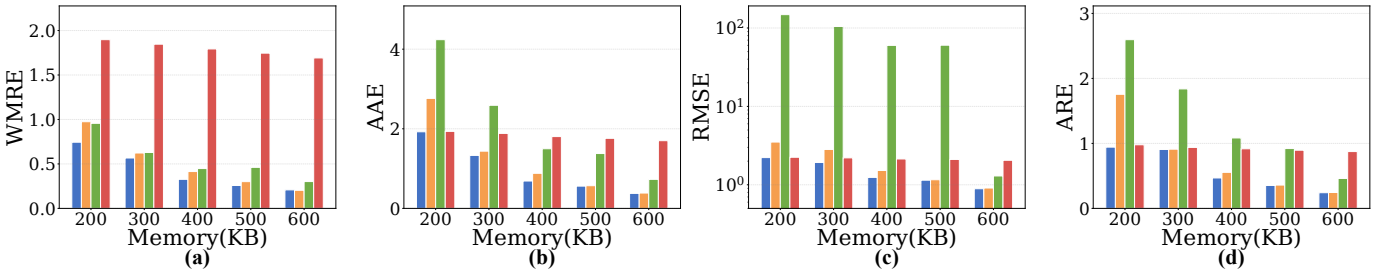

Fig. 8: Accuracy Metrics Comparison of Sketches for Synthetic Workload (Memory: 300 $\sim$ 600 KB).

subsystem through reset optimization and multi-core lock-free techniques, effectively reducing collision-induced errors among light flows and improving the measurement accuracy of the software sketch. The synergy of these two mechanisms enables Duet to achieve higher measurement accuracy while sustaining high throughput.

(EXP3: Convergence). We further evaluate the convergence time of Duet in reaching its maximum processing rate. In the experiments, high-intensity traffic is continuously generated at the sender using pktgen, while the system throughput and packet rate are recorded in real time at the receiver. To separately assess throughput and packet processing capability, 1024-byte packets are transmitted in the throughput tests, whereas 64-byte packets are used in the packet-rate tests. All other experimental settings are kept identical to those in the performance evaluation phase.

The experimental results are presented in Figure 6. For the packet-rate test using 64-byte small packets, the system packet processing rate converges to its maximum stable value within approximately 60 seconds. For the throughput test using 1024-byte large packets, the system throughput converges within approximately 100 seconds. Overall, Duet is able to reach stable peak performance on a minute-level timescale under both workloads.

In contemporary cloud computing environments, data centers typically sustain millions of concurrent flows, each with relatively low per-flow utilization. The overall traffic pattern tends to remain stable and evolves gradually on an hourly timescale [32]. In this context, the minute-level convergence characteristic of Duet is sufficient for practical deployment scenarios. Its performance sensitivity aligns well with the temporal dynamics of typical cloud traffic, thereby ensuring both the stability and practicality of measurement results.

V. Related Work

Sketch Optimization. OctoSketch addresses multi-core scalability on the host by parallelizing sketch updates across CPU cores, significantly improving measurement throughput [18]. However, it is primarily designed for host-side deployment and continues to consume substantial CPU resources, making it difficult to avoid interference with application performance. [1], [21], [33] implement a general-purpose measurement framework on the host by leveraging techniques such as randomized sampling and voting-based eviction. However, it does not incorporate hardware-specific optimization mechanisms tailored to particular architectural features.

Hardware Assistance. MPU deploys multiple sketch instances on DPU platforms to reduce Arm-side resource contention while preserving measurement accuracy [14]. Nevertheless, MPU executes all measurement logic entirely on the Arm subsystem and does not leverage the high-performance data path provided by the DPU hardware forwarding tables. This design limits its scalability and performance potential under high traffic loads. [34], [35] primarily focus on performing measurements on programmable switches. Compared with DPUs, programmable switches are more expensive, offer limited programming flexibility, and are less portable across heterogeneous deployment environments.

DPU offloading. [36] prior studies offload stateful network functions to DPUs and adopt hierarchical processing strategies based on connection duration and packet volume. [37] Other works offload storage functionalities to DPUs and perform tiered management according to traffic characteristics and application types, thereby optimizing cache behavior across different levels. However, these approaches are not specifically designed to optimize Sketch-based measurement tasks and do not provide targeted enhancements for such scenarios.

Duet is orthogonal to the above studies. Software optimization techniques are equally applicable to the Arm subsystem of a DPU, and Duet likewise adopts a hardware–software co-design approach to further enhance system performance.

VI. Conclusion

This paper presents Duet, a layered flow-aware measurement framework via coordinated DPU offloading. Duet separates heavy and light flows through hardware–software collaboration, enabling heavy hitters to be accurately tracked by hardware flow tables while retaining lightweight sketch-based measurement for the remaining traffic on the Arm subsystem. By reducing unnecessary flow-table interactions and minimizing contention on DPU cores, Duet effectively improves resource utilization and measurement efficiency. Extensive experiments demonstrate that Duet improves throughput by a factor of approximately $7.1\times$, packet rate by a factor of approximately $28.9\times$ and enhances measurement accuracy by approximately 50.1% compared to traditional sketch-only approaches.

VII. Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant Nos. 62572105 and U22B2005, as well as the LiaoNing Revitalization Talents Program under Grant No. XLYC2403086.

References

[1] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, and V. Sekar, "Nitrosketch: robust and general sketch-based monitoring in software switches," *ACM*, 2019.

[2] 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 Transactions on Parallel and Distributed Systems*, vol. 34, no. 11, pp. 2876–2894, 2023.

[3] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 113–126.

[4] L. Tang, Q. Huang, and P. P. Lee, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2026–2034.

[5] M. Yu, L. Jose, and R. Miao, "Software {Defined}{Traffic} measurement with {OpenSketch}," in *10th USENIX symposium on networked systems design and implementation (NSDI 13)*, 2013, pp. 29–42.

[6] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[7] S. Miano, X. Chen, R. B. Basat, and G. Antichi, "Fast in-kernel traffic sketching in ebpf," *ACM SIGCOMM Computer Communication Review*, vol. 53, no. 1, pp. 3–13, 2023.

[8] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "Flowatcher-dpdk: Lightweight line-rate flow-level monitoring in software," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1143–1156, 2019.

[9] X. Wu, P. Li, Y. Ran, and Y. Luo, "Network measurement for 100 gbe network links using multicore processors," *Future Generation Computer Systems*, vol. 79, pp. 180–189, 2018.

[10] J. Ye, L. Li, W. Zhang, G. Chen, Y. Shan, Y. Li, W. Li, and J. Huang, "Ua-sketch: an accurate approach to detect heavy flow based on uninterrupted arrival," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[11] F. Li, Q. Chen, J. Shen, X. Wang, and J. Cao, "Performance characteristics and guidelines of offloading middleboxes onto bluefield-2 dpu," *IEEE Transactions on Computers*, vol. 74, no. 2, pp. 609–622, 2024.

[12] Z. Huang, Y. Tan, Y. Zhu, H. Tan, and K. Li, "Dynamic dpu offloading and computational resource management in heterogeneous systems," *IEEE Transactions on Computers*, 2025.

[13] S. Lee, M. You, S. Kim, and T. Park, "Comprehensive performance analysis of security applications on the bluefield-3 smartnic," *Computer Networks*, p. 111830, 2025.

[14] X. Chen, X. Sun, W. Zhang, X. Yao, Z. Wang, H. Liu, Q. Huang, G. Pan, X. Liu, H. Zhou, and C. Wu, "Accelerating sketch-based end-host traffic measurement with automatic DPU offloading," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 171–180.

[15] "Nvidia BlueField SmartNIC," https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet.

[16] "Amd Pensando," https://www.amd.com/products/accelerators/pensando.

[17] "Netronome Agilio SmartNICs," https://www.netronome.com/products/smartnic.

[18] Y. Zhang, P. Chen, and Z. Liu, "OctoSketch: Enabling real-time, continuous network monitoring over multiple cores," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 2024)*. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/zhang-yinda

[19] 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, 2004.

[20] M. C. A, K. C. B, and F. C. C, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2002.

[21] T. Yang, J. Jiang, P. Liu, Q. Huang, and S. Uhlig, "Elastic sketch: adaptive and fast network-wide measurements," *ACM*, 2018.

[22] "Nvidia bluefield dpu," https://tinyurl.com/3cady99f.

[23] F. Li, Q. Chen, J. Shen, X. Wang, and J. Cao, "Performance characteristics and guidelines of offloading middleboxes onto bluefield-2 dpu," *IEEE Transactions on Computers*, 2024.

[24] "Doca flow programming guide," https://docs.nvidia.com/doca/sdk/doca-programming-guide/.

[25] "Dpdk," https://doc.dpdk.org/guides-20.11/.

[26] "Pktgen," https://pktgen-dpdk.readthedocs.io/.

[27] A. Goyal, H. Daumé, and G. Cormode, "Sketch algorithms for estimating point queries in nlp," in *Empirical Methods in Natural Language Processing*, 2012.

[28] Q. Huang, X. Jin, P. P. C. Lee, R. Li, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Acm Sigcomm Conference*, 2017.

[29] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," *ACM*, 2004.

[30] "Caida traces," http://www.caida.org/data/overview/.

[31] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," 2016.

[32] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao *et al.*, "Alibaba hpn: A data center network for large language model training," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 691–706.

[33] 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 *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 207–222.

[34] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "SketchLib: Enabling efficient sketch-based monitoring on programmable switches," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 743–759.

[35] ——, "Sketchovsky: Enabling ensembles of sketches on programmable switches," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1273–1292.

[36] X. Wang, D. Li, Z. Wang, L. Jiang, S. Wen, D. Kang, E. Arslan, P. He, X. Qian, B. Niu, J. Pi, X. Ding, K. Lin, and H. Luo, "Byte vswitch: A high-performance virtual switch for cloud networking," ser. EuroSys '25, 2025, p. 1417–1432.

[37] A. Kashyap, Y. Li, and X. Lu, "Dpu-kv: On the benefits of dpu offloading for in-memory key-value stores at the edge," in *Proceedings of the 34th International Symposium on High-Performance Parallel and Distributed Computing*. Association for Computing Machinery, 2025.