HARMONIA: A Swift and Accurate Approximate Data Structure for Real-Time Heavy Flow Detection in High-Speed Networks

Weihe Li[1], Tianyue Chu[2], Christos-Savvas Bouganis[3], and Paul Patras[1]
[1] University of Edinburgh, Edinburgh EH8 9AB, United Kingdom
[2] Telefónica Research, Barcelona 08019, Spain
[3] Imperial College London, London SW7 2AZ, United Kingdom
weihe.li@ed.ac.uk, tianyue.chu@telefonica.com,
christos-savvas.bouganis@imperial.ac.uk, paul.patras@ed.ac.uk

Abstract. In high-speed networks, real-time monitoring of heavy flows is vital for reliable communication and rapid anomaly detection (e.g., DDoS). This task is challenging due to limited fast memory and the inefficiency of existing sketches, which struggle with skewed traffic and hardware constraints. We present HARMONIA, a sketch for efficient and accurate heavy flow detection. Unlike prior methods that use multi-dimensional features at high memory cost, HARMONIA leverages traffic skewness: once a flow's frequency exceeds a threshold, it is deemed heavy and protected from eviction during hash collisions. Implemented on CPU, programmable switches, and FPGA, and evaluated on real-world traces, HARMONIA improves detection accuracy by up to 27.83% while requiring at most 18% and 10% of available resources on switches and FPGAs.

Keywords: approximate data structures · sketch · high-speed data streams · heavy hitter detection · anomaly detection · programmable switch · FPGA.

1 Introduction

The exponential growth of network traffic from applications like video streaming, high-frequency trading, and cloud computing has created high-volume flows across modern networks. Real-time monitoring is crucial to identify heavy flows or "heavy hitters," which may hide anomalies such as (D)DoS attacks or impact quality-of-service (QoS). Detecting heavy flows in real time is challenging due to rapidly increasing traffic and limited high-speed memory, such as the typical 64KB L1 cache [11–13, 16]. Tracking every flow is impractical in such environments. To address this, approximate data structures, or sketches, often implemented as multi-row hash tables, are used to detect heavy flows efficiently while conserving memory [7, 10, 17].

Limitations of Existing Methods: Despite advances in heavy flow detection, current solutions face key limitations. (i) Methods like Stable-Sketch [13] and Tight-Sketch [12] combine flow frequency with stability or continuity to protect heavy flows. While accurate when few heavy flows exist, their extra metadata reduces available buckets, hurting memory efficiency as heavy flows increase. (ii) Sketches such as MV-Sketch [16] and Count-Min Sketch [7] require large memory to be accurate, and their replacement strategies often evict heavy flows under tight memory and frequent collisions. (iii) Programmable switches, widely used in data centers and IoT networks [2], have limited stages (e.g., 12)

and only support basic arithmetic, making sketches with complex updates (e.g., counter merging [3]) impractical.

Contributions: We propose HARMONIA, a new sketch for accurate heavy flow detection. When hash collisions occur across all rows, HARMONIA employs a probabilistic replacement strategy, leveraging the skewed nature of network traffic where only a small fraction of flows are heavy [8]. Our analysis shows that once a flow's frequency exceeds a threshold, it is highly likely to be heavy; thus, we disable replacements to prevent eviction by non-heavy flows and reduce estimation errors. This design improves detection accuracy under limited memory while avoiding complex operations such as matrix multiplication or counter merging, making it practical for programmable switch deployment.

We implement HARMONIA on CPU (C++), programmable switches (P4 [5]), and FPGA (Verilog). Evaluations on real-world traces [1] show that HARMONIA improves detection accuracy by up to 15.4% and reduces estimation error by 32.27% compared to Stable-Sketch [13]. In hardware, HARMONIA requires modest resources, using at most 18% of switch and 10% of FPGA capacity.

2 Background and Related Work

2.1 Sketches

In memory-constrained scenarios, recording information for every flow is impractical. To address this challenge, hashing-based sketches are commonly used for high-speed network measurements [16]. A well-known instance is the Count-Min Sketch [7], which organizes data in a table with r rows, each containing multiple buckets paired with a unique hash function. Each bucket has a counter to track flow occurrences. When a packet arrives, its flow key (e.g., source IP address) is hashed into r buckets across the rows, incrementing the corresponding counters by 1. For querying, a flow hashes into buckets in each row, and the smallest counter value among them serves as its estimated frequency.

2.2 Heavy Hitter Detection

A heavy hitter is a flow e in a network traffic set D whose frequency $f(e)$ meets or exceeds a specified threshold $\phi \cdot |D|$, where ϕ is a predefined fraction (e.g., $\phi = 0.001$) and $|D|$ is the total number of packets in D. Formally, e is a heavy hitter if $f(e) \geq \phi \cdot |D|$.

For detecting heavy hitters, various sketch-based methods have been introduced [14]. Count-Min Heap [7] incorporates an additional heap to track flows exceeding the threshold, but the extra data structure reduces memory efficiency, making it less suitable for memory-constrained environments such as programmable switches. MV-Sketch [16] uses a majority vote algorithm to retain heavy flows but involves a larger memory footprint to achieve high detection accuracy. UA-Sketch [19] utilizes a probabilistic eviction strategy that considers flow arrival patterns to evict non-heavy flows. However, its design is based on the assumption that all heavy flows arrive continuously without interruptions, resulting in reduced detection accuracy when this assumption does not hold. Tight-Sketch [12] and Stable-Sketch [13] introduce additional features to enhance the protection of potential heavy flows. However, the additional features result in memory inefficiency and deployment complexity. Other methods, such

as SALSA [3], utilize flexible counters that adjust based on traffic distribution. However, these methods rely on complex hierarchical data structures and computationally intensive operations, which result in lower processing speeds and complicate deployment on practical hardware.

2.3 Hardware Platforms

Programmable Switches offer high-speed packet processing with flexibility, enabling tailored traffic monitoring at the network edge or in data centers. However, they have limitations: (i) limited per-stage memory (e.g., 1.4MB shared between forwarding and monitoring) [15]; (ii) support only basic arithmetic; (iii) a fixed number of physical stages (e.g., 12); and (iv) no cross-stage memory access. Sketch update algorithms must follow a unidirectional workflow [20].

FPGAs are reconfigurable chips made of programmable logic blocks, I/O modules, and SRAM-based interconnects, programmed via hardware description languages like Verilog [6]. They provide high throughput for network processing and are popular for sketch implementations [17], but have constraints such as limited arithmetic support (e.g., division), requiring careful sketch design.

3 HARMONIA

3.1 Design Philosophy

To maintain high accuracy under limited memory budgets, the sketch design needs to effectively adapt to the skewed traffic distributions commonly observed in practice [12]. This adaptation helps prevent heavy flows from being mistakenly evicted by large volumes of non-heavy flows when hash collisions are frequent. Overall, the design of HARMONIA is guided by the following two principles.

Principle 1: Unlike update strategies such as the majority vote algorithm [16], which demand substantial memory for accuracy under skewed traffic, we adopt a probabilistic replacement strategy where eviction likelihood decreases with flow frequency. This simple approach also eases hardware implementation on programmable switches and FPGAs [20]. Our analysis of CAIDA [1] and ToN_IoT [4] traces (Table 1) shows that over 95% of flows have a frequency of 50 or less, while very few exceed 300 (e.g., 1.22% in CAIDA 2018 and 0.05% in IoT_Scanning). Thus, under limited memory and frequent collisions, heavy flows are prone to eviction by numerous small flows–a problem HARMONIA addresses.

Table 1: Flow Frequency Distribution in Practical Traces.

Frequency	CAIDA2015	CAIDA2018	IoT_DDoS	IoT_Scanning
[1, 50]	95.63%	96.37%	98.79%	99.57%
(50, 100]	1.50%	1.30%	0.37%	0.25%
(100, 300]	1.44%	1.11%	0.51%	0.12%
>300	1.43%	1.22%	0.33%	0.05%

Principle 2: Unlike state-of-the-art approaches such as Stable-Sketch [13] and Tight-Sketch [12], which leverage dual-feature mechanisms to protect heavy flows and consequently require additional memory, resulting in fewer buckets and increased hash collisions that elevate the risk of erroneously evicting heavy flows, our method protects heavy flows using only a single parameter, namely the actual frequency tracked in the value counter, ensuring efficient memory usage.

3.2 Data Structure

Existing sketch data structures can be broadly categorized into two types: flat [7, 16, 19] and hierarchical [14]. While hierarchical structures are generally more memory efficient due to their carefully designed counters of varying lengths, their complexity, such as bit-level counter management, renders them impractical for deployment on programmable switches. Therefore, we adopt a flat-based structure for its simplicity and ease of hardware implementation.

Fig.1 illustrates the data structure of HARMONIA, which consists of r rows, each associated with a pairwise-independent hash function h_i $(1 \leq i \leq r)$. Each row contains b columns (*buckets*), where each bucket $B(i, j)$ $(1 \leq j \leq b)$ has two fields: the flow key K and its frequency counter V.

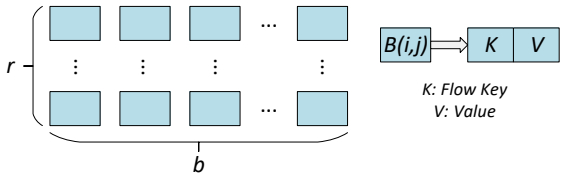


Fig. 1: HARMONIA's data structure.

3.3 Main Operations

Update: When an incoming packet belonging to a flow arrives, the update falls into two possible stages, as listed in Algorithm 1 (see our technical report [9]).

HARMONIA begins by hashing the key of the incoming flow into a bucket in the first row using the hash function h_1. If the corresponding bucket $B(1, h_1(e_m.key))$ is empty (i.e., $key = \emptyset$ and $count = 0$), the flow key is inserted into the bucket, and the count is initialized to 1, terminating the update process. If the bucket already contains the same flow key, the count is incremented by 1, and the process ends. Otherwise, if the bucket contains a different flow key, the algorithm proceeds to the next row. This process is repeated row by row using the corresponding hash functions $h_2, h_3, ..., h_r$. In each row, if the bucket is empty or contains the same flow key, the algorithm updates the bucket and terminates. If no matching or empty bucket is found after traversing all rows, HARMONIA identifies the bucket with the smallest count across all rows and proceeds to the next stage.

In the second stage, HARMONIA decides whether to replace the bucket with the smallest count. If the smallest count $count \geq \Omega$, where Ω is a predefined threshold (*analyzed in detail in Section 6.2*), the bucket retains its current key, and the incoming flow is discarded. Otherwise, HARMONIA calculates the replacement probability. A random number k is generated within the range 0 and 1, and the replacement threshold is computed as $\frac{1}{count+1}$. If the replacement succeeds, the bucket's key is updated with the new flow key, and the count is incremented by 1. If the replacement fails, the incoming flow is discarded, and the process terminates).

Query: To retrieve the heavy flows, HARMONIA simply scans all the buckets. If the frequency value of a tracked flow exceeds the predefined threshold for heavy hitters, the flow is identified as a heavy hitter.

4 Formal Analysis

In this section, we prove that HARMONIA achieves (ϵ, δ)-counting [13], thus demonstrating its low error rate in estimating heavy flows.

Theorem 1. *In Algorithm 1, given a small positive number ϵ, for any heavy flow e_m with frequency $f(e_m)$, the estimated frequency $\hat{f}(e_m)$ satisfies:*

$$\Pr\left(\hat{f}(e_m) \leq f(e_m) - \epsilon|D|\right) \leq \delta$$

where $\delta = \frac{1}{\epsilon b}\left[1 - \frac{\Omega}{2|D|^2}\right]$. Here, b represents the number of buckets in each row, $|D|$ is the total number of packets, and Ω is the predefined threshold.

Proof. A comprehensive proof is available in our technical report [9].

5 Implementation on Hardware Platforms

The implementation details for the programmable switch and FPGA are provided in our technical report [9].

6 Evaluation

6.1 Setup

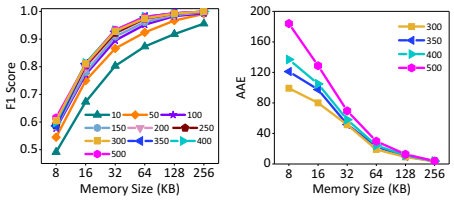
Platform (1) CPU Platform. We implement HARMONIA in C++ and evaluate it on a laptop with an Intel® Core™ i5-1135G7 @ 2.40GHz, running Ubuntu 20.04 LTS. **(2) Programmable Switch.** We prototype HARMONIA in P4 [5] and compile it with Intel® P4 Studio for deployment on Intel® Tofino™ ASICs [2]. **(3) FPGA.** We implement HARMONIA in Verilog, compiled with Vivado 2024.2, targeting the UltraScale+ XCU250-L2FIGD2104E FPGA.

Datasets We evaluate HARMONIA using real-world CAIDA traces [1]. Two were introduced in Section 3.1, and one from 2019 contains 29.5M packets across 1.53M flows. The heavy hitter threshold $\phi = 10^{-4}$ is used, with source IP as the flow key.

Benchmarks We compare HARMONIA with four state-of-the-art methods: MV-Sketch [16], UA-Sketch [19], Tight-Sketch [12], and Stable-Sketch [13], all configured with two rows. Other methods (e.g., CocoSketch [20], Elastic [18]) are omitted since Stable-Sketch already outperforms them.

Memory Allocations We test all methods under memory sizes of 8KB, 16KB, 32KB, 64KB, 128KB, and 256KB [13], with fixed bucket sizes. In HARMONIA, keys and counters each occupy 4 bytes; e.g., 8KB yields 1024 buckets $\left(\frac{8\times1024}{8}\right)$. Smaller memory also reserves resources for other applications and speeds up heavy hitter queries.

Metrics Detection accuracy is evaluated using: (1) F1 Score: $\frac{2\times\text{recall}\times\text{precision}}{\text{recall}+\text{precision}}$, where recall is the fraction of ground-truth heavy hitters detected, and precision the fraction of correct detections; (2) Average Absolute Error (AAE): the mean absolute difference between true and estimated frequencies; (3) Update Speed: throughput in millions of packets per second (Mpps).



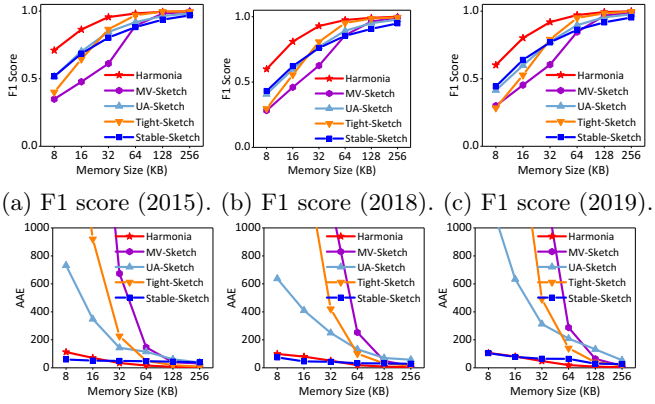
(a) F1 score. (b) AAE.

Fig. 2: Detection accuracy (F1 score) for varying parameter Ω.

6.2 Parameter Setting

Recall that HARMONIA uses a threshold parameter Ω to enhance the protection of heavy hitters. To determine the value, following the approach of prior works [11, 14, 18], we conduct empirical evaluations on the CPU platform to determine suitable parameter configurations for our method. The CAIDA 2018 trace is used as the test trace, with similar trends observed across other traces.

As shown in Fig. 2(a), the F1 score rises sharply as Ω increases from 10 to 50, since small Ω allows non-heavy flows to occupy buckets, reducing accuracy. Gains continue up to $\Omega = 300$, beyond which Fig. 2(b) shows higher estimation errors due to heavy flows being evicted by non-heavy ones. We therefore choose $\Omega = 300$ as the proper threshold for subsequent evaluations.



(a) F1 score (2015). (b) F1 score (2018). (c) F1 score (2019).

(d) AAE (2015). (e) AAE (2018). (f) AAE (2019).

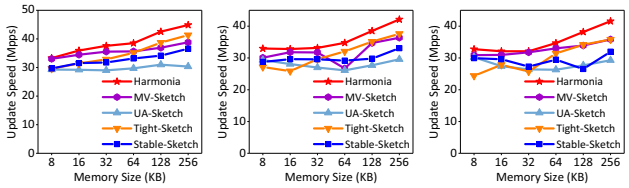
Fig. 3: Detection accuracy of various methods on the CPU platform.

6.3 CPU Platform

Detection Accuracy: Figs. 3(a)-(c) show F1 scores across different memory configurations and network traces. HARMONIA consistently outperforms baseline methods, improving F1 by 11.72%-27.83%, 15.4%-26.82%, and 14.49%-27.02% on the 2015, 2018, and 2019 traces, respectively. While other metrics like recall and precision are omitted for brevity, HARMONIA also leads in these measures. Figs. 3(d)-(f) show that HARMONIA achieves the lowest AAE, reducing estimation error by 37.27% on the CAIDA 2019 trace compared to Stable-Sketch.

Update Speed: Fig. 4 shows the update speeds of various approaches. Observe that HARMONIA achieves the highest update speed thanks to its simple and

effective update strategy that avoids extra processing of additional features. On average, it improves update speed by 8.53%-30.35%, 12.01%-27.74%, and 7.66%-26.37% compared to the baselines over the CAIDA 2015, 2018, and 2019 traces.



(a) Speed (2015). (b) Speed (2018). (c) Speed (2019).

Fig. 4: Update speed of different methods for heavy hitter detection.

6.4 Resource Utilization on Hardware Platforms

Programmable Switch Table 2 shows HARMONIA's resource usage on the programmable switch. The Logical Table ID consumes the most resources (17.19%), followed by Gateways (15.62%) for conditional logic. Meter ALUs and the Exact Match Search Bus each use 10.42%, supporting efficient packet processing and accurate flow recording. SRAM usage is minimal (1.56%), with Map RAM (1.74%) and Stash (0.52%) also lightly used. Overall, HARMONIA maintains a small resource footprint, leaving ample capacity for additional applications.

Table 2: Resource Usage and Percentage Details on the Programmable Switch.

Resource	Exact Match Input xbar	Hash Bit	Hash Dist Unit	Gateway	SRAM	Map RAM
Usage	86	103	6	30	15	10
Percentage (%)	5.60	2.06	8.33	15.62	1.56	1.74
Resource	VLIW Instr	Meter ALU	Stash	Exact Match Search Bus	Logical Table ID	
Usage	19	5	1	20	33	
Percentage (%)	4.95	10.42	0.52	10.42	17.19	

FPGA Table 3 lists the overhead of HARMONIA on the FPGA platform. It shows that the design requires a relatively small fraction of the available lookup tables, LUTRAM and registers, while the usage of BRAM (7.14%) and I/O (9.91%) is well within the limits of the device, leaving enough resources for other applications. Also, the throughput of HARMONIA is 82.2 Mpps, which demonstrates a high processing speed. These results validate that HARMONIA can be effectively deployed on the FPGA platform.

Table 3: FPGA Resource Utilization.

Metric	Look-Up Tables (LUTs)	LUTRAM	Registers	BRAM Blocks	I/O Pins
Usage	1,075	33	685	192	67
Percentage(%)	0.06	0.004	0.02	7.14	9.91

7 Conclusion

Detecting heavy flows is vital in scenarios such as mitigating malicious traffic in IoT and data center networks. We present HARMONIA, a sketch that achieves accurate detection under tight memory constraints by combining a probabilistic replacement strategy with a threshold mechanism to prevent heavy flows from being evicted by non-heavy ones. Its simple, resource-efficient design makes it practical for constrained hardware while maintaining high detection accuracy.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council under Grant EP/V038699/1 and the EU Horizon Europe TaRDIS project (Grant Agreement 101093006). Weihe Li was partially supported by Cisco through the Cisco University Research Program Fund (Grant No. 2019-197006).

References

1. The caida anonymized internet traces, http://www.caida.org/data/overview/
2. Agrawal, A., Kim, C.: Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In: Proc. IEEE Hot Chips Symp. pp. 1–32. IEEE Computer Society (2020)
3. Basat, R.B., et al.: Salsa: self-adjusting lean streaming analytics. In: Proc. IEEE ICDE. pp. 864–875. IEEE (2021)
4. Booij, T.M., et al.: Ton_iot: The role of heterogeneity and the need for standardization of features and attack types in iot network intrusion data sets. IEEE Internet Things J. **9**(1), 485–496 (2021)
5. Bosshart, P., et al.: P4: Programming protocol-independent packet processors. ACM SIGCOMM Comput. Commun. Rev. **44**(3), 87–95 (2014)
6. Boutros, A., Betz, V.: Fpga architecture: Principles and progression. IEEE Circuits Syst. Mag. **21**(2), 4–29 (2021)
7. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
8. Huang, J., et al.: Chainsketch: An efficient and accurate sketch for heavy flow detection. IEEE/ACM Trans. Netw. **31**(2), 738–753 (2022)
9. Li, W., et al.: Harmonia technical report. https://github.com/WeiheLi/Harmonia, accessed: 2025-08-18
10. Li, W., et al.: Pontus: A memory-efficient and high-accuracy approach for persistence-based item lookup in high-velocity data streams. In: Proc. ACM Web Conf. pp. 1783–1794 (2025)
11. Li, W., Patras, P.: P-sketch: A fast and accurate sketch for persistent item lookup. IEEE/ACM Trans. Netw. **32**(2), 987–1002 (2023)
12. Li, W., Patras, P.: Tight-sketch: A high-performance sketch for heavy item-oriented data stream mining with limited memory size. In: Proc. ACM CIKM. pp. 1328–1337 (2023)
13. Li, W., Patras, P.: Stable-sketch: A versatile sketch for accurate, fast, web-scale data stream processing. In: Proc. ACM Web Conf. pp. 4227–4238 (2024)
14. Li, Y., et al.: Pyramid family: Generic frameworks for accurate and fast flow size measurement. IEEE/ACM Trans. Netw. **30**(2), 586–600 (2021)
15. Sivaraman, V., et al.: Heavy-hitter detection entirely in the data plane. In: Proc. SoS Research. pp. 164–176 (2017)
16. Tang, L., et al.: A fast and compact invertible sketch for network-wide heavy flow detection. IEEE/ACM Trans. Netw. **28**(5), 2350–2363 (2020)
17. Tang, M., et al.: Towards memory-efficient streaming processing with counter-cascading sketching on fpga. In: Proc. ACM/IEEE DAC. pp. 1–6. IEEE (2020)
18. Yang, T., et al.: Elastic sketch: Adaptive and fast network-wide measurements. In: Proc. ACM SIGCOMM Conf. pp. 561–575 (2018)
19. Ye, J., et al.: Ua-sketch: an accurate approach to detect heavy flow based on uninterrupted arrival. In: Proc. ICPP. pp. 1–11 (2022)
20. Zhang, Y., et al.: Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In: Proc. ACM SIGCOMM Conf. pp. 207–222 (2021)