SCIENCE CHINA
Information Sciences

• LETTER •

August 2026, Vol. 69, Iss. 8, 189302:1–189302:2
https://doi.org/10.1007/s11432-025-4874-x

Persistent finder: fast and accurate detection of persistent flows in high-speed networks

Lu CAO[1,2], Weiqiang XIAO[1], Lin YAO[3] & Weizhe ZHANG[1,2*]

[1]*School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China*
[2]*Department of New Networks, Peng Cheng Laboratory, Shenzhen 518066, China*
[3]*School of Software, Dalian University of Technology, Dalian 116024, China*

Received 28 July 2025/Revised 21 November 2025/Accepted 19 March 2026/Published online 27 July 2026

Citation Cao L, Xiao W Q, Yao L, et al. Persistent finder: fast and accurate detection of persistent flows in high-speed networks. Sci China Inf Sci, 2026, 69(8): 189302, https://doi.org/10.1007/s11432-025-4874-x

In recent years, rapid increases in network traffic have introduced significant challenges for efficient updating and maintenance of fast query responses for persistent flows. As a result, identifying sustained patterns in network traffic across multiple time windows is more challenging than basic frequency counting, especially in large communication networks. The associated detection process necessitates repeated memory access for each arriving packet to track flow persistence, potentially leading to performance bottlenecks.

Existing methods for persistent-flow detection are mainly sketch-based, such as on-off sketch (OO) [1], TightSketch (TS) [2], and more recent designs [3–5]. While these structures improve accuracy, they typically rely on multiple hash evaluations and complex replacement or eviction policies, which lead to high update overhead, increased memory pressure, or degraded query performance under dynamic workloads. To address these issues, we propose the persistent finder (PF), a fast and accurate framework for detecting persistent flows in high-speed networks, such as Internet backbones and data centers.

Problem statement. Let S be a packet stream and let time be partitioned into fixed-length windows $W_1, W_2, \ldots, W_n$ (e.g., 1 s each). For a flow x, its *persistence* is defined as the number of windows in which x appears at least once:

$$P(x) = \big|\{\, W_i \mid x \text{ appears in } W_i,\ 1 \leqslant i \leqslant n\}\big|. \tag{1}$$

Given a user-defined threshold k, the goal is to identify, in real time and under tight memory constraints, all flows with $P(x) \geqslant k$. Unlike frequency, which counts the total number of packets, persistence captures how widely a flow is sustained across windows, making it well-suited for detecting long-lived behaviors such as persistent attacks or anomalies.

Framework Overview. PF uses a flat multi-array sketch tailored for persistent-flow detection. Unlike hierarchical sketches that require multiple independent hash functions, PF hashes each packet e once into a 64-bit value $h(e)$ and splits it into d sub-indices to select one cell per array, deriving bucket and in-bucket indices to address d parallel arrays, as shown in Figure 1(a). Each array has M buckets; each bucket holds N compact cells. Every cell maintains a small persistence counter and a one-bit flag indicating whether the flow has already been counted in the current window. At the beginning of each window, all flags are reset to `On`, while counters accumulate the number of windows in which a flow is observed. Under a unified rule, PF skips updates if all mapped cells have `Off` flags; otherwise, it increments minimum counters of mapped `On` cells, resolves ties, and sets all d mapped flags to `Off` to ensure one contribution per flow per window. This flat organization yields regular memory accesses and constant-time updates, making PF friendly to high-speed software data planes. PF's key design choice is forgoing in-sketch key storage to maximize memory efficiency, with flow identifiers retrievable via independent query or post-processing as needed.

PF's efficiency comes from two core ideas. *Speed optimization via one-hashing*: instead of computing d independent hash functions per packet, PF evaluates a single 64-bit hash and partitions it into d disjoint bit ranges, each used as an index to one array. This preserves the randomness of a multi-hash design while avoiding redundant hash evaluations, thereby reducing per-packet hash cost and CPU utilization compared with conventional d-hash sketches, and keeping collision probability under control.

Accuracy optimization via ostrich policy: PF aims to count each flow at most once per window. For a packet mapped to d cells, PF checks the per-cell flags and increments only the mapped cell(s) with the smallest counter value among those whose flag is still `On`, then flips the flags of all d mapped cells to `Off`. This fixed read-compare-write sequence is compatible with typical programmable-switch pipelines and does not require packet recirculation. Subsequent packets of the same flow in the same window see all flags as `Off` and perform no further updates. By ignoring repeated arrivals, PF suppresses redundant memory writes, lowers the effective number of distinct updates per window, and reduces both update latency and overestimation error.

The operational example in Figure 1(a) illustrates this process. Packets e_1, e_2, and e_3 are mapped to three arrays; for each, PF updates only the highlighted cells and then sets the flags of all its mapped cells to `Off`. When packets e_4 and e_5 arrive, their mapped cells already have flags `Off`, so PF performs no update; thus, no counters are modified, and these cells are labeled "Not changed" in the figure, demonstrating how PF achieves near-constant update

* Corresponding author (email: wzzhang@hit.edu.cn)

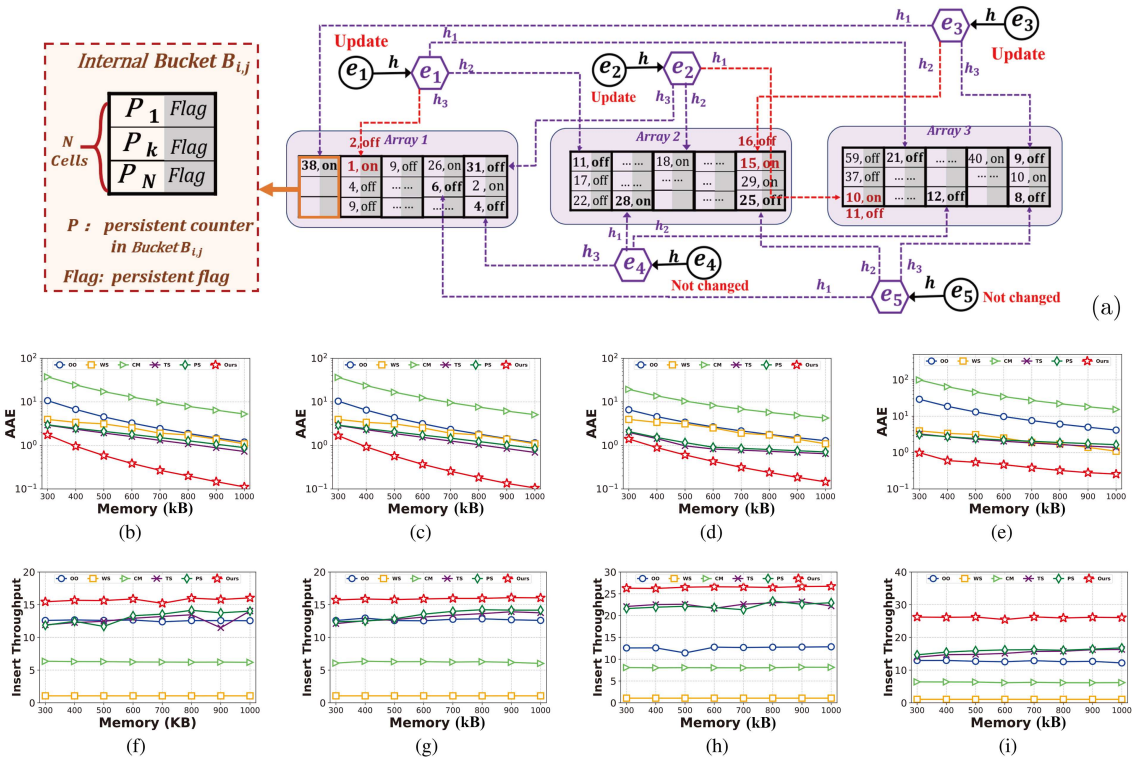

Figure 1 (Color online) Performance evaluation for persistence detection. (a) PF framework and operational example; (b)–(i) performance evaluation results.

cost while precisely counting per-window persistence.

Experiment. PF was evaluated on real-world traces from Center for Applied Internet Data Analysis (CAIDA) and an ISP backbone, each containing up to 30 million packets, with all algorithms sharing identical memory budgets (100–1000 kB). Figures 1(b)–(i) report persistence estimation results under identical sketch-memory budgets, without key storage for ID reporting. As shown in Figures 1(f)–(i), PF achieved an insertion throughput up to 1.9× higher than OO, while Figures 1(b)–(e) show that its average absolute error (AAE) is about 30% lower than that of WavingSketch. The measured throughput reached 28 MOPS, supporting line-rate analysis even under constrained resources.

PF was further compared with recent persistent measurement frameworks, including TS [2] and P-sketch (PS) [3], and achieved comparable accuracy but significantly higher update throughput (up to 2.3×), mainly owing to the one-hashing optimization scheme.

We conducted comprehensive evaluations of AAE, average relative error (ARE), and throughput across multiple datasets. The AAE and ARE metrics exhibited highly consistent trends: PF consistently reduced estimation errors compared with all baselines, with AAE closely tracking ARE across varying memory sizes. Due to space constraints, only AAE and insertion-throughput results are presented in Figure 1; ARE results are provided in the supplementary file and demonstrate similar relative improvements. Overall, PF achieves both high accuracy and fast updates under strict memory budgets.

Conclusion. We present PF, a fast and accurate framework for detecting persistent flows in high-speed networks. PF achieved near $O(1)$ complexity for both update and query operations by integrating a one-hashing technique and an ostrich policy. Extensive experiments on real network traces demonstrated that PF outperformed baseline sketches in terms of both accuracy and throughput under memory constraints. Given its simplicity and scalability, PF provides a promising foundation for deployment in programmable switches and online monitoring systems.

Acknowledgements This work was supported by Joint Funds of the National Natural Science Foundation of China (NSFC)/Research Grants Council (RGC) Collaborative Research Scheme (Grant Nos. 62461160332, CRS HKUST602/24), and PCL-CMCC Foundation for Science and Innovation (Grant No. 2024ZY2B0050).

Supporting information Appendixes A–J. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Zhang Y, Li J, Lei Y, et al. On-off sketch: a fast and accurate sketch on persistence. Proc VLDB Endow, 2020, 14: 128–140
2 Li W, Patras P. TightSketch: a high-performance sketch for heavy item-oriented data stream mining with limited memory size. In: Proceedings of the ACM International Conference on Information and Knowledge Management, 2023
3 Li W, Patras P. P-sketch: a fast and accurate sketch for persistent item lookup. IEEE ACM Trans Netwing, 2024, 32: 987–1002
4 Li W, Bütün B, Chu T, Fiore M, Patras P. PALLAS: a data-plane-only approach to accurate persistent flow detection on programmable switches in high-speed networks. In: Proceedings of IEEE the 33rd International Conference on Network Protocols (ICNP), 2025. 1–11
5 Li W. Pandora: an efficient and rapid solution for persistence-based tasks in high-speed data streams. In: Proceedings of the ACM on Management of Data, 2025