Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.078464

ARTICLE

# PIF-Identifier: Accurate Low-Overhead Identification of Persistent Infrequent Flows in Network Traffic

**Bing Xiong[1], Zhuoxiong Li[1], Yongqing Liu[1], Yu Tang[1] and Jinyuan Zhao[2,*]**

[1]School of Computer Science and Technology, Changsha University of Science and Technology, Changsha, China
[2]School of Information Science and Engineering, Changsha Normal University, Changsha, China

*Corresponding Author: Jinyuan Zhao. Email: zhaojy@csnu.edu.cn

Received: 31 December 2025; Accepted: 28 February 2026; Published: 08 May 2026

**ABSTRACT:** Persistent Infrequent Flows (PIFs) refer to the packet flows that last for a long time but always at low frequencies in network traffic. Accurate identification of the PIFs plays a vital role in intrusion detection, attack prevention, traffic engineering, and other network fields. However, existing methods often require to save all flows for finding out the PIFs due to their infrequency feature, which brings about the problem of low identification accuracy and high memory overhead. To solve this problem, this paper proposes an accurate PIF identification method with low overhead called PIF-Identifier, composed of a new-flow discriminator and a PIF tracker. Specifically, we first design a compact new-flow discriminator by applying probabilistic data structures, to quickly determine whether a packet flow arrives for the first time within current time window. Then we design a PIF tracker to accurately identify and report persistent infrequent flows. In the PIF tracker, we configure a small-size frequency counter for each tracked flow in accordance with the frequency threshold of the PIF, without sacrificing the accuracy of PIF identification. Furthermore, we design a probabilistic replacement strategy based on the number of time windows of flow persistence, to accommodate newly arrived potential PIFs when there is no vacancy in their mapped buckets of the PIF tracker. Finally, we evaluate the performance of our proposed PIF-Identifier by theoretical analysis and experimental verification with real network traffic traces. Experimental results indicate that the PIF-Identifier achieves the precision of 100%, much higher recall rate and F1 score, as well as lower average relative error than the state-of-the-art methods, significantly promoting the identification performance of persistent infrequent flows.

**KEYWORDS:** Network traffic measurement; persistent infrequent flows; low-overhead PIF identification; new-flow discriminator; probabilistic replacement strategy

## 1 Introduction

Network traffic measurement provides key information for a variety of network management tasks, including traffic behavior analysis, QoS (quality of service), performance diagnosis, and anomaly detection, etc. [1,2]. It has become an indispensable method to effectively understand and manage network performance. Until now, a popular measurement method is to apply the compact data structure Sketch to perform approximate estimation tasks in network measurement, such as elephant flow identification [3], flow frequency estimation [4]. In recent years, researchers have begun to focus on other important characteristics of packet flows, such as liveness [5,6], persistence [7], and bursty [8], steadiness [9] and persistent infrequence [10]. Persistent Infrequent Flows (PIFs) refer to the flows where packets arrive persistently but at low frequency. The identification of PIFs has widespread applications in the field of network security, such as the detection of advanced persistent threats (APT) [11,12] and low-rate DDoS (distributed denial of

service) attacks [13,14]. Therefore, it is crucial to efficiently identify and accurately report PIFs for supporting intrusion detection, attack prevention, traffic engineering, and other fields.

A straightforward PIF identification approach is to find the intersection of persistent flows and infrequent ones. Inspired by elephant flow identification, many researchers usually measure persistent flows by applying sketched-based methods, such as PIE [15] and On-Off Sketch [16], for low memory overhead. Meanwhile, existing methods estimate flow frequency usually by improving Count-Min (CM) Sketch [17], such as Elastic Sketch [18] and Augmented Sketch [19]. These methods primarily focus on the identification of frequent flows rather than infrequent ones. Up to now, PISketch [10] is the first method that combines and optimizes the above two tasks, and has achieved good performance in identifying PIFs. However, this method has to allocate large bits for its weight counters, to accommodate varying number of packets in various flows, resulting in excessive memory overhead. In addition, the method merely outputs PIFs at the end of whole measurement process, which can not report the specific time windows of flow persistence, and may misjudge frequent flows as infrequent ones. To address above problems, this paper propose a low-overhead identification method of persistent infrequent flows called PIF-Identifier, with main contributions summarized as follows:

- Proposing a low-overhead identification method of persistent infrequent flows called PIF-identifier, which designs a compact new-flow discriminator to quickly determines whether a flow appears for the first time in current time window, and a PIF tracker to track potential PIFs and report real ones.
- Designing a minimal-size configuration strategy of flow frequency counters in the PIF tracker, which configures a minimal size for each counter based on the frequency threshold of infrequent flows, reducing the memory overhead of PIF-identifier.
- Devising a probabilistic replacement strategy to accommodate newly arrived potential PIFs when there is no vacancy in their mapped buckets, enhancing the identification accuracy of persistent infrequent flows.
- Proving that PIF-Identifier has no false positive error and deriving an upper bound on its probability of false negative error, for confirming its superior performance on identifying persistent infrequent flows.

## 2 Related Work

Sketch-based data structures are widely used for flow frequency estimation, achieving high accuracy and speed with limited memory. The most typical Sketch algorithm is CM Sketch [17] which increments all mapping counters of arriving packet by one, and returns the minimum of these counters as the estimated size of the flow to which the packet belongs. To improve accuracy, Conservative Update(CU) Sketch [20] increments only the smallest mapped counter, but frequent hash collisions can still lead to overestimation. To address the inherent skewness of network traffic, researchers have conducted various research on data structures and statistical strategies. For example, Cold Filter [21] employs a two-layer filter to pre-filter a large number of small flows, reducing hash collisions and improving estimation accuracy. HeavyGuardian [22] adopts an exponentially attenuated decay strategy to aggressively purge small flows, enhancing the frequency estimation of large ones. Meanwhile, Elastic Sketch [18] separates large and small flows into heavy and light parts and utilizes voting strategy to expel small flows to light part in time, achieving high frequency estimation accuracy of large flows and small ones.

Several Sketch-based algorithms have been developed for flow persistence estimation. Small Space [23] adopts the idea of “sampling and counting” to track the persistence of flows by maintaining a hash table. Although sampling reduces space overhead, Small Space still records many non-persistent items, resulting in suboptimal memory efficient. To address this issue, PIE [15] encodes each flow ID into multiple Raptor code segments and randomly stores a subset in each space-time Bloom filter to reduce memory overhead.

However, its memory requirements increase with the growing number of time windows, and high decoding complexity prevents line-rate processing. On-off Sketch [16] sets a switch flag for each counter to control whether each mapped counter increases its value, ensuring that each counter increases by at most once per time window. Although it avoids underestimation, hash conflicts can still lead to overestimation of flow persistence. On this basis, P-Sketch [24] introduces the concept of flow heat and proposes a probabilistic replacement algorithm based on the number of duration window and heat. This strategy prioritizes the retention of highly persistent flows, improving identification accuracy of persistent flows.

The above mentioned methods have achieved good performance in estimating flow frequency and persistence separately, but none of them can directly identify persistent infrequent flows. As a representative solution of identifying PIFs, PISketch [10] defines a flow weight that comprehensively reflects the persistence and frequency of each flow. The weight is adjusted through a reward and punishment mechanism, and the flows with higher weight are reported as PIFs. However, weight counters often require a large number of bits to accommodate flows with various sizes, resulting in excessive memory overhead. At the same time, PISketch adopts measurement results in all time windows for overall judgment, which ignores interrupt arrivals and high-frequency phenomenon in some time windows, resulting in misjudgments of PIFs. In addition, it can only report the number of time windows with flow persistence for a PIF, but not specific starting and ending time windows.

## 3 PIF-Identifier

### *3.1 Methodology*

Different from the aforementioned solutions, this paper further clarifies the criterion of PIF as a certain number of consecutive time windows with infrequent packet arrivals. According to this criterion, we limit the size of frequency counter for each potential PIF in terms of its frequency threshold. If the frequency of a potential PIF reaches the threshold, we will not increase its frequency counter as it no longer satisfies infrequency characteristics. Furthermore, we record starting and ending time windows to obtain the number of consecutive time windows, for each potential PIF maintaining infrequency. Subsequently, we design a probabilistic replacement strategy based on the number of consecutive time windows, to accommodate new potential PIFs when there is no vacancy in their mapped buckets. With these design concepts, we propose a low-overhead method that accurately identifies persistent infrequent flows called PIF-Identifier. Fig. 1 illustrates the data structure of our proposed PIF-Identifier, comprised of a new-flow discriminator and a PIF tracker. Table 1 shows the symbols commonly used in the PIF-Identifier and their meanings.

As shown in Fig. 1, the new-flow discriminator is a Bloom filter consisting of $L$ counters. For each arrived packet, it is mapped to $k$ counters of the new-flow discriminator through $k$ independent hash functions $h_1(.)$, $h_2(.)$, ..., $h_k(.)$. If the mapped counters are not all 1, the new-flow discriminator will report that the flow corresponding to the packet arrives for the first time in current time window. In this case, we will record the flow to the PIF tracker and set all mapped counters in the new-flow discriminator to 1. Otherwise, we update the flow corresponding to the arrived packet in the PIF tracker. At the end of each time window, We reset all counters of the new-flow discriminator.

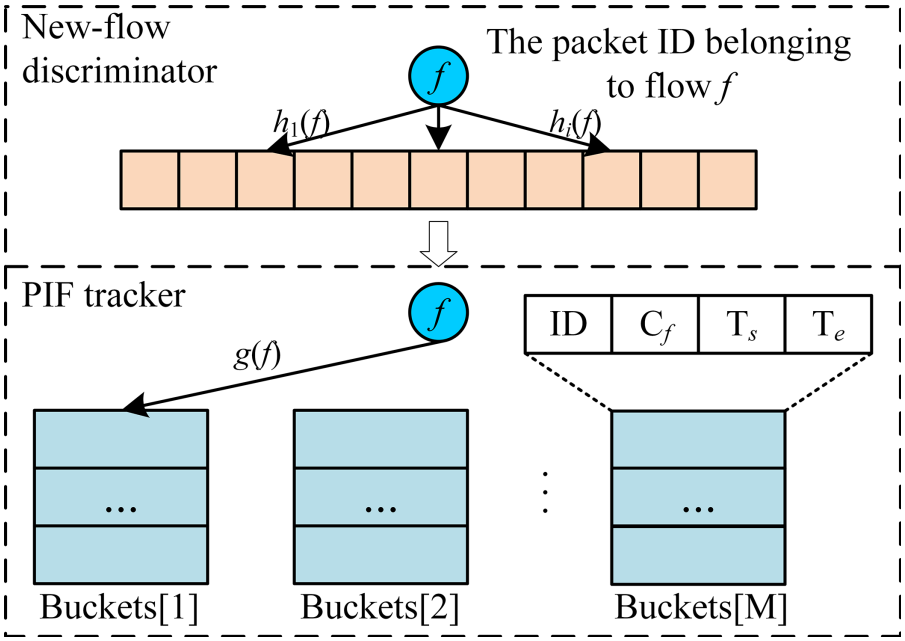


**Figure 1:** The data structure of PIF-Identifier [1,2].

**Table 1:** Frequently-used notations in this paper.

| Symbol | Meaning |
|---|---|
| $h_i(.)$ | The $i$-th hash function of the new-flow discriminator |
| $B[i]$ | The $i$th bucket of the PIF tracker |
| $M$ | The number of buckets in the PIF tracker |
| $N$ | The number of cells contained in each bucket in the PIF tracker |
| $\alpha$ | The frequency threshold of packet arrivals within a persistent infrequent flow(PIF) at each time window |
| $\beta$ | The number threshold of consecutive time windows with infrequent packet arrivals in a PIF |
| $C_f$ | The flow frequency counter in the PIF tracker |
| $T_s$ | The starting window recorder in the PIF tracker |
| $T_e$ | The ending window recorder in the PIF tracker |

The PIF tracker is a hash table composed of $M$ hash buckets $B[1], B[2], ..., B[M]$, associated with a hash function $g(.)$. Each bucket contains $N$ cells, and each cell stores the information of a packet flow: flow ID, frequency counter $C_f$, starting time window $T_s$, and ending time window $T_e$. Flow ID uniquely identify the packet flow, and $C_f$ record the number of packets arrived within the flow in current time window, no more than the frequency threshold $\alpha$. By this way, the counter only requires a few bits to determine the infrequency characteristics of each flow, reducing the memory overhead of PIF-Identifier. $T_s$ and $T_e$ respectively record the starting and ending time window of flow infrequency duration. A flow is reported as a PIF only if it persistently arrives within at least $\beta$ time windows, and the number of packets in each time window is less than $\alpha$.

### *3.2 Basic Operations*

**Packet processing:** When a packet $e$ belonging to the flow $f$ arrives in current time window numbered as $i$, we first query the new-flow discriminator to check whether the flow $f$ has appeared in this window. If the new-flow discriminator reports positive, it indicates that the flow $f$ has appeared in this window. Otherwise, it indicates that the flow $f$ appears for the first time in this window. Then we map the flow $f$ into the bucket $B[g(f)]$ in the PIF tracker by the hash function $g(.)$, and perform different operations for the following cases:

Case 1: The flow $f$ is stored in a cell of the bucket $B[g(f)]$ and the new-flow discriminator reports negative. This indicates that the flow $f$ appears for the first time in this window, and the flow $f$ arrived consecutively until current time window. In this case, we update $B[g(f)].T_e = T_i$, and set $B[g(f)].C_f = 1$.

Case 2: The flow $f$ is stored in a cell of the bucket $B[g(f)]$ and the new-flow discriminator reports positive. This indicates that the flow $f$ has appeared in the $i$th time window. In this case, we increment the frequency counter $B[g(f)].C_f$, if it does not reach the frequency threshold $\alpha$. Otherwise, it will not be updated.

Case 3: The flow $f$ is not stored in the bucket $B[g(f)]$ and the new-flow discriminator reports negative, but there are at least one empty cell in the bucket $B[g(f)]$. In this case, we store the flow $f$ into the first empty cell and set the flow ID as $f$, and $C_f = 1$, $T_s = T_i$, $T_e = T_i$.

Case 4: The flow $f$ is not stored in the bucket $B[g(f)]$, the new-flow discriminator reports negative and there is no empty cell in the bucket $B[g(f)]$. In this case, we calculate the the number of consecutive time windows of the flow in the $j$th $(0 \le j < N)$ cell of the bucket $B[g(f)]$ as $\Delta T = B[g(f)][j].T_e - B[g(f)][j].T_s + 1$. Then we replace the flow with the smallest $\Delta T$ in the bucket $B[g(f)]$ with the flow $f$ by the probability $1/\min\Delta T$. If the replacement succeeds, we set the flow ID as $f$, and $C_f = 1$, $T_s = T_i$, $T_e = T_i$.

Case 5: The flow $f$ is not stored in the bucket $B[g(f)]$ and the new-flow discriminator reports positive. In this case, we perform a probabilistic flow replacement operation similarly to Case4.

**Examples:** Fig. 2 shows some examples of packet processing. Suppose the frequency threshold $\alpha = 15$, the number threshold of consecutive time windows $\beta = 5$, and the number of current time window $T_i = 10$. (1) As for an arrived packet in the flow $f_1$, the new-flow discriminator reports negative, which means that the packet is the first one of the flow $f_1$ in the $T_{10}$ time window. So we set all mapped counter in the new-flow discriminator to 1, and insert packet into the PIF tracker. Then we set $B[g(f_1)].C_f$ to 1 and $B[g(f_1)].T_e$ to $T_{10}$ due to the flow $f$ has already been stored in the PI tracker. (2) As for an arrived packet in the flow $f_1$, the new-flow discriminator reports positive, and we insert the packet into the PIF tracker. Then we increment the frequency counter $B[g(f_2)].C_f$, due to $B[g(f_2)].C_{f_2} = 7 < \alpha$. (3) As for an arrived packet in the flow $f_3$, its processing is similar to that of the flow $f_2$, but we no longer increment the counter $B[g(f_3)].C_f$ because it has reached the frequency threshold $\alpha$. (4) As for an arrived packet in the flow $f_4$, the new-flow discriminator reports negative, and we insert the packet into the PIF tracker. Then we insert the information $<f_4, 1, T_{10}, T_{10}>$ into an empty cell. (5) As for an arrived packet in the flow $f_5$, the new-flow discriminator reports negative, and we insert the packet into the PIF tracker. Then we calculate the $\Delta T$ of all flows in the mapped bucket of the flow $f_5$. Finally, we successfully replace the flow $f_9$ with $f_5$. (6) As for an arrived packet in the flow $f_6$, the new-flow discriminator reports positive, and we insert the packet into the PIF tracker. The flow $f_6$ is not stored in its full mapped bucket, and we perform a probabilistic flow replacement operation.

**Clearing and reporting:** At the end of each time window, we check each stored flow in the PIF tracker. As for the flow $f$, it will be cleared out if its frequency counter $C_f$ reaches the threshold $\alpha$ or its ending time window $T_e$ is not current one. Otherwise, we calculate the number of consecutive time windows $\Delta T = T_e - T_s + 1$ of the flow $f$, and check whether it has reached the threshold $\beta$. If so, we report the flow $f$ with

its starting and ending time windows. After all cells in the PIF tracker have been checked, we enter the next time window.

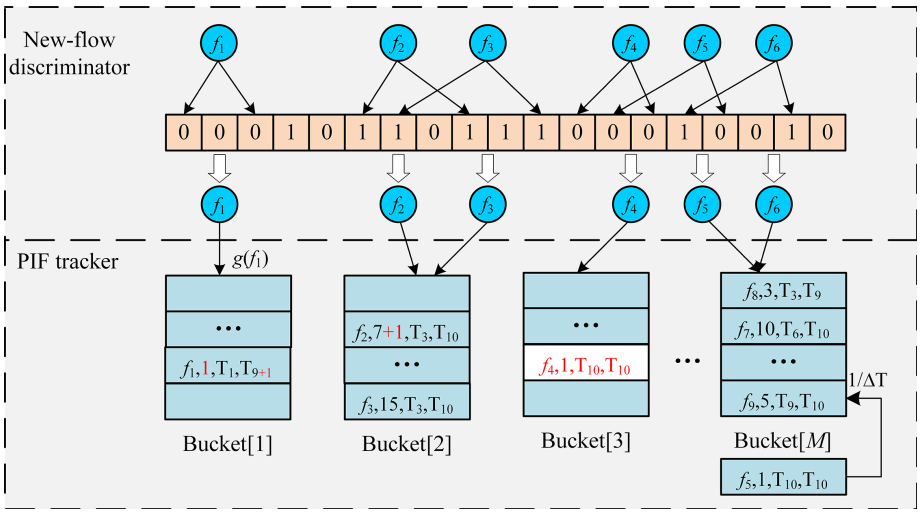


**Figure 2:** The examples of flow insertion [1–3].

### *3.3 Theoretical Analysis*

**Theorem 1:** *Given that measured network traffic consists of n packets, which belong to m flows containing l PIFs, and are uniformly divided into w time windows. Suppose all flows are mapped into the new-flow discriminator and the PIF tracker by uniform k hash functions* $h_i()(0 < i \le k)$ *and* $g(.)$*. PIF-Identifier has no false positive error, and its false negative rate (FNR) satisfies:*

$$P_{FNR} \le \frac{l}{m}\left(1 - e^{-\frac{kn}{wL}}\right)^k + 1 - \left(1 - \frac{M}{N}\right)^w \tag{1}$$

We first analyze the packet processing of the PIF-Identifier. It can be inferred from Section 3.3 that: (1) The PIF-Identifier does not generate false negative errors in Case1 and Case3, because the new-flow discriminator reports negative and Bloom filters do not have false negative errors. (2) In Case2, we will overestimate the frequency of a real PIF if there is a false positive error for the new-flow discriminator. (3) In Case4, false negative errors can also occur in the PIF-Identifier when flow replacement operations evict real PIFs. (4) In Case5, the PIF-Identifier does not produce a false positive error, and its false negative error rate is negligible because there is still a chance for a real PIF to perform a flow replacement in current time window.

As seen from the above analysis, the PIF-Identifier has no false positive errors, while false negative errors are mainly generated in Case2 and Case4. Then we analyze the probability of false negative errors in these two cases. Suppose the new-flow discriminator has $L$ counters initialized to 0, we can express the probability that a counter in the new-flow discriminator is not mapped by a packet through $k$ uniform hash functions as $(1-1/L)^k$. Since there are $n$ packets in measured network traffic uniformly divided into $w$ time windows, each time window has $n/w$ packets. Consequently, we can express the probability that any counter in the new-flow discriminator being 1 after inserting all packet in single time window as $1-(1-1/L)^{kn/w}$

Therefore, we can derive the false positive error rate of the new-flow discriminator as:

$$P = \left[1 - \left(1 - \frac{1}{L}\right)^{kn/w}\right]^k \simeq \left(1 - e^{-\frac{kn}{wL}}\right)^k \tag{2}$$

Since there are $l$ PIFs in measured network traffic, we can infer the false negative error rate of the PIF-Identifier in Case2 as:

$$P_{FNR1} \leq \frac{l}{m}\left(1 - e^{-\frac{kn}{wL}}\right)^k \tag{3}$$

As for Case4, we can obtain the probability of a PIF being incorrectly replaced as $1/MN$, since there are $N$ cells in each bucket and $M$ hash buckets in the PIF tracker. Subsequently, we can infer the probability that all PIFs are not replaced in all $w$ time windows as $(1-1/MN)^w$. Consequently, we can obtain the false negative error rate of the PIF-Identifier by calculating the ratio of unreported true flows to total flows as:

$$P_{FNR2} = \frac{l \times \left(1 - \left(1 - \frac{1}{MN}\right)^w\right)}{l} = 1 - \left(1 - \frac{1}{MN}\right)^w \tag{4}$$

In summary, we can infer the false negative error rate of PIF-Identifier $P_{\text{FNR}}$ in Eq. (5), by combining Eqs. (3) and (4).

$$P_{\text{FNR}} < p_{FNR1} + P_{FNR2} \leq \frac{l}{m}\left(1 - e^{-\frac{kn}{wL}}\right)^k + 1 - \left(1 - \frac{1}{MN}\right)^w \tag{5}$$

Time complexity analysis: For each arrived packet, the new-flow discriminator needs to perform $k$ constant-time hash operations and memory accesses. Meanwhile, the PIF tracker performs lookup on $N$ cells at most in the mapped bucket of the packet, and inserts or updates flow information. Therefore, it takes the time complexity $O(k+N)$ for packet processing in the worst case. This complexity is usually low enough to support line-rate processing, because $k$ and $N$ are typically configured as small constants.

## 4 Experiments

### *4.1 Experimental Setup*

**Datasets:** We select three high-speed network traffic traces TRACE20130222, TRACE20141109 and TRACE20180308 released by Jiangsu Province Key Laboratory of Computer Network Technology as experimental datasets. The three traces were collected from the Jiangsu Province border 10 Gbps backbone link of China Education and Research Network on 22 February 2013, 09 November 2014 and 06 December 2018, respectively. Specifically, the TRACE20130222, TRACE20141109 and TRACE20180308 have a total of 2,991,496, 3,064,040 and 3,063,134 packets respectively, belonging to 419,447, 302,279 and 212,782 flows. As for each packet in the dataset, we extract its timestamp, and flow ID composed of 5 tuples (source IP address and port number, destination IP address and port number, and protocol type).

**Performance metrics:**

1) **Precision:** The proportion of true PIFs in all PIFs reported by each PIF identification method.
2) **Recall:** The proportion of all true PIFs correctly reported by each PIF identification method.
3) **F1 Score:** $\frac{2 \cdot PR \cdot RR}{PR+RR}$, where $PR$ represents the precision rate, $RR$ represents the recall rate, and comprehensively reflects the accuracy of each PIF identification method.
4) **Average Relative Error (ARE):** $\frac{1}{\varphi}\sum_{i=1}^{\varphi}\frac{|p_i - \hat{p}_i|}{p_i}$, where $\varphi$ is the total number of PIFs reported by each PIF identification method, $p_i$ is the true number of consecutive time windows for the $i$th flow, and $\hat{p}_i$ is the measured number of consecutive time windows for the $i$th flow. It is used to evaluate the measurement error of the number of consecutive time windows for all PIFs reported by each identification method.

**Implementation:** We implement PISketch, On-Off/CM Sketch and our proposed PIF-Identifier by C++ programming in this paper, and conduct comparative experiments on a physical machine with dual 6-core CPU (24 threads, Intel Xeon Silver 4214R @2.4 GHz) and 32 GB DRAM. For the PISketch, memory is allocated for its Bloom filter and weight Sketch with the ratio 1:9. The Bloom filter adopts three hash functions, while the weight Sketch sets the number of cells per bucket to 5. As for the On-Off/CM Sketch, we combine an On-Off Sketch for estimating flow persistence and a CM Sketch for estimating flow frequency to identify PIFs. Specifically, memory is allocated for them with the ratio 1:3. Meanwhile, we set the number of rows in the CM Sketch and the number of cells in each bucket of the On-Off Sketch both to 3. As for our proposed PIF-Identifier, memory is split between the new-flow discriminator and the PIF tracker with the ratio 1:9. The new-flow discriminator employs three hash functions, and the PIF tracker is configured with 5 cells per bucket. In our experiments, we set the frequency threshold $\alpha$ to 30, and the number threshold of consecutive time windows $\beta$ to 10.

With the above configuration, we perform each PIF identification method on three datasets. Each method reports a set of PIFs, along with their measured number of consecutive time windows. We obtain the number of correctly identified PIFs by verifying reported results with the ground truth. Finally, we compute the performance metrics of each PIF identification method under identical memory constraints.

### *4.2 Parameter Setting*

This subsection evaluates the impact of key parameters for PIF-Identifier on its performance. The parameters include the frequency threshold $\alpha$, the number threshold of consecutive time windows $\beta$, the number of cells per bucket $N$, and the memory allocation ratio between the new-flow discriminator and the PIF tracker. Experiments are conducted on the TRACE20180308 dataset, with the F1 score and ARE as evaluation metrics.

**Impact of the number of cells per bucket *N*:** We vary $N$ from 5 to 25 while holding other parameters constant. Fig. 3 shows the effect of varying $N$ on the performance of PIF-Identifier. As seen from Fig. 3, the F1 score of PIF-Identifier achieves the maximum value at $N = 5$ for different memory sizes. The F1 score of PIF-Identifier shows a downward trend overall as $N$ increases, while its ARE is stable overall. This is because the number of buckets in PIF-Identifier decreases as the number of cells per bucket increases. At this moment, the probability of a hash collision increases when PIFs are mapped to an identical bucket, leading to performance degradation. Therefore, we set $N$ to 5.

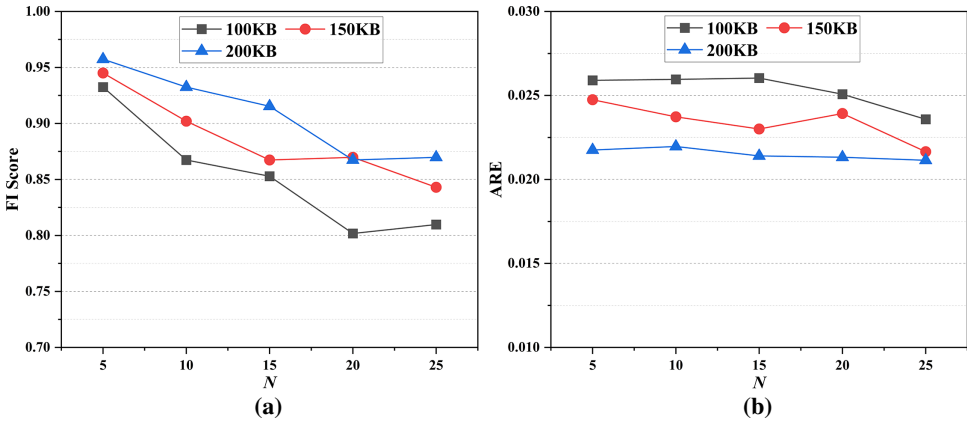


**Figure 3:** Effect of $N$. (**a**) F1 Score. (**b**) Average Relative Error(ARE).

**Impact of the memory allocation ratio of the new-flow discriminator to the PIF tracker:** We vary the proportion of memory allocated to the new-flow discriminator while keeping other parameters fixed. Fig. 4 shows the effect of memory allocation proportion of the new-flow discriminator on the performance of the PIF-Identifier. As shown in Fig. 4, the F1 score of the PIF-Identifier gradually decreases as the memory allocation proportion of the new-flow discriminator increases, when the total memory is 100 KB. However, the F1 score of PIF-Identifier first increases and then decreases, with the increasing memory allocation proportion of the new-flow discriminator, for the total memory of 150 KB and 200 KB. This is because the increase of the memory allocation proportion of the new-flow discriminator means the reduction on the number of buckets in the PIF tracker. This will increase the hash collisions and reduce the measurement accuracy of the PIF tracker. In summary, we set the memory allocation ratio of the new-flow discriminator to the PIF tracker as 1:9 in consideration of memory constraint.

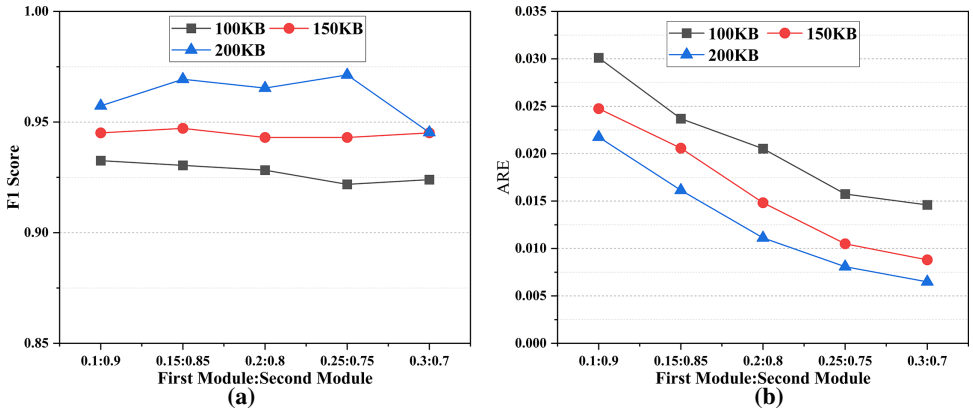


**Figure 4:** Effect of memory allocation. (**a**) F1 Score. (**b**) ARE.

**Impact of the frequency threshold $\alpha$:** We gradually increase the frequency threshold $\alpha$ from 15 to 60 while keeping the remaining parameters unchanged. Fig. 5 demonstrates the impact of $\alpha$ on the performance of PIF-Identifier. As seen from Fig. 5, the PIF-Identifier maintains stable performance regardless of how $\alpha$ changes. In summary, we set it to 15 in our experiments.

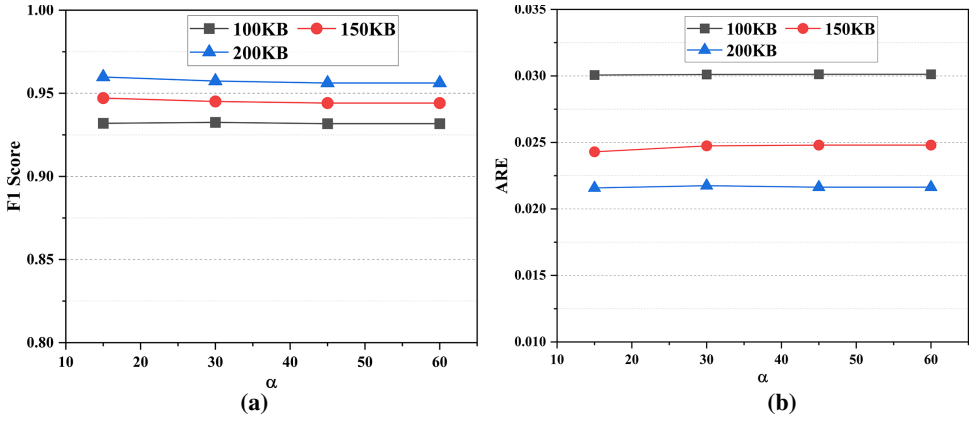


**Figure 5:** Effect of $\alpha$. (**a**) F1 Score; (**b**) ARE.

**Impact of the number threshold of consecutive time windows $\beta$:** We increase $\beta$ from 5 to 25 while keeping the remaining parameters unchanged. Fig. 6 exhibits the effect of $\beta$ on the performance of PIF-Identifier. As seen from Fig. 6, the F1 score of PIF-Identifier shows an overall increasing trend, as $\beta$ increases. This is because the number of PIFs will decrease with the increase of $\beta$ in the definition of PIF, resulting in fewer hash collisions and flow replacements. We set the number threshold of consecutive time windows $\beta$ to 10 in this experiment to verify the performance of PIF-Identifier.

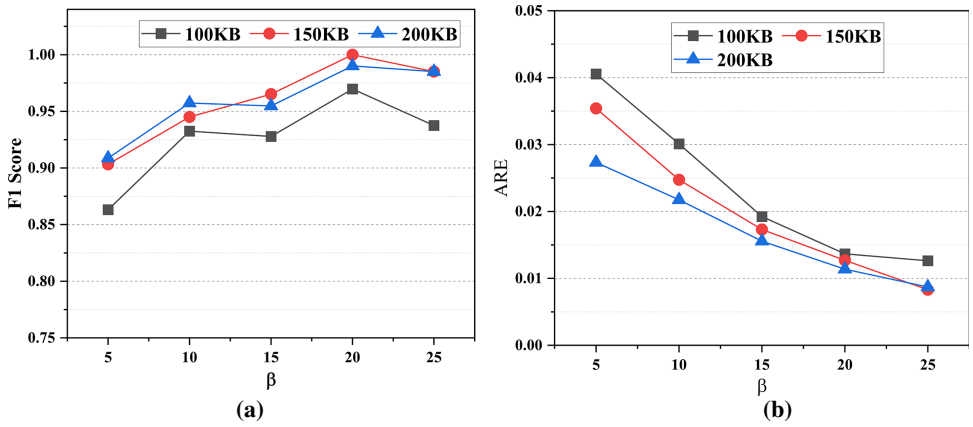


**Figure 6:** Effect of $\beta$. (**a**) F1 Score. (**b**) ARE.

Advanced Persistent Threats (APTs) [25] typically maintain stealth command and control channels, by low-frequency periodic beaconing with the intervals ranging from tens of seconds to several hours. This results in extremely low packet rates often below one packet per second (pps), while the channels may persist for weeks or even months. In our experiments, setting $\alpha$ to 15 implies that a detected flow must not exceed 15 packets per time window. This threshold is significantly higher than the typical beaconing rates of APT traffic (usually <1 pps), ensuring the capture of such activities. Simultaneously, it is sufficiently low to effectively filter out most normal traffic. Furthermore, incidental low-rate network events rarely sustain a regular low-frequency pattern across more than 10 consecutive time windows. Setting $\beta$ to 10 helps distinguish genuinely persistent malicious activities from such benign incidental events.

### *4.3 Performance of PIF Identification*

In this subsection, we conduct extensive experiments on three datasets to compare the performance of PIF-Identifier with existing PIF identification methods. We evaluate the performance of each method by precision, recall rate, F1 score and ARE. In the experiments, we increase the memory capacity of each method from 100 to 300 KB while keeping other parameters constant.

**Precision:** Fig. 7 illustrates the precision of each PIF identification method on different datasets. As seen from Fig. 7, PIF-Identifier consistently achieves higher precision than other existing methods for identical memory capacity. Specifically, the precision of PIF-Identifier consistently achieves 100% under different memory capacity on three datasets, which aligns with its zero false-positive rate by theoretical proof in Section 4. In contrast, the precision of existing methods depends on their memory capacity. Specifically, the precision of PISketch decreases to 81% and 63.8% in the dataset 20141108, respectively for the memory capacity 100 and 300 KB. On-Off/CM Sketch has the lowest precision in three datasets and its precision gradually decreases with the increasing memory capacity. Taking the dataset 20130222 for example, its

precision decreases to 60% and 43.5%, respectively for the memory capacity 100 and 300 KB. Overall, PIF-Identifier outperforms the other two methods, improving the precision by 9.5% over PISketch and 46.2% over On-Off/CM Sketch under identical memory capacity.

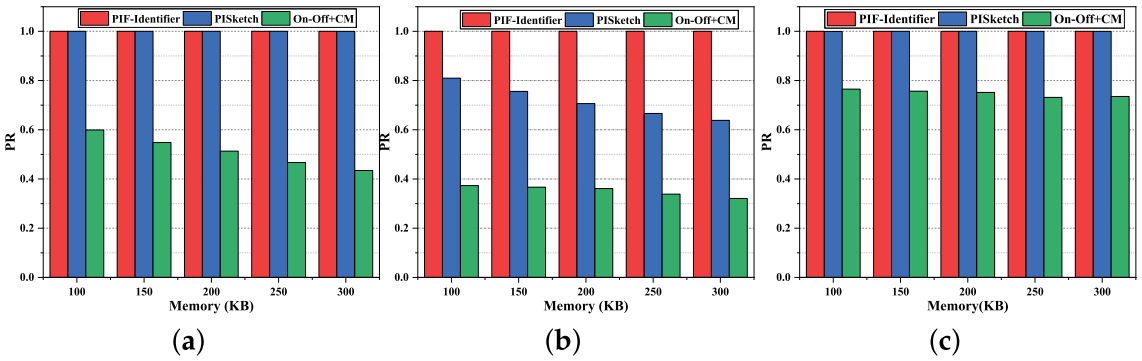


**Figure 7:** Precision. (**a**) TRACE20130222; (**b**) TRACE20141108; (**c**) TRACE20180308.

**Recall:** Fig. 8 shows the recall rate of each PIF identification method on different datasets. As shown in Fig. 8, the recall rate of each method exhibits a growth trend with the increase of memory capacity, while PIF-Identifier achieves highest recall rates on all three datasets. Specifically for the dataset 20130222, PIF-Identifier achieves a recall rate of 87.7% with the memory capacity 100 KB, while PISktech and On-Off/CM Sketch achieve recall rate of 32.3% and 59.9%, respectively. when the memory capacity is increased to 300 KB, their recall rates are 96.7%, 53.8%, and 51.5%, respectively. For the dataset 20141108, their recall rates decrease to 88.5%, 40.7%, and 19.8% respectively with the memory capacity 100 KB, and increase to 97.4%, 74.8%, and 51.3% respectively when the memory capacity is increased to 300 KB. In summary, our proposed PIF-Identifier achieves the highest recall rate with the promotion of 52.3% and 60.5%, respectively compared to PISketch and On-Off/CM Sketch.

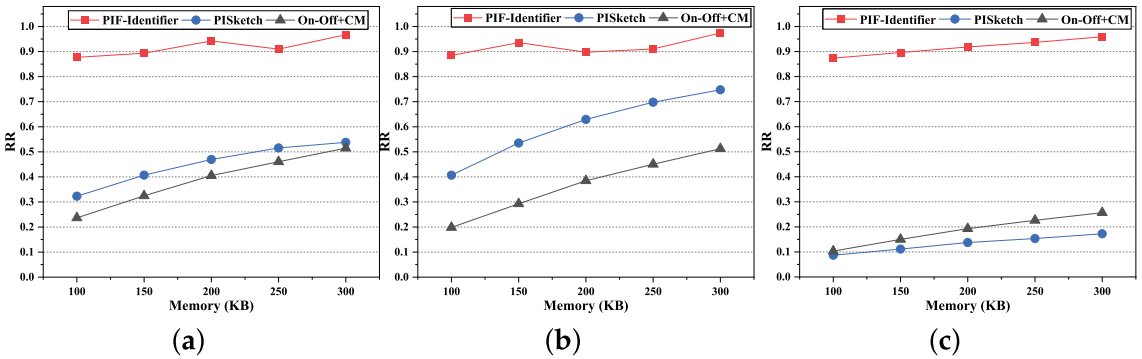


**Figure 8:** Recall. (**a**) TRACE20130222; (**b**) TRACE20141108; (**c**) TRACE20180308.

**F1 Score:** Fig. 9 exhibits the F1 score of each PIF identification method on different datasets. As seen from Fig. 9, the F1 score of PIF-Identifier consistently outperform those of other methods on all three datasets, above 0.85 even with small memory capacity. Specifically, PIF-Identifier achieves a F1 score of 0.934 with the memory capacity 100 KB on the dataset 20130222, while PISketch and On-Off/CM Sketch only obtain 0.488 and 0.339, respectively. When the memory capacity is increased to 300 KB, the F1 score of PIF-Identifier reaches up to 0.983, while PISketch and On-Off/CM Sketch only obtain 0.699 and 0.471, respectively. On the 20180308 dataset, PIF-Identifier achieves a F1 score of 0.933 with the memory capacity

100 KB. When the memory capacity is 300 KB, PIF-Identifier achieves a F1 score of 0.979, while PISketch and On-Off/CM Sketch only obtain 0.294 and 0.381, respectively. In summary, PIF-Identifier achieves the highest F1 score on all three datasets, outperforming PISketch and On-Off/CM Sketch respectively by 46% and 60.1% on average.

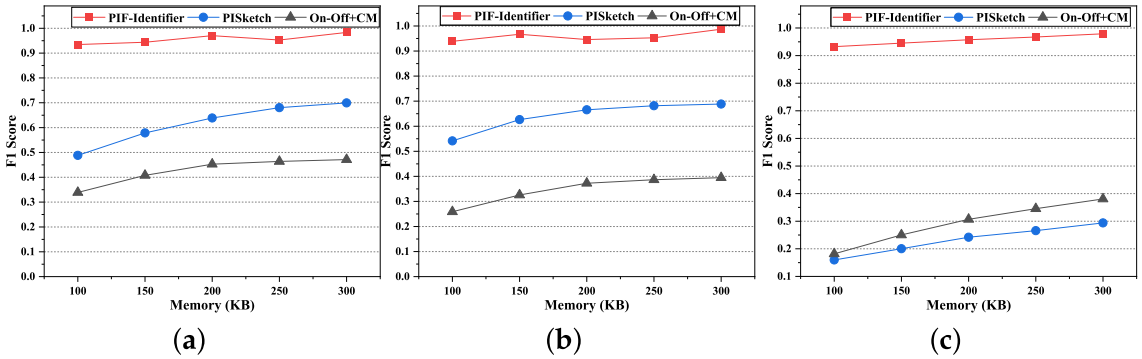


**Figure 9:** F1 score. (**a**) TRACE20130222; (**b**) TRACE20141108; (**c**) TRACE20180308.

**ARE:** Fig. 10 illustrates the log10ARE value of each PIF identification method on different datasets. As seen from Fig. 10, the log10ARE value of each method decrease gradually with the increase of memory capacity, and PIF-Identifier achieves relatively lower log10ARE values on all three datasets. Taking the dataset 20180308 for example, the log10ARE value of PIF-Identifier is −1.521 for the memory capacity 100 KB, while PISketch and On-Off/CM Sketch is −0.212 and 0.945, respectively. When the memory capacity is increased to 300 KB, the log10ARE of PIF-Identifier is −1.804, while the log10ARE of PISketch and On-Off/CM Sketch is −0.213 and 0.455, respectively. Overall, the log10ARE of PIF-Identifier is 136 and 2374 times respectively lower than those of PISketch and On Off/CM Sketch on average.

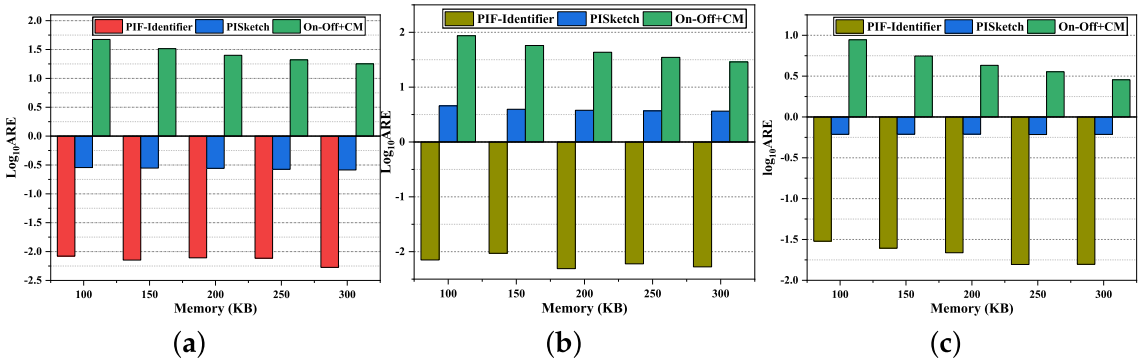


**Figure 10:** ARE vs. memory size. (**a**) TRACE20130222; (**b**) TRACE20141108; (**c**) TRACE20180308.

### *4.4 Result Analysis and Discussion*

The above experimental results demonstrate the superiority of PIF-Identifier in identifying PIFs. We next analyze the reasons for its better results than existing PIF identification methods.

On-Off/CM Sketch employs two data structures to track the persistence and low frequency. Furthermore, it adopts a simple replacement strategy to replace persistent flows, which leads many non-persistent flows to be misidentified as persistent ones, reducing the identification precision of PIF. At the same time, both structures need massive memory, and CM sketch has overestimation error. Therefore, each structure

has a fewer number of counters in the case of identical memory capacity, leading to frequent hash collisions. Furthermore, infrequent flows may be mistakenly identified as frequent ones, resulting in the missed identification of PIFs. As a result, the On-Off/CM Sketch has poor performance in identifying PIFs, especially under low memory capacity.

PISketch designs a weight to integrate persistence with low frequency, significantly reducing memory overhead compared to On-Off/CM Sketch. Therefore, it achieves relatively better performance in the case of low memory capacity. However, it brings about limited performance improvement, because its weight counter still needs to allocate more bits to record packet flows with various sizes. At the same time, PISketch ignores interruptions and high frequency phenomenons of recorded flows, leading to the misidentification of non-PIFs. In addition, PISketch reports PIFs based on flow weights, and cannot promptly clear out the flows those do not meet the conditions of PIFs. Therefore, it may not be able to store real PIFs, resulting in the miss of PIF identification. In summary, PISketch achieves insufficient precision and relatively large error.

In our PIF definition, we limit the packet frequency of PIFs in each time window within $\alpha$, and the number of consecutive time windows with infrequent packet arrivals over $\beta$. With the limitation, our proposed PIF-Identifier adopts frequency counters, starting and ending time windows to track the persistence and infrequency characteristic of each potential PIF. At the same time, it will promptly eliminate the flows that no longer meet the PIF definition at the end of each time window. In addition, we set small-sized frequency counters based on the frequency threshold of PIF, which significantly reduces the memory overhead without sacrificing the identification precision of PIFs. We also design a probability replacement strategy to accommodate newly arrived potential PIFs when there are no vacancies in the mapped bucket of the PIF tracker, further enhancing the identification precision of PIFs. In conclusion, our proposed PIF-Identifier can accurately identify PIFs with low memory overhead.

## 5 Conclusion

It is a crucial task to identify persistent infrequent flows in network traffic measurements, but existing identification methods suffer from high memory overhead and low accuracy. To solve this problem, this paper proposes a low-overhead method for accurate identification of persistent infrequent flows, called PIF-Identifier. It adopts the minimal-size configuration strategy of flow frequency counters to design a compact PIF tracker, which greatly reduces memory overhead. At the same time, it utilizes fast cleanup mechanism to clear out the flows that did not meet PIF definition, which effectively avoids the false positives of PIFs. Additionally, it employs probabilistic replacement strategy to make room for potential PIFs when there is no space in the PIF tracker, significantly enhancing the accuracy and efficiency of PIF identification.

We prove that the PIF-Identifier has superior performance on identifying persistent infrequent flows through mathematical analysis and comparative experiments. Experimental results demonstrate that PIF-Identifier outperforms existing methods with a precision of 100% and a recall rate of over 95%. Moreover, PIF-Identifier can achieve an F1 score higher than 0.9 even with the memory capacity 100 KB. At the same time, the error rate of PIF-Identifier is reduced by two orders of magnitude on average compared with existing methods. In summary, the PIF-identifier achieves high precision and memory efficiency in PIF identification, with good adaptability to network environments with limited memory resources.

In our future work, we will explore the pre-filtering strategy and adaptive capacity expansion mechanism for PIF-Identifier. Furthermore, we also intend to improve identification accuracy and adaptability to network traffic jitter and even malicious attacks. In the future, we also plan to deploy PIF-Identifier into various network applications, such as intrusion detection systems, network management platforms, etc.

**Acknowledgement:** The authors would like to thank Changsha University of Science and Technology.

**Funding Statement:** This work was supported in part by Hunan Provincial Natural Science Foundation of China (2023JJ30053, 2026JJ81174), Scientific Research Fund of Hunan Provincial Education Department (22A0232, 23A0735).

**Author Contributions:** Bing Xiong: Conceptualization, Writing—review and editing, Funding acquisition, Resources. Zhuoxiong Li: Formal analysis, Writing—original draft, Validation. Yongqing Liu: Methodology, Visualization. Yu Tang: Software, Data curation. Jinyuan Zhao: Project administration, Funding acquisition, Investigation, Supervision. All authors reviewed and approved the final version of the manuscript.

**Availability of Data and Materials:** Data is available upon request from the authors.

**Ethics Approval:** Not applicable.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Zhou Z, Abawajy J. Reinforcement learning-based edge server placement in the intelligent internet of vehicles environment. IEEE Trans Intell Transp Syst. 2025. doi:10.1109/TITS.2025.3557259.
2. Kirubavathi G, Pulliyasseri A, Rajesh A, Ajayan A, Alfarhood S, Safran M, et al. Enhancing IoT resilience at the edge: a resource-efficient framework for real-time anomaly detection in streaming data. Comput Model Eng Sci. 2025;143(3):3005–31. doi:10.32604/cmes.2025.065698.
3. Ye J, Li L, Zhang W, Chen G, Shan Y, Li Y, et al. Ua-sketch: An accurate approach to detect heavy flow based on uninterrupted arrival. In: Proceedings of the 51st International Conference on Parallel Processing (ICPP '22); 2022 Aug 29–Sep 1; Bordeaux, France. New York, NY, USA: ACM; 2022. p. 1–11. doi:10.1145/3545008.3545017.
4. Shahout R, Sabek I, Mitzenmacher M. Learning-augmented frequency estimation in sliding windows. In: 2024 IEEE 32nd International Conference on Network Protocols (ICNP); 2024 Oct 28–31; Charleroi, Belgium. Piscataway, NJ, USA: IEEE; 2024. p. 1–6. doi:10.1109/ICNP61940.2024.10858536.
5. Xiong B, Chang Y, Zhang Y, Zhang J, Zhao B, Li K. AF-detector: an accurate low-overhead method for detecting active flows in network traffic. Comput Netw. 2025;270(C):111562. doi:10.1016/j.comnet.2025.111562.
6. Xiong B, Liu Y, Liu R, Zhao J, He S, Zhao B, et al. ActiveGuardian: an accurate and efficient algorithm for identifying active elephant flows in network traffic. J Netw Comput Appl. 2024;224:103853. doi:10.1016/jjnca.2024.103853.
7. Xiao Q, Li Y, Wu Y. Finding recently persistent flows in high-speed packet streams based on cuckoo filter. Comput Netw. 2023;237:110097. doi:10.1016/j.comnet.2023.110097.
8. Xiong B, Xiao Q, Huang S, Cao J, Zhao B, Li K. BurstDetector: an accurate low-overhead detection algorithm for burst flows in network traffic. In: Proceedings of the 2025 IEEE International Symposium on Parallel and Distributed Processing with Applications; 2025 Oct 10–12; Shenyang, China. Piscataway, NJ, USA: IEEE; 2025. p. 642–9. doi:10.1109/ISPA67752.2025.00088.
9. Li X, Fan Z, Li H, Zhong Z, Guo J, Long S, et al. SteadySketch: finding steady flows in data streams. In: 2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS); 2023 Jun 19–21; Orlando, FL, USA. Piscataway, NJ, USA: IEEE; 2023. p. 1–9. doi:10.1109/IWQoS57198.2023.10188743.
10. Fan Z, Hu Z, Wu Y, Guo J, Liu W, Yang T, et al. Pisketch: finding persistent and infrequent flows. In: Proceedings of the ACM SIGCOMM Workshop on Formal Foundations and Security of Programmable Network Infrastructures; 2022 Aug 22–26; Amsterdam, The Netherlands. New York, NY, USA: ACM; 2022. doi:10.1145/3528082.3544834.
11. Alshamrani A, Myneni S, Chowdhary A, Huang D. A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities. IEEE Commun Surv Tutor. 2019;21(2):1851–77. doi:10.1109/COMST.2019.2891891.
12. Khaleefa EJ, Abdulah DA. Concept and difficulties of advanced persistent threats (APT): survey. Int J Nonlinear Anal Appl. 2022;13(1):4037–52. doi:10.22075/ijnaa.2022.6230.

13. Xiao Q, Qiao Y, Zhen M, Chen S. Estimating the persistent spreads in high-speed networks. In: 2014 IEEE 22nd International Conference on Network Protocols; 2014 Oct 21–24; Raleigh, NC, USA. Piscataway, NJ, USA: IEEE; 2014. p. 131–42. doi:10.1109/ICNP.2014.33.
14. Rios VD, Inácio PR, Magoni D, Freire MM. Detection and mitigation of low-rate denial-of-service attacks: a survey. IEEE Access. 2022;10:76648–68. doi:10.1109/ACCESS.2022.3191430.
15. Dai H, Shahzad M, Liu AX, Li M, Zhong Y, Chen G. Identifying and estimating persistent items in data streams. IEEE/ACM Trans Netw. 2018;26(6):2429–42. doi:10.1109/TNET.2018.2865125.
16. Zhang Y, Li J, Lei Y, Yang T, Li Z, Zhang G, et al. On-off sketch: a fast and accurate sketch on persistence. Proc VLDB Endow. 2020;14(2):128–40. doi:10.14778/3425879.3425884.
17. Cormode G, Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. J Algor. 2005;55(1):58–75. doi:10.1016/j.jalgor.2003.12.001.
18. Yang T, Jiang J, Liu P, Huang Q, Gong J, Zhou Y, et al. Elastic sketch: adaptive and fast network-wide measurements. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication; 2018 Aug 20–25; Budapest, Hungary. New York, NY, USA: ACM; 2018. p. 561–75. doi:10.1007/s11704-016-6053-x.
19. Roy P, Khan A, Alonso G. Augmented sketch: faster and more accurate stream processing. In: SIGMOD '16: Proceedings of the 2016 International Conference on Management of Data; 2016 Jun 26–Jul 1; San Francisco, CA, USA. New York, NY, USA: ACM; 2016. p. 1449–63. doi:10.1145/2882903.2882948.
20. Estan C, Varghese G. New directions in traffic measurement and accounting. In: Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications; 2002 Aug 19–23; Pittsburgh, PA, USA. New York, NY, USA: ACM; 2002. p. 323–36. doi:10.1145/633025.633056.
21. Zhou Y, Yang T, Jiang J, Cui B, Yu M, Li X, et al. Cold filter: a meta-framework for faster and more accurate stream processing. In: Proceedings of the 2018 International Conference on Management of Data; 2018 Jun 10–15; Houston, TX, USA. New York, NY, USA: ACM; 2018. p. 741–56. doi:10.1145/3183713.3183726.
22. Yang T, Gong J, Zhang H, Zou L, Shi L, Li X. Heavyguardian: separate and guard hot items in data streams. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2018 Aug 19–23; London, UK. New York, NY, USA: ACM; 2018. p. 2584–93. doi:10.1145/3219819.3219978.
23. Lahiri B, Chandrashekar J, Tirthapura S. Space-efficient tracking of persistent items in a massive data stream. In: Proceedings of the 5th ACM International Conference on Distributed Event-Based System; 2011 Jul 11–15; New York, NY, USA. New York, NY, USA: ACM; 2011. p. 255–66. doi:10.1145/2002259.2002294.
24. Li W, Patras P. P-sketch: a fast and accurate sketch for persistent item lookup. IEEE/ACM Trans Netw. 2023;32(2):987–1002. doi:10.1109/TNET.2023.3306897.
25. Sharma A, Gupta BB, Singh AK, Saraswat VK. Advanced persistent threats (APT): evolution, anatomy, attribution and countermeasures. J Ambient Intellig Human Comput. 2023;14(7):9355–81.