applied sciences

MDPI

Article

MSLCFinder: An Algorithm in Limited Resources Environment for Finding Top-*k* Elephant Flows

Xianlong Dai [1,2,3], Guang Cheng [1,2,3,*], Ziyang Yu [1,2,3], Ruixing Zhu [1,2,3] and Yali Yuan [1,2,3]

[1] School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China
[2] Jiangsu Province Engineering Research Center of Security for Ubiquitous Network, Nanjing 211189, China
[3] Purple Mountain Laboratories, Nanjing 211189, China
* Correspondence: chengguang@seu.edu.cn

Featured Application: The results of this paper can be used in the fields related to network measurement, especially in the fields of network traffic sampling, network traffic measurement, and finding top-*k* elephant flows.

Abstract: Encrypted traffic accounts for 95% of the total traffic in the backbone network environment with Tbps bandwidth. As network traffic becomes more and more encrypted and link rates increase in modern networks, the measurement of encrypted traffic relies more on collecting and analyzing massive network traffic data that can be separated from the support of high-speed network traffic measurement technology. Finding top-*k* elephant flows is a critical task with many applications in congestion control, anomaly detection, and traffic engineering. Owing to this, designing accurate and fast algorithms for online identification of elephant flows becomes more and more challenging. Existing methods either use large-size counters, i.e., 20 bit, to prevent overflows when recording flow sizes or require significant space overhead to measure the sizes of all flows. Thus, we adopt a novel strategy, called *count-with-uth-level-sampling*,in this paper, to find top-*k* elephant flows in limited resource environments. Moreover, the proposed algorithm, called MSLCFinder, incurs lightweight counter and uth-level multi-sampling with small, constant processing for millions of flows. Experimental results show that MSLCFinder can achieve more than 97% precision with an extremely limited hardware resource. Compared to the state-of-the-art, our method realizes the statistics and filtering of millions of data streams with less memory.

Keywords: network measurement; top-*k* finding; elephant flow; MSLCFinder; data flow; traffic sampling; network security; massive traffic

Citation: Dai, X.; Cheng, G.; Yu, Z.; Zhu, R.; Yuan, Y. MSLCFinder: An Algorithm in Limited Resources Environment for Finding Top-*k* Elephant Flows. *Appl. Sci.* **2023**, *13*, 575. https://doi.org/10.3390/app13010575

Academic Editor: Rubén Usamentiaga

Received: 30 November 2022
Revised: 24 December 2022
Accepted: 27 December 2022
Published: 31 December 2022

1. Introduction

Since the late 1960s, following the ARPANET's birth and the development of network technologies over five decades, the Internet has achieved great success. According to the report [1], Internet users increased by 3.5 percent from October 2022, reaching 5.07 billion as we enter the year's final quarter. One hundred and seventy-one million new users over the past 12 months have taken global Internet penetration to 63.5%. Global mobile users have reached 5.48 billion, with smartphones accounting for almost four in five of the mobile handsets in use today, with 68.6% of all the people on Earth now using some form of mobile phone. The scale of global Internet users is unprecedented, the existing network traffic is characterized by encryption, and the network link has entered the era of high speed. The line rate in modern high-speed networks has reached hundreds of Gbps or multiple Tbps. Encrypted traffic accounts for 95% of the total traffic in the backbone network environment with Tbps bandwidth [2]. The different types of networks have mushroomed in our life based on preeminent network infrastructure, such as IoT, Internet of Vehicles, 5G, cloud computing, blockchain, satellite networks, etc. However, the network forms are diversified

and heterogeneous. The nodes at different levels in the end-edge-cloud architecture on the Internet are heterogeneous regarding function, performance, and data coverage. Thus, the security of cyberspace depends more on the traffic measurement of high-speed networks and encrypted traffic analysis and classification.

At this stage, the analysis, classification, and measurement of encrypted traffic rely more on the collection and analysis of massive network traffic data, and establishing more extensive and comprehensive encrypted traffic sample datasets is one of the long-term and primary goals of such work. The collection and analysis of massive network traffic data cannot happen without the support of high-speed network traffic measurement technology. Generally, the network traffic on the high-speed link is viewed as a set of flows. For traffic measurement, complete packet processing must occur in a few nanoseconds. For example, in a 100 Gbp Ethernet link, to process packets with an average size of 64 bytes and continuous arrival, the average processing time is about 4.768 ns. Thus, the measurement of network traffic has more technical challenges, such as using extremely high computing resources and storage resources, because of the continuous improvement of network link rate and the sharp increase of network data flows.

The flow is composed of the packets carrying the same FlowIDs such as source IP, destination IP, or others. We can define the size of flow as the number of packets in each flow. The network traffic conforms to the heavy-tailed distribution model in the real world: 80% of the network traffic consists of 20% of the flow, and the remaining 80% of the flow only constitutes 20% of the network traffic. In other words, compared with mouse flows (e.g., the number of packets less than 200), the elephant flows (e.g., the number of packets more than 10,000) can signify the real characteristics of the network. To provide high-quality operation services (such as QoS management and QoE improvement [3]), detect anomalies in the network, control network congestion, or detect DDoS attacks, the different statistics and measurements of flows should be collected and analyzed based on high-speed network traffic measurement technology by the Internet service providers (ISPs) or the department of cyberspace security supervisions.

Thus, finding top-k elephant flows and aggregating information about flow distributions can help us to achieve the above requirements. The significance of finding top-k flows is shown in Figure 1. In addition, the data flow has the characteristics of real time, continuity, and boundedness, which requires that the algorithm for processing the data flow can only perform calculations on the network flow once and can only use limited computing resources and memory resources. It also needs to ensure accuracy and timely awareness of large-scale data flows. Thus, finding top-k flows is challenging, especially with limited resources.

High-speed network traffic measurement technology can mainly solve this problem, a challenging research field. For traffic measurement in the high-speed network environment, there are generally three solutions [4]: (1) based on high-performance dedicated hardware (such as TCAM, ASIC), but high-performance hardware equipment is costly; (2) based on sampling technology; (3) based on data flow technology and method. The advantages and disadvantages of different solutions are shown in Figure 2.

The top-k elephant flow identification method based on sampling extracts some representative packets and then uses the probability theory to calculate the characteristics of the overall network traffic. The most typical method is the Sampled NetFlow [5] proposed and used by Cisco. In high-speed network links, the processing speed of hardware cannot match the enormous data flow, so the $1/N$ sampling method is used to relieve the pressure of elephant flow identification caused by high-speed network links. In the router, the flow memory will create a new flow record based on the five-tuples of each arriving packet. Whenever a new flow is drawn into the router, a new flow record will be created in the flow memory, and a quintuple will be extracted as the unique identification of the flow. Different from NetFlow, the sample and hold [6] method first queries whether the currently arrived packet exists in the flow memory and, if so, updates the flow record; otherwise, the data packet is sampled with probability p, and a piece of new flow information is created

in the flow memory after being sampled. However, the accuracy of these two elephant flow recognition methods is coarse granularity, depending on the flow memory size. They are unsuitable for elephant flow recognition in high-speed networks.

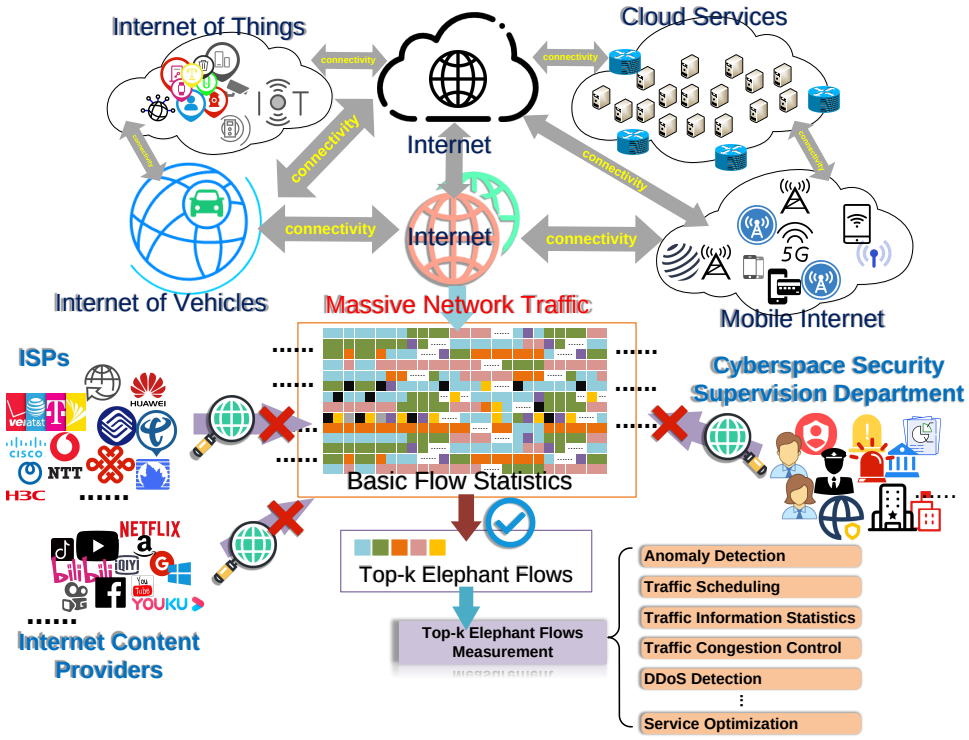


Figure 1. The significance of top-*k* flows measurement in networks.

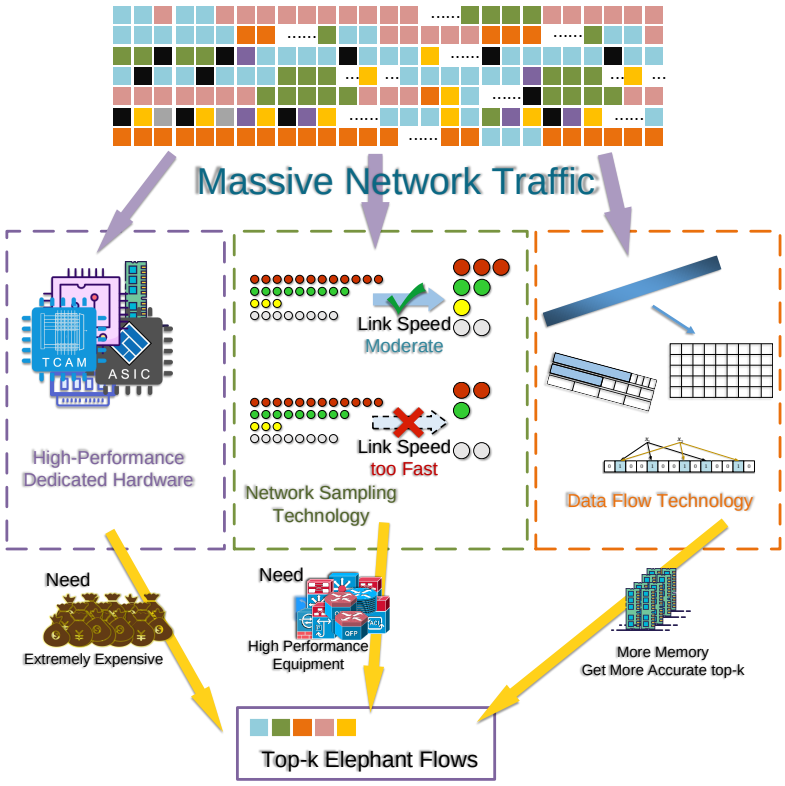


Figure 2. Advantages and disadvantages of different solutions in finding top-*k* elephant flows.

Based on data flow technology, the top-*k* elephant flow identification method has two basic strategies: *count-all* and *admit-all-count-some* [7]. The similarity between the two strategies is that they handle all traffic in the network rather than sampling. Therefore, the error of these two methods is smaller than the sampling technique; however, the two strategies are different in the process of handling traffic. The *admit-all-count-some* strategy handles all traffic but does not store all traffic. The main idea is to recognize all new flows and, at the same time, remove the smallest flow from the cache. Examples are frequent [8], efficient counting [9], lossy counting [10], space-saving [11], and CSS [12]. They can effectively eliminate the flow by introducing some constraint rules and controlling the error of top-*k* flow measurement within a certain range. The *count-all* strategy is based on sketches such as count–min sketch [13] and count sketch [14] to measure the size of all flows. It stores and counts all traffic. It is necessary to use sketches to summarize and count data flows to reduce the demand for memory space and resources on the premise of storing all traffic.

Furthermore, the *admit-all-count-some* strategy assumes that every new incoming flow is an elephant flow and expels the smallest one, in summary, to make room for the new one. Nevertheless, most flows are mouse flows. Such an assumption causes a significant error, especially under tight memory. The *count-all* strategy needs a large sketch to count all flows. These solutions could not be more memory-efficient. Additionally, it needs more memory to scan the entire counter and sort elements to answer the top-*k* query. On the other hand, because the flow sizes are unknown a priori, almost existing methods set a large length (such as direct use of the integers in C++ or a 20-bit counter).

However, if the hardware development resources and costs are limited, for example, they use only 1 Mbit to complete the top-*k* measurement task of millions of flows. Using a large counter such as existing methods will not only lead to the waste of counter space caused by counting most flows whose sizes are under 200 (the mouse flows) but also cause fewer counters, affecting measurement accuracy. Therefore, existing top-*k* finding solutions all share the same limitation: most counters will only record a mouse flow and waste significant on-chip memory, especially with resource constraints.

On the whole, we aim to conquer six challenges of finding top-*k* in a resource-constrained environment: (1) the scheme should be completed within limited on-chip resources as far as possible; (2) the algorithm that needs to process data flows can only perform calculations on network flows once; (3) the calculation logic is simple and the storage space is small; (4) to achieve measurement of millions or more flows; (5) to restore valid top-*k* elephant flows information; (6) measuring top-*k* elephant flow on the fixed and short-length counter, with satisfactory accuracy.

In this paper, we adopt a novel strategy called *count-with-uth-level-sampling*. Moreover, we propose a novel core component named Multi-Sampled Lightweight Counting Finder (MSLCFinder) to achieve a measurement scheme for finding top-*k* flows based on hardware resource-constrained environments. The MSLCFinder has three primary modules: a lightweight counting module, a multi-sampling module, and a flow label recording module. The lightweight counting module can work under extreme storage requirements and requests a few bits for counting. The multi-sampling module can sample flows by using multi-sampling probabilities generated by itself and the lightweight counting module. They complement and constrain each other to maximize the optimization of the MSLCFinder in the measurement task. The flow label recording module can promptly update and record top-*k* elephant flows information. Finally, with the MSLCFinder we propose, we can effectively find top-*k* elephant flows with less memory overhead and counters under resource constraints.

Our main contributions can be summarized as follows:

- We propose a novel method for finding top-*k* elephant flows with little memory, reducing the waste of memory space of mouse flows. It is different from the two strategies in the previous study. It compresses the count by sampling and can accurately obtain the top-*k* elephant flows measurement results.

- We design a uth-level sampling algorithm. It can generate multiple different sampling probabilities by initial sampling threshold θ, size of MSLCFinder counter, and the sampling level u. It is used to ensure that the counters of MSLCFinder can measure elephant flows more optimally before the end of the measurement task and reduce the space occupied by mouse flows.
- We verify the effectiveness of the method in this paper on CAIDA public datasets and the effectiveness of the method in real network traffic.
- We propose a flow recording algorithm for MSLCFinder to make up for the defect that existing top-k finding solutions generally cannot restore stream ID and other information.

The rest of the paper is structured as follows. Section 2 describes the background of this research and reviews the related work. Section 3 focuses on the measurement method we proposed, including the design of MSLCFinder and the algorithm of its core components. In Section 4, the MSLCFinder is evaluated through experiments in many aspects. Finally, we summarize our research and point out future work.

2. Background and Related Work

Finding top-k elephant flows is quite challenging and complicated. Extremely high line rates of modern networks make it practically impossible to track all flow information accurately. The basis of finding top-k elephant flows depends on high-speed network traffic measurement technology and principle, and the core difficulties are based on hash function choice, data flow structure design, high-speed flow sampling technology, and flow information recording technology. Therefore, we first summarize and analyze the background methods in high-speed network traffic measurement. In this section, we also summarize the existing research progress of finding top-k elephant flows.

2.1. Related Backgrounds

2.1.1. Hash Algorithm

In order to solve the problems with computing resources and high-speed network traffic, it is necessary to deal with the network traffic by some measuring technologies, such as sampling measurement and load balance, etc., while the hash algorithm is one of the key measuring technologies [15]. Hash algorithms are mainly used for sampling stream records and checking the existence of network data flow. Commonly used hashes include MD5, CRC32 [16], IPSX [15], BobHash [16], XOR_SHIFT [17], and 2-universal hash [18]. Unlike the security considerations in the encryption field, the key to high-speed network measurement is the randomness of the hash value rather than the security. Therefore, the hash algorithm similar to the MD5 algorithm does not apply to this problem.

It can be divided into two types of methods. One is to directly use the identification field in the message [19]. This method is efficient, but it is difficult to ensure sufficient randomness of the hash value; the other is to use hash functions to calculate hash values [20]. The selection of hash functions is a crucial factor in ensuring the randomness of hash values and the efficiency of algorithms. G.C. et al. [15] proposed a random measure to evaluate the performance of the hash algorithm and theoretically proved that the exclusive OR operation and displacement operation between bits can improve the random characteristics of the hash value. Finally, they proposed the principles of the hash algorithm between bit streams, designed the characteristics of the 4-tuple (source I.P., destination I.P., source port, and destination port) based on I.P. packets, and proposed the IPSX algorithm. Through a large number of experiments, XOR has a high hash performance. The bit stream hash algorithm based on XOR and shift principle has good performance in terms of execution efficiency and uniformity of hash value, which can meet the requirements of high-speed network traffic measurement.

2.1.2. Network Traffic Sampling Methods

With the improvement of link rate and the diversification of applications, the vast network traffic has brought tremendous pressure to traffic collection, transmission, storage,

and analysis. In order to solve the problem of passive measurement in high-speed networks, sampling technology is applied to high-speed network traffic measurement, which can reduce the amount of data used for measurement, storage, and processing under the condition of meeting the statistical accuracy of the problem. The main idea of traffic sampling technology is to select a representative packet subset from the original traffic data and then infer the characteristics of the original traffic data through the packet subset. In high-speed network traffic measurement, the implementation of sampling methods is limited by technology and resources, and it is often necessary to compromise between sampling rate and estimation accuracy. The sampling collection dramatically reduces the processing load of the system and has good scalability. It can also reflect the original flow characteristic parameters from the sample characteristic parameters and has a specific measurement accuracy. For the traffic on the Internet, from the level of packets and flows, the sampling methods are mainly divided into packet sampling and flow sampling.

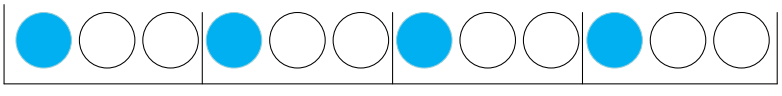
- Packet Sampling

Packet sampling refers to sampling the packets that make up the network traffic. Each packet is independent and does not consider the correlation between the packets. Standard packet sampling methods include systematic, simple, and stratified random sampling.

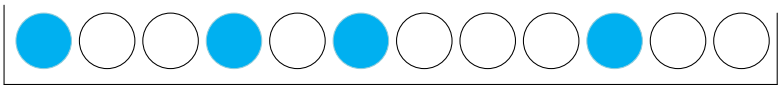
Systematic sampling refers to extracting objects at fixed intervals. After selecting the first object, select the next for every N object, as shown in Figure 3a. Systematic sampling is a widely used sampling method, but systematic sampling has a certain periodicity.

Simple random sampling refers to sampling objects with a certain probability, as shown in Figure 3b. The sampling probability of each object can be the same or different, and this probability generally follows a certain probability distribution function. In flow measurement, simple random sampling and random incremental sampling are commonly used in random sampling methods. These two random incremental sampling methods can avoid the synchronization problem of systematic sampling.

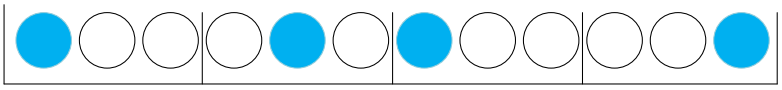
Stratified random sampling refers to dividing the population into several levels or type groups and then sampling from each level according to a certain proportion. This layering can be divided according to the arrangement order of elements, as shown in Figure 3c, or it can be layered according to some characteristics of elements (such as packet length, protocol type), and then sampled separately. The commonly used stratified sampling in network traffic measurement is uniform stratified random sampling. This method can ensure that the sampling is unbiased relative to the element's attribute, reduce the error of grouping statistics, and make the estimation result closer to the original data.



(a) systematic sampling



(b) simple random sampling



(c) stratified random sampling

Figure 3. Three packet sampling methods. (**a**) Description of what is systematic sampling. (**b**) Description of what is simple random sampling. (**c**) Description of what is stratified random sampling.

Due to the self-similarity of network traffic, He et al. [21] proposed an improved systematic sampling BSS (systematic biased sampling). Compared with systematic static and simple random sampling, it improves the estimation accuracy of the mean value and reduces the sampling cost. However, most packet sampling methods uniformly select packets without considering the size of the packets, which makes the network measurement obtain some uncertainties. Raspall [22] proposed EBS (efficient byte sampling) to improve measurement quality, make measurement accuracy less dependent on the traffic characteristics, and reduce measurement overhead.

- Flow Sampling

Flow sampling refers to the sampling of network flows within the measurement time. The packets that constitute network flows are not isolated. They are generated to complete specific applications. There is a certain correlation between them. Flow is a way to reflect this correlation. There are two main sampling methods for flow sampling: (1) sampling the packets first and then merging the packets into the flows; (2) the packets are merged into flows first, and then the sample the flows.

The statistical characteristics of flow and packet are entirely different, and the requirements of flow sampling and packet sampling are also different. Because the transmission technology limits the size of the packet, its maximum length will not exceed the maximum value the network can support, but the size of the flow is not affected. For network traffic measurement, which measurement and sampling method to use is determined by the purpose of the network measurement.

Sampling is one of the most widely used methods to reduce memory consumption and packet processing time. There are some problems in the static selection of sampling rate because the worst-case resource use is multiple orders of magnitude of average resource use. Sampled NetFlow [5] needs to configure a fixed sampling rate. This method's main problem is selecting safe parameters to ensure that the network device can continue operating under adverse traffic conditions. Therefore, the sampling rate is set in the worst case. Many researchers have solved the problem of dynamic sampling rate selection and overcome the defect of setting static sampling rate through an adaptive network environment, such as Adaptive NetFlow(ANF) [5]. However, adaptive sampling methods often consume many CPU resources and rely on complex data structures and algorithms, which hinder their implementation in network hardware.

Yang Du et al. [23] presented a framework to sample each flow with a probability adapted to flow size/spread to achieve self-adaptive sampling. They proposed two algorithms, SAS-LC and SAS-LOG, geared towards per-flow spread estimation and per-flow size estimation using different compression functions. Sanjuàs-Cuxart et al. [24] proposed a measurement method based on adaptive flow sampling, named Cuckoo sampling. The algorithm is based on a simple and random data structure, which requires tiny packet overhead and is easy to parameterize. Compared with previous methods, this algorithm is based on simpler algorithms and requires fewer hardware resources, so it is suitable for hardware implementation. In order to reduce the estimation error of mouse flows, probability counter update algorithms are proposed, such as ANLS [25]. These algorithms use the function $p(c)$ of counter value c to replace the static sampling rate p, so the sampling rate varies according to the number of packets sampled. However, because the measurement accuracy is affected not only by the sampling function but also by the flow length distribution, it is not enough to choose an independent static sampling function. Therefore, Ma et al. [26] proposed the Smart Selection Sampling (S3) method. By using the flow length distribution information to select an appropriate sampling function, the sampling function can be adjusted to obtain higher measurement accuracy.

2.1.3. Network Data Flow Methods

With the rapid development of hardware, it is possible to measure the real-time performance of high-speed networks. The data flow methods are real-time measurement methods that use limited computing and memory resources to perform a calculation on the

network flow. Sampling technology is widely used in network traffic measurement and analysis. Although the sampling method produces a representative subset of the original data, the statistical information of network traffic inferred from the sampling data has some errors and cannot accurately reflect the characteristics of the original traffic. The data stream method has the characteristics of single-pass scanning, limited computing, and memory resources. It is a powerful method for high-speed network traffic measurement.

A practical and available data structure is a crucial prerequisite of the data flow method in network traffic measurement. The optimized data structure helps to improve the execution efficiency and estimation accuracy of the algorithm and reduces the computational and storage costs. The most representative and universal data structures are bitmap [27], bloom filter [28], and sketch [13].

Bitmap [27] is a simple data structure. Its essence is a bit group. It maps a field to a bit group to count the number of duplicate elements. The direct bitmap is a stream number estimation algorithm that uses the hash function to map the stream ID to a bit in the bitmap. The bitmap is initialized to 0. When a packet arrives, its ID will be mapped a bit in bitmap, and the bit will be set to 1. All packets belonging to the same class are mapped to the exact location in bitmap. Therefore, no matter how many packets are sent in each stream, each stream corresponds to at most one bit in bitmap.

Bloom filter [28] can be seen as an extension of bitmap. It is a simple, efficient, probability-based random data structure, and its primary data structure is a bit vector. Bloom filter can allow for processing more data under inevitable error and then judging whether the mapping is repeated. Bloom filter can generally judge whether an element is in the collection. It is characterized by fast operation and small memory consumption, but it can only tell us whether an element is "definitely not in the collection" or "probably in the collection". Bloom filter is a widely used tool based on multi-hash. It combines multiple hash functions to perform the hash process and verification and uses multiple hash functions to replace a single hash function to improve the accuracy of hash verification. Various network systems have used bloom filter algorithms, such as Web proxy and cache, database servers, and routers.

Sketch [13] is a type of sublinear space and probabilistic data structure widely used in network measurement. Sketch algorithms often use probabilistic methods such as hash functions to map elements to continuous memory space and achieve small space consumption and swift constant-level processing time by sacrificing sure accuracy. Among them, the count–min sketch [13] is a classic sketch algorithm. Count–min sketch allows basic queries in the data flow summary, such as point queries, range queries, and inner product queries. It can also be used to solve significant problems in the data flow, such as finding quantiles and identifying elephant flows. The essence of sketch is also a data structure based on multiple hashes. It maintains an independent hash storage space for each hash function and finds data meeting special requirements through a series of complex and unique mapping rules and several additional rules. The most significant difference from bloom filter is that sketch is based on the time series model, and the counters in the hash space can be reduced.

2.2. Review of Related Research

2.2.1. Network Measurement Technology Based on Sketch

Count–min sketch (CM sketch) [13] has the advantages of simple, efficient operation and small memory usage with a high accuracy guarantee. CM sketch has linear space–time complexity and can perform frequency statistics of data streams, frequent item mining, and other tasks. It can overestimate but only requires one memory access per counter. Currently, CM sketch is one of the most popular sketch-based algorithms for network measurement.

Although many sketch structures have high accuracy, they can only perform specific detection tasks. For example, FMSketch [29] can perform the measurement task of cardinality estimation. OpenSketch [30] is a universal and efficient measurement framework that can measure a variety of tasks in real-time elephant flow detection, DDoS/SuperSpreader

network anomaly detection, flow size statistics, etc. It is also deployed in SDN. SketchVisor [31] uses fast path technology and compressive sensing technology to reduce the number of packets to be processed, improve the processing speed of packets, and realize the statistics of multiple traffic characteristics. To solve the problem of complex resource configuration in the measurement process, SketchLearn[32] designed a multi-level sketch structure. According to the feature that small data streams obey the Gaussian distribution in multi-level sketches, the algorithm automatically infers and extracts large data streams by using the statistical characteristics of data streams, effectively reducing system overhead. NitroSketch [33] uses adaptive sampling to update only the values of some counters, reducing the number of data packets to be processed, fundamentally improving the robustness of the measurement framework, and significantly speeding up the processing of data packets.

Relevant research [34] shows that the combination of sketch structure and machine learning algorithm can dynamically adapt to changes in the network environment, automatically extract network features, and achieve high-precision network measurement. In Jiang et al. [34], measurement accuracy was dramatically improved. Zhou et al. [35] combined the skeleton structure with reinforcement learning and proposed the RL sketch method to detect abnormal data streams in the network. RL sketch uses DQN (Deep Q-Network) to predict the size of data streams, filter out mouse flows, and only count elephant flows in the skeleton. High-accuracy network measurement is achieved with small memory. Fu et al. [36] used the K-means algorithm to map similar data streams to the same bucket for clustering, replacing individual values with average category values, and used random mappers to correct statistical values, which improved processing speed while ensuring accuracy.

In a word, the research of the sketch method can be roughly divided into two directions: (1) the transformation from poor scalability of the single task to strong versatility of multi-task; (2) it combines machine learning and artificial intelligence algorithms to adapt to the diversity of a network environment.

2.2.2. Top-*k* Flows Finding Technology

Although the probability of top-*k* comes from the discovery of frequent items in massive data in the database research field, the top-*k* lookup in the network environment differs from the task in the database. The top-*k* elephant flow identification method has two basic strategies: *count-all* and *admit-all-count-some* [7], but the core problem is still elephant flow detection, including heavy hitter [37] detection and heavy change [37,38] detection. Table 1 summarizes the research on top-*k* elephant flow identification methods in related work.

Due to memory limitation, hash conflicts will inevitably occur in processing massive data, making the measured value more extensive than the actual value. Research has found that the statistical error in the sketch can be effectively reduced by separating the elephant and mouse flows.

Cold Filter [39] has designed a two-layer sketch structure. When data packets arrive, it first uses the first layer of the sketch to filter out most of the mouse flows, and then sends the filtered elephant flows to the second layer of the sketch for storage. This classified statistical method reduces the super error between elephant flows and mouse flows and also reduces the false positives in the statistical results. HeavyKeeper [7] uses the attenuation formula to make the elephant flows decay slowly, and the mouse flows violently decay to screen out the elephant flows. HeavyGuardian [40] divides each bucket in the hash table into a heavy part and a light part. The heavy part stores key-value pairs of multiple elephant flows, and the light part only stores the count values of mouse flows. High measurement accuracy is achieved with limited memory. The ActiveKeeper [41] uses a two-mode active counter to record flow sizes, reducing memory usage while ensuring high accuracy. ActiveKeeper also avoids the waste of mouse flow on the counter. The FastKeeper [42] can identify large real-time flows with a primary goal of simultaneously achieving low overhead, high performance, and high accuracy. FastKeeper employs a sliding-window-based algorithm

for accurate measurement of real-time flow rates and a bitmap-voting algorithm for the timely replacement of flows that have become small in the measurement data structure. Xiao et al. [43] presented a universal online lightweight sketch named ActiveCM + for tracking heavy hitters, which ensures that per-packet overhead is a small constant (four hash and four memory accesses) in the worst case, making it much more suitable for online operations, especially for pipeline implementation.

Table 1. Research on top-*k* elephant flow identification methods.

Research Category	Strategy	References	Method
Based on Data Flow	*count-all*	CM sketch [13]	sketch & min-heap
		Count sketch [14]	sketch & min-heap
	admit-all-count-some	Frequent [8]	constraint rules
		Efficient counting [9]	constraint rules
		Lossy counting [10]	error variable
		SpaceSaving [11]	stream-summary
		CSS [12]	compact space-saving & sliding window
	separate elephant/mouse flows	HeavyKeeper [7]	count-with-exponential-decay
		Cold Filter [39]	two-layer sketch structure
		HeavyGuardian [40]	heavy and light part
		ActiveKeeper [41]	two-mode active counter
		FastKeeper [42]	sliding window & bitmap voting
		ActiveCM + [43]	ActiveCM + & LUS
Based on Sampling		Sampled NetFlow [5]	sampling⇒recording
		Sample and hold [6]	query⇒sampling⇒recording

In summary, those researchers mainly studied the method of separating elephant and mouse flows to optimize the counters counting or reduce the collision probability of the hash function.

3. Top-*k* Measurement Scheme Based on MSLCFinder

In this section, we make a systematic expound measurement algorithm based on MSLCFinder, a novel and efficient scheme for finding top-*k* flows in a limited resource environment. The measurement scheme overview is, understandably, described first. Then, we focus on three core module elements: *lightweight counting module*, *uth-level multi-sampling module*, and *flow label recording module*. They are the three core components of MSLCFinder, and they interact and restrict each other to complete the measurement task together.

3.1. Problem Definition

Finding the top-*k* elephant flows task is finding the k flows with the largest size with some measures of the measurement task. Typically, the network traffic is a set of N flows $F = \{f_1, f_2, ..., f_N\}$, where each flow $f_i (i = 1, 2, ..., N)$ is composed of a set of packets $\{Pac_1, Pac_2, ..., Pac_n\}$ with the sameFlowID such as network quintuple. With this measure, the number of packets, i.e., n_pac of each flow, can be defined as the flow size. We let the n_pac_i be the real size of the flow f_i and the $\widehat{n\_pac}_i$ is the estimated size of f_i. Thus, finding top-*k* elephant flows can be described as outputtinga set of $Top - k = \{(\widehat{f}_1, \widehat{n\_pac_1}), (\widehat{f}_2, \widehat{n\_pac_2}), ..., (\widehat{f}_1, \widehat{n\_pac_k})\}$. The set Top-*k* with *k* elements can track the top-*k* flows with their estimated sizes.

3.2. Measurement Scheme Overview

The overall design of the measurement scheme is shown in Figure 4. The scheme parses each message arriving by the traffic to be measured packet by packet, calculates the hash value corresponding to the data flow ID through the hash function, and is used

to query, judge, and update the information of the MSLCFinder and flow sampling. The sampling probability is calculated by the given threshold and updated by the size of the counter in MSLCFinder.

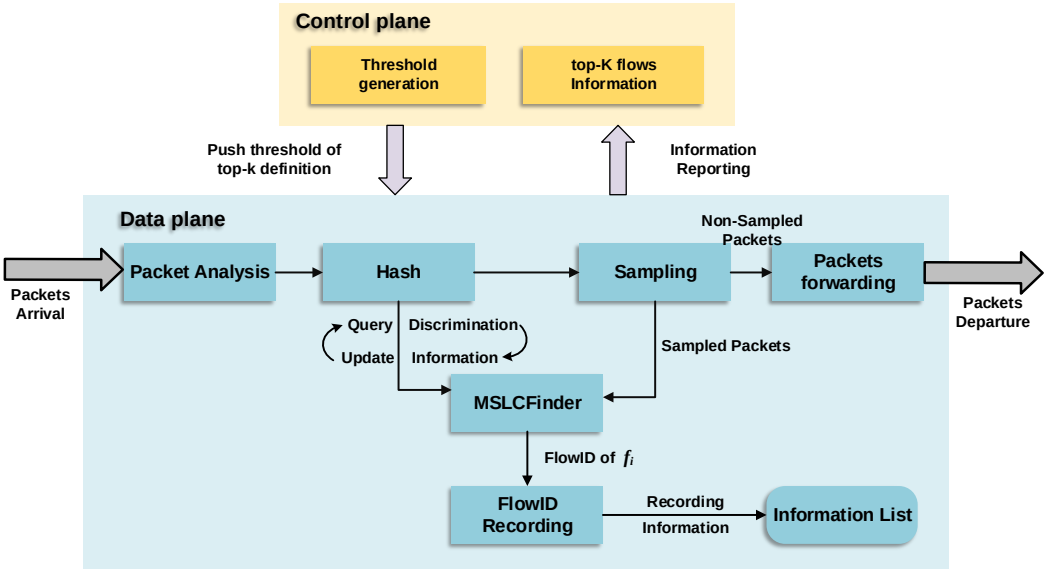


Figure 4. The scheme overview of top-*k* flows measurement task in networks based on MSLCFinder.

At the beginning of the measurement, the hash value is calculated for the parsed FlowID, and the sampling process will use the initial sampling probability. If the FlowID is sampled, the counter at the corresponding location of the FlowID in MSLCFinder will be updated. Otherwise, the counter is not updated to reduce the probability of counting small flows in the MSLCFinder structure and reduce the pressure on on-chip resource storage.

Here, the size of the counter in the MSLCFinder is a fixed n bit. It uses a uth-level multi-sampling module to sample the flow. When the value of the MSLCFinder counter exceeds u thresholds, the sampling probability is recalculated, respectively, and a new sampling probability resamples the arriving flow. If the sampling is successful, the counter is incremented by 1; otherwise, it will be forwarded directly. When the count exceeds the last-level threshold, the flow corresponding to the location flow ID in the MSLCFinder is the top-*k* flow. The flow information is stored in the information storage table through the flow label recording algorithm. At the end of the measurement cycle, the entire information storage table is reported to the control end to obtain the specific information of the top-*k* flow.

3.3. Multi-Sampling Lightweight Counting Finder

3.3.1. Rationale

Our goal is to find top-*k* elephant flow with resource-constrained environments. It requires us to use less memory to complete the measurement task of more than one million flows. We aim to use a small hash table to find it, and it does not require highly accurate estimates. We are committed to using smaller length counters to be competent for the challenging task; for example, the counter length is less than 8 bits, or even smaller. Thus, before the counter increases the count, it is necessary to perform flow sampling by some regulars for the arriving traffic to determine whether the current packet can increase the counter's value at the corresponding location in the hash table. In order to solve minimizing the error of flow sizes, we use multiple hash tables with different hash functions. The MSLCFinder has three primary modules: a lightweight counting module, a multi-sampling module, and a flow label recording module. We designed the MSLCFinder lightweight counting module based on the basic principle of sketches and smaller counter lengths.

The design of the multi-sampling module will use the counts in counters and the initial sampling threshold. Based on the min-heap, we designed the flow label recording module.

3.3.2. The Lightweight Counting Module

As shown in Figure 5, the lightweight counting module has d hash functions, and each function space comprises w counters. The counter of MSLCFinder is only n bits long. The hash functions can calculate the hash value of the incoming packet as the FlowID for MSLCFinder counting. It can also help the multi-sampling module to achieve flow sampling.

Multi-Sampling: Multi-sampling is a core component module in MSLCFinder, which is responsible for sampling the arrived packets, thus telling MSLCFinder whether to add the current packet to the counter at the corresponding location. Unlike the count–min sketch, the count–min sketch uses a more extended counter to record the information of all streams and count each stream. Because our method needs to complete the top-k measurement task under resource constraints, sampling and judging the traffic before updating the counter can significantly reduce the counting pressure of the counter.

Insertion: At the beginning of the measurement task, all counters are zero. For each coming packet Pac_ϑ belonging to flow f_φ, MSLCFinder calculates the d hash functions. After mapping f_φ to the counter node, it should be sampled by sampling probability and the hash value. The hash value of Pac_ϑ will be the FlowID of flow f_φ. If Pac_ϑ is sampled, the counter will addone. Otherwise, the counter will not be processed.

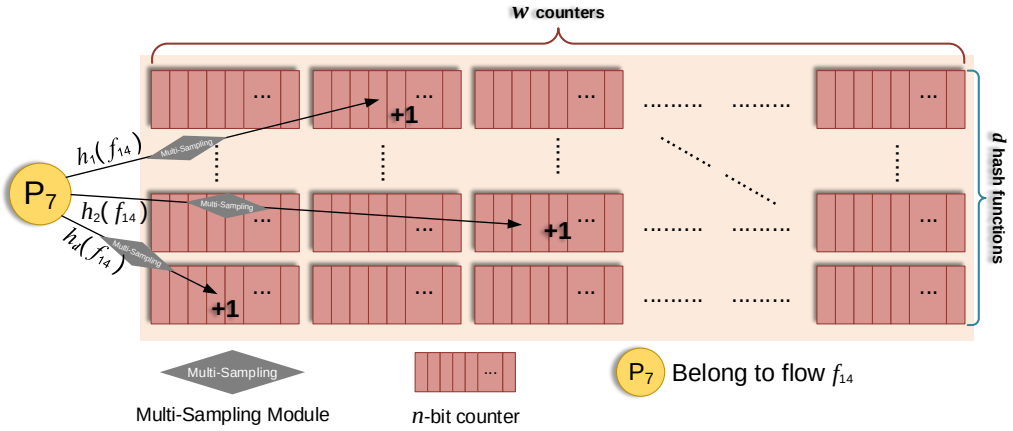


Figure 5. The structure of the lightweight counting module in MSLCFinder.

Query: To query the counts of a flow f_φ, MSLCFinder reports the minimum value of d counter nodes that f_φ mapped.

Analysis: The minimum error tolerated for the measurement task is δ. The flow f_φ true size is n_pac_φ, and its estimated size is $\widehat{n\_pac_i}$. We want to set an error range ($n\_pac_\varphi \leq \widehat{n\_pac_i} \leq n\_pac_\varphi + \varepsilon N$), in which N is the total number of packets counted by the module. In the optimal case, $w = \lceil e/\varepsilon \rceil$, $d = \lceil \ln(1/\delta) \rceil$ (where e is the base of the natural logarithm, i.e., 2.71828..., a constant chosen to optimize the space for fixed accuracy requirements). When ε is smaller (i.e., w is larger), the error of element estimation is smaller; when δ is smaller (i.e., d is smaller), the probability of error in element estimation is smaller.

For the insertion procedure, because computing each hash function takes $O(1)$ (constant) time, the total time to perform an update is $O(d)$, independent of w. Since d is typically small in practice (often less than 10), updates can be processed at high speed. The space complexity is $O(wd)$. For the query procedure, the time complexity is $O(\ln(1/\delta))$, and the space complexity is $O((1/\delta^2)\log(1/\delta))$.

3.3.3. The Multi-Sampling Module

Multi-sampling is a core component module in MSLCFinder, which is responsible for sampling the arrived packets, thus telling MSLCFinder whether to add the current packet to the counter at the corresponding location. Unlike the count–min sketch, the count–min sketch uses a more extended counter to record the information of all streams and count each stream. Because our method needs to complete the top-k measurement task under resource constraints, sampling and judging the traffic before updating the counter can significantly reduce the counting pressure of the counter.

The multi-sampling module has two functions: (1) to conduct flow-based sampling for consecutive arriving packets to guide MSLCFinder on whether to count this packet; (2) to calculate the flow sampling probability corresponding to the FlowID based on the current number of packets in the MSLCFinder.

The multi-sampling module can calculate the sampling probability p according to the current value in the counter and guide the MSLCFinder on whether to process or directly forward the currently arrived packets. The sampling probability p needs to obtain the count in counter and the initial threshold θ. Let a be the counts of the single counter. We choose uth-level sampling. The calculation method of sampling probability p is as follows:

- Initial sampling probability:

$$q_1 = \theta - [(2^{n-u+1} - 1) - 1] = \theta - [2^{n-u+1} - 2] \tag{1}$$

$$p_1 = 1/q_1 \tag{2}$$

- When a exceedsthe $2^{n-u+1} - 1$, calculate 1st-level sampling probability:

$$q_2 = [(2^{n-u+2} - 2) - (2^{n-u+1} - 2)] + q_1 = q_1 + 2^{n-u+1} \tag{3}$$

$$p_2 = 1/q_2 \tag{4}$$

- When a exceedsthe $2^{n-u+i} - 1 (i = 0 \sim u - 1)$, calculate $i + 1$-level sampling probability:

$$q_{i+1} = [(2^{n-u+i+1} - 2) - (2^{n-u+i} - 2)] + q_i = q_i + 2^{n-u+i} \tag{5}$$

$$p_{i+1} = 1/q_{i+1} \tag{6}$$

The uth-level ($u < n - 1$) sampling method can generate u probabilities for sampling. When the measurement task is over, the computational function $Esti(a, q_1, q_2, ..., q_u)$ with count a of counter can obtain estimates.

- When $1 \leq a < 2^{n-u+1}$:

$$Esti(a, q_1, q_2, ..., q_u) = (a - 1) \cdot q_1 \tag{7}$$

- When $2^{n-u+1} \leq a < 2^{n-u+2}$:

$$Esti(a, q_1, q_2, ..., q_u) = (2^{n-u+1} - 1) \cdot q_1 + (a - 2^{n-u+1}) \cdot q_2 \tag{8}$$

- When $2^{n-u+i} \leq a < 2^{n-u+i+1}$ $(2 \leq i \leq u - 1)$:

$$Esti(a, q_1, q_2, ..., q_u) = (2^{n-u+1} - 1) \cdot q_1 + \sum_{j=1}^{i-1} (q_{j+1} \cdot 2^{n-u+j}) + (a - 2^{n-u+i}) \cdot q_{i+1} \tag{9}$$

- We can use the a to calculate the i by using the following method:

$$i = \lfloor \log_2 a \rfloor + u - n \tag{10}$$

For example, when we use the 3rd-level sampling method to count the array named B, which has 2 million elements, let the threshold θ be 10,000, and the counter Co has 8-bit

length. According to the above formula, $u = 3, q_1 = 9938, q_2 = 10,002$, and $q_3 = 10,130$. When the subscript of the element in array B is sampled, the value of counter Co is plus 1. When the array traversal is over and the $Co = 200$, the actual estimated value is 1,995,582 according to the 3rd-level sampling method, which proves that our sampling plan is highly deterministic.

When the measurement task starts, the probability p_1 is directly used to sample each flow. If it is selected, it will be mapped to the corresponding position of the ID in the lightweight counting module, and its count will be increased by 1; if the packet is not sampled, it will be forwarded directly, and MSLCFinder will not process it. Here, we take the flow f_φ as an example. When the packet's counts in f_φ exceed the threshold value of the second level, we start to calculate the sampling probability p_2 of the second level and continue to process the arrived packets according to p_2, and so on until the threshold value of the u-level is exceeded. When, and only when, the count of f_φ reaches $2^n - 1$, the packets corresponding to the FlowID will not be processed anymore, and the FlowID will be regarded as a top-k elephant flow.

3.3.4. The Flow Label Recording Module

Figure 6 shows that the flow label recording module is designed to record and find top-k elephant flows. The MSLCFinder achieves this goal based on a min-heap data structure.

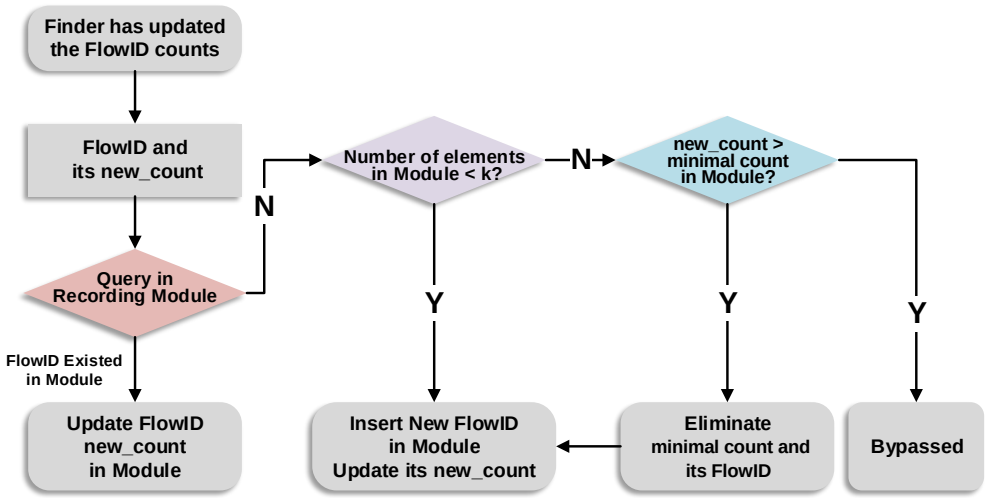


Figure 6. The structure of the flow label recording module in MSLCFinder.

Algorithm 1 shows the flow label recording algorithm. For each packet Pac_τ belonging to flow f_ϑ, which is counted by MSLCFinder, the module first inserts it into itself. Suppose that MSLCFinder reports the size of f_ϑ as $\widehat{n\_pac}_\vartheta$. If f_ϑ is already in the module, it will update the estimated flow size with $\max(\widehat{n\_pac}_\vartheta, min\_heap[f_\vartheta])$, where $min\_heap[f_\vartheta]$ is the record of the size of f_ϑ in the module. Alternatively, determine whether the module's number of elements is fewer than k of the top-k. If the number of elements is fewer than k of the top-k, insert it into the module. Otherwise, judge whether the new count of the flow is greater than the current minimum in the module. If so, eliminate the minimum in the module, and insert the ID information of the new flow and its estimated count. To query top-k flows, we simply report the $\hat{k}$ IDs of the k flows in the module. If the measurement task wants to obtain the estimated flow size of these k flows, we let the module record and output this information; otherwise, it is not necessary to know. The FlowID of top-k flows is the top target of MSLCFinder.

Analysis: (1) The time complexity: when a new element comes, it needs $O(\log k)$ time to query or update the min-heap in the flow label recording module of MSLCFinder. Therefore, the total time complexity is $O(n \log k)$. (2) The space complexity: the flow label recording module requires $O(k)$ space.

Algorithm 1: Flow label recording algorithm.

```
Function Record(Flow.ID, new_cnt):
    if Flow.ID ∈ Module then
        Flow.ID.cnt ← new_cnt;
    else if module.elements.number < k then
        Insert Flow.ID;
        Flow.ID.cnt ← new_cnt;
    end
    else
        if new_cnt > Modsule.min_cnt then
            Delete Module.min_cnt;
            Delete Module.min_cnt.FlowID;
            Insert Flow.ID;
            Flow.ID.cnt ← new_cnt;
        end
    end
```

3.4. Description of Top-k Measurement Algorithm Based on MSLCFinder

Figure 7 shows the overall algorithm process, divided into two stages: measurement and FlowID recording. Low-proportion sampling reduces the probability of mouse flows being counted by MSLCFinder. It reduces operations, saves the MSLCFinder space, and reduces the probability of hash collisions. The overall algorithm can only access the memory in four cases: no access, one read zero write, one read one write, and two read one write, which meets the requirement of this problem that one read one write is the average.

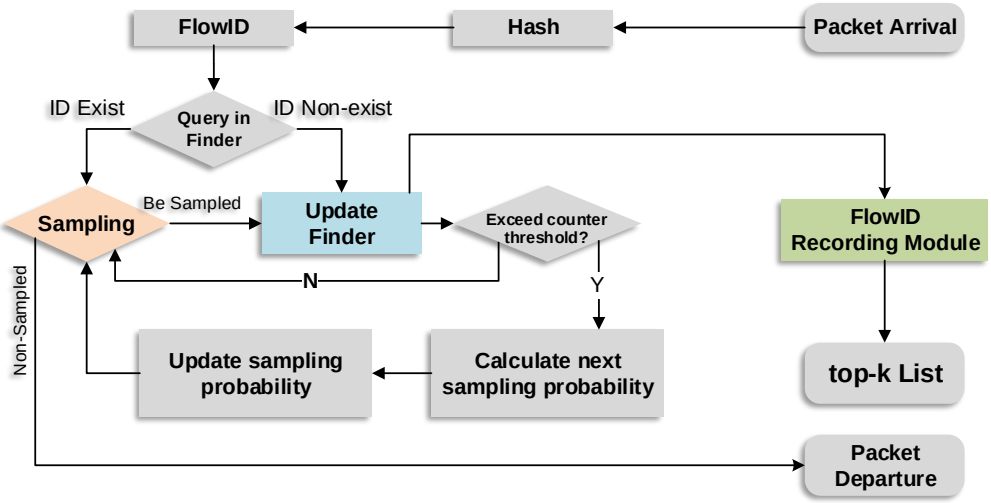


Figure 7. The structure of the flow label recording module in MSLCFinder.

Initialization: Let MSLCFinder be $MF[m, n]$, where m is the total number of counters in the MSLCFinder structure, and n is the number of bits in each counter. In addition, set the count in each counter in MSLCFinder as C, and the initial value of C is 0. Let the initial threshold of the top-k flow measurement task be θ. Choose a uth-level sampling method for flow sampling.

The minimum error that can be tolerated for the measurement task is δ. Calculate the number d of hash functions in MSLCFinder according to Formula (11).

$$d = \lceil \ln(1/\delta) \rceil \tag{11}$$

Then, calculate the initial sampling probability p_1 with Formulas (1) and (2).

To more simply describe the process of this algorithm, we choose the 3rd-level sampling methods. That is, u = 3. For each incoming packet Pac_ϑ belonging to flow f_φ, these are the following two steps for each insertion:

Step 1: The f_φ will be inserted in the MSLCFinder, shown in lines 3–24 in Algorithm 2. Hash the identity of the stream corresponding to each arriving packet Pac_ϑ, and the FlowID of f_φ is the hash results $h_1(f_\varphi), h_2(f_\varphi), ..., h_d(f_\varphi)$. If the count C is already equal to 2^n, do not insert anything into the counter and forward the packet. Otherwise, determine the size of C and the three thresholds, and use the sampling probability corresponding to the threshold interval to perform the flow sampling. If the Pac_ϑ has been sampled, the counter will be plus one, and the flag is true, which means that the count of f_φ is updated.

Step 2: We use the MSLCFinder flow label recording module to record the FlowID of f_φ, which is shown in lines 25–27 in Algorithm 2. We use the minimal $MF[hash_d(f_\varphi)].C$ to record into the module. The record in the module is timely updated by the function *Record*.

Algorithm 2: Top-k measurement algorithm based on MSLCFinder.

Input: A packet Pac_ϑ belong to flow f_φ, Sampling threshold θ, Counter length n

1 $new\_cnt \leftarrow 0$;
2 $u \leftarrow 3$;
3 $p_1 \leftarrow \frac{1}{\theta - [2^{n-2}-2]}$;
4 **for** $j \in 1 \rightarrow d$ **do**
5 $\quad C \leftarrow MF[hash_j(f_i)].C$;
6 $\quad Flow.ID \leftarrow MF[hash_j(f_i)].ID$;
7 $\quad$ **if** $C == 2^n$ **then**
8 $\quad\quad$ pass;
9 $\quad$ **end**
10 $\quad$ **if** $C < 2^{n-2}$ **then**
11 $\quad\quad$ Flow Sampling as p_1;
12 $\quad$ **end**
13 $\quad$ **if** $2^{n-2} \leq C < 2^{n-1}$ **then**
14 $\quad\quad$ Flow Sampling as p_2;
15 $\quad$ **end**
16 $\quad$ **if** $2^{n-1} \leq C < 2^n$ **then**
17 $\quad\quad$ Flow Sampling as p_3;
18 $\quad$ **end**
19 $\quad$ **if** Pac_ϑ *Be Sampled* **then**
20 $\quad\quad MF[hash_j(f_i)].C++$;
21 $\quad\quad Flag \leftarrow True$;
22 $\quad$ **else**
23 $\quad\quad$ pass;
24 $\quad$ **end**
25 **end**
26 **if** $Flag == True$ **then**
27 $\quad new\_cnt \leftarrow min(MF[hash_1(f_i)].C, MF[hash_2(f_i)].C, ..., MF[hash_d(f_\varphi)].C)$;
28 $\quad cnt \leftarrow Record(Flow.ID, new\_cnt)$;
29 **end**

Query top-k flows: After the measurement task, the MSLCFinder will report the information of k flows recorded in the flow label recording module, such as network quintuple.

Analysis: Our method's time complexity and space complexity are as follows: (1) The time complexity is $O(n \log k)$. When a new element comes, it needs $O(1)$ time to query in the lightweight counting module of MSLCFinder. Then, it needs $O(\log k)$ time to query or update the min-heap in the flow label recording module of MSLCFinder. Therefore, the total time complexity is $O(n \log k)$. (2) The space complexity: the lightweight counting module needs a $w \times d$ two-dimensional array, and the flow label recording module requires

$O(k)$ space, but k is usually tiny, and the space can be negligible. Thus, the total space complexity is $O(wd)$.

4. Evaluation

In this section, we display the experimental settings, including platform, datasets, and development environment. In addition, the experimental procedure and results are demonstrated after them. The depth analysis of experimental results is also displayed.

4.1. Experimental Settings and Datasets

Platform: We use a laptop with 4-core CPUs (8 threads, Intel Core i7-4940MX @3.10 GHz) and 32 GB total system memory to complete the development and implementation of the algorithmin this paper. We also use it to complete our experiments.

Dataset: We use three datasets to perform the experiment. All information of datasets is shown in Table 2.

(1) CAIDA-2018 [44]: The first dataset is a CAIDA Anonymized Internet Trace from 2018, made of anonymized IP packets. It has 27,121,713 packets belonging to 1,000,551 flows.

(2) CERNET-30 and CERNET-60 [45]: The second and third datasets comprise IP packets captured from the CERNET Backbone Node of East China and North China at different dates. For ethical reasons, we anonymized the IP information to a certain extent. The collection time of CERNET-30 is 30 minutes in the peak period, and the data size is about 35 GB, including 17,596,416 packets, about 11,941,109 flows in total. The collection time of CERNET-60 is one hour in the off-peak period, and the data size is about 46 GB, including 32,674,187 packets, with a total of about 16,082,037 flows.

Table 2. Information about the experimental datasets.

Name	Number of Packets	Number of Flows
CAIDA-2018	27,121,713	1,000,551
CERNET-30	17,596,416	11,941,109
CERNET-60	32,674,187	16,082,037

As shown in Table 3, we roughly compared the flow size distribution of the three datasets, selected the samples with the first 100 flow lengths, and analyzed the maximum, minimum, variance, and standard deviation of the flow size distribution. Our work aims to illustrate the influence of flow size distribution on our proposed method.

Table 3. The flow size distribution information about the experimental datasets.

Name	Maximum	Minimum	Variance	Standard Deviation
CAIDA-2018	460,200	15,982	$\approx 3.99 \times 10^9$	$\approx 6.29 \times 10^4$
CERNET-30	1,334,369	6568	$\approx 1.77 \times 10^{10}$	$\approx 1.32 \times 10^5$
CERNET-60	537,966	12,808	$\approx 1.16 \times 10^{10}$	$\approx 1.07 \times 10^5$

Implementation: The implementation of MSLCFinder is performed in C++. For the hash function, we use the improved IPSX [15] algorithm to support different measurement task requirements. To achieve at least 95% precision, we can calculate $d = 3$ by Formula (11). The total space cost is fixed as 2^{20}bit $= 1$ Mbit $= 128$ KB, and the counter size is fixed as $n = 8$ bit. There are two selected measurement tasks. One is to output the source IP and destination IP of top-k flows, and the other is to output the quintuple information of top-k flows.

The total number of counters is determined by the total memory space (1 Mbit), the k-value of the top-k task, and the size of the measure required for the final maintenance

of the top-k flow information table. To improve the versatility in different measurement tasks, we can also use half of the total memory to support the measurement function of MSLCFinder, and the other half of the memory to record and maintain the top-k flow information required in measurement tasks. However, if the available memory is abundant, it is unnecessary to provide half of the memory resources for recording and maintaining the top-k flow information.

4.2. Evaluation Metrics

Precision: Precision is defined as $\frac{k'}{k}$. k' means that the k' flows are real top-k flows.

Average relative error (ARE): ARE is defined as $\frac{1}{|\Phi|}\sum_{f_i \in \Phi}\frac{|\hat{n}_i - n_i|}{n_i}$, where Φ is estimated set of top-k flows, $\hat{n}_i$ is the estimated size of flow f_i, and n_i is the real size of flow f_i. ARE evaluates the error rate of the estimated flow size reported by the algorithm.

4.3. Experiments on Precision

The application scenario of our method is resource-constrained, so we use fixed-size memory and fixed-length counter in each experiment. In order to ensure the fairness of the comparative experiment, we choose to change only one specific factor rather than other factors in the experiment. We conduct the experiments for varying k and fixed θ or varying θ and fixed k on the CAIDA-2018, CERNET-30, and CERNET-60 datasets. We carry out two groups of experiments, using different measures as the measurement standard. One uses the five-tuple information as the flow ID, and the other uses the source/destination IP address (SrcIP/DstIP) pair as the flow ID.

In each group of experiments, we aim to carry out the measurement tasks of top-50, top-100, top-200, top-500, and top-1000 for three datasets, respectively, to verify the effect of this experimental scheme. The threshold θ is 74, 84, 94, 104, 114, 124, 174, and 200, respectively.

4.3.1. Result of Precision vs. Quintuple of Flows

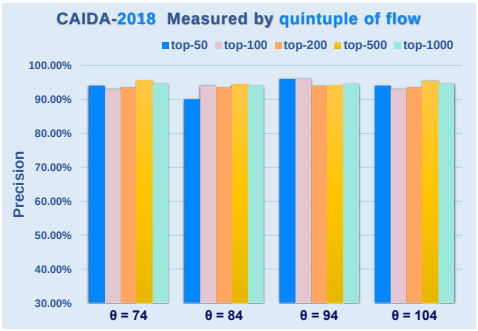
In this subsection, we use the quintuple information to identify the flow f_i. We summarize the experimental results and show them in Figure 8.

CAIDA-2018 with quintuple of flows: The result of the CAIDA-2018 dataset is shown in Figure 8a,b. In the experiment, with the gradual increase of the initial sampling threshold θ, the precision rate will increase and then decrease because the number of packets contained in the stream in the dataset is uneven. Secondly, with the increase of θ, the sampling probability will gradually decrease, and the randomness will increase, which will bring some errors to the measurement. In the experimental results, when $\theta = 84\sim104$, the precision rate of 94%∼97% can be obtained, which shows that our method can achieve top-k measurement tasks with certain requirements under resource constraints and can ensure high precision.

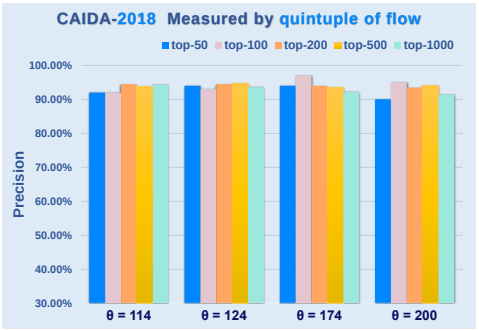
CERNET-30 with quintuple of flows: The result of the CERNET-30 dataset is shown in Figure 8c,d. In the experiment, with the gradual increase of the initial sampling threshold θ, the precision rate is kept at about 95% on average. Secondly, with the increase of θ, the sampling probability will gradually decrease, and the randomness will increase, which will bring some errors to the measurement. In the experimental results, the measurement precision of top-50/100/200 is as expected, and when $\theta = 84\sim114$, the precision rate can be 96%∼98.5%, which shows that our method can achieve the top-k measurement task with certain requirements under the condition of limited resources, and can ensure high precision. However, due to the limitation of the experimental dataset, the performance of the top-500 task was average, and the overall precision rate was 91.8%, which was related to the actual situation that the dataset was in line with the measurement threshold θ.

CERNET-60 with quintuple of flows: The result of the CERNET-60 dataset is shown in Figure 8e,f. In the experiment, with the gradual increase of the initial sampling threshold θ, the precision rate is kept at about 95% on average. Secondly, with the increase of θ, the sampling probability will gradually decrease, and the randomness will increase, which

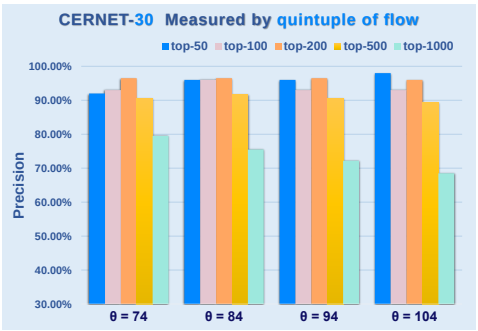
will bring some errors to the measurement. In the experimental results, the measurement precision of top-50/100/200/500 is excellent, and the precision rate of 95%∼98% can be obtained when $\theta = 84\sim104$, which shows that our method can achieve the top-k measurement task with certain requirements under the condition of limited resources, and can ensure high precision.



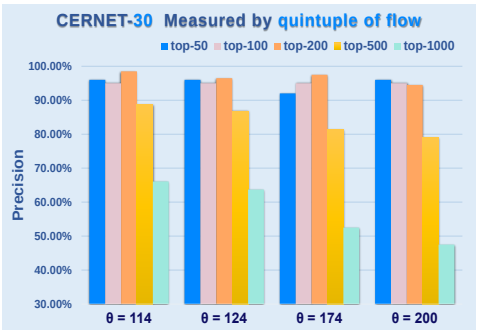
(a) Precision in CAIDA-2018
θ= 74/84/94/104



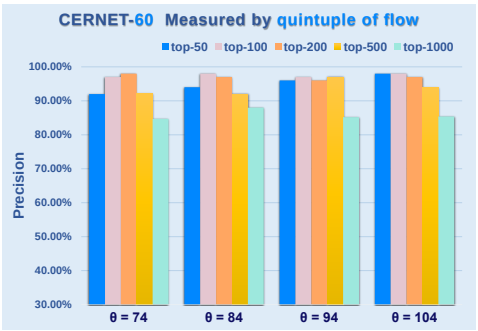
(b) Precision in CAIDA-2018
θ= 114/124/174/200



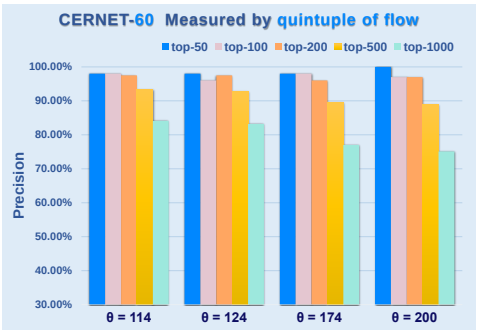
(c) Precision in CERNET-30
θ= 74/84/94/104



(d) Precision in CERNET-30
θ= 114/124/174/200



(e) Precision in CERNET-60
θ= 74/84/94/104



(f) Precision in CERNET-60
θ= 114/124/174/200

Figure 8. Experiments on precision vs. quintuple of flows. (**a**) Precision in CAIDA-2018. (**b**) Precision in CAIDA-2018. (**c**) Precision in CERNET-30. (**d**) Precision in CERNET-30. (**e**) Precision in CERNET-60. (**f**) Precision in CERNET-60.

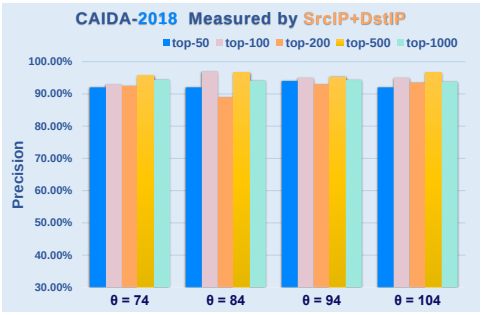
4.3.2. The Precision vs. SrcIP/DstIP

In this subsection, we use the SrcIP/DstIP information to identify the flow f_i. We summarize the experimental results and show them in Figure 9.

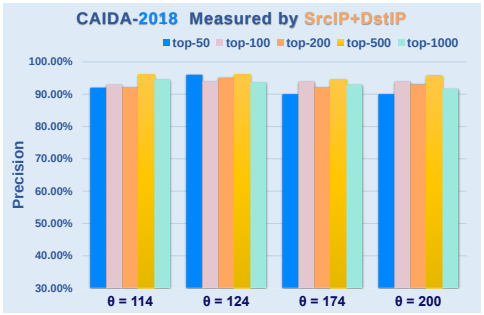
CAIDA-2018 with SrcIP/DstIP: The result of the CAIDA-2018 dataset is shown in Figure 9a,b. The experimental effect is equivalent to that of the previous five-tuple FlowID, and the precision of top-100 and top-200 is still about 95%. In the experiment of top-500, its precision is better than the former. When $\theta = 84\sim104$, the precision rate of 92%∼97%

can be obtained, which shows that our method can achieve top-k measurement tasks with certain requirements under resource constraints and can ensure high precision.

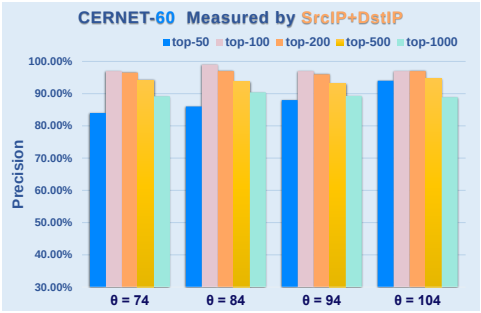
CERNET-30 with SrcIP/DstIP: The result of the CERNET-30 dataset is shown in Figure 9c,d. Due to different measures, the effect of this group of experiments is better than that of the previous group of experiments, especially in the top-500 experiment; its precision is 93%. When $\theta = 94 \sim 124$, the precision rate can be 97%∼98%, especially in the top-50/100/200 task. However, due to the limitation of the experimental dataset, the performance of the top-500 task was average, and the overall precision rate was 91.8%, which was related to the actual situation that the dataset was in line with the measurement threshold θ.



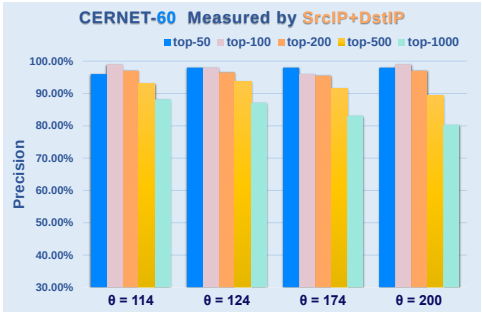
(a) Precision in CAIDA-2018 θ= 74/84/94/104



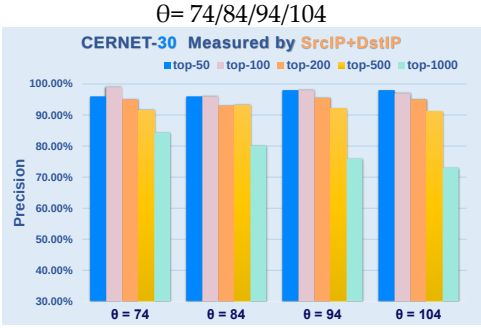
(b) Precision in CAIDA-2018 θ= 114/124/174/200



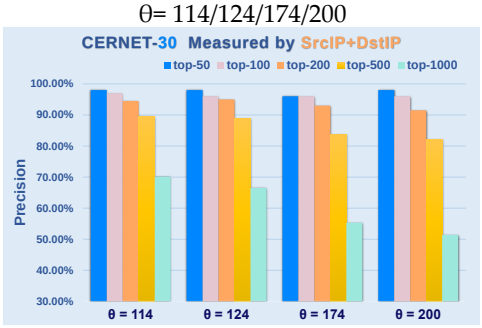
(c) Precision in CERNET-30 θ= 74/84/94/104



(d) Precision in CERNET-30 θ= 114/124/174/200



(e) Precision in CERNET-60 θ= 74/84/94/104



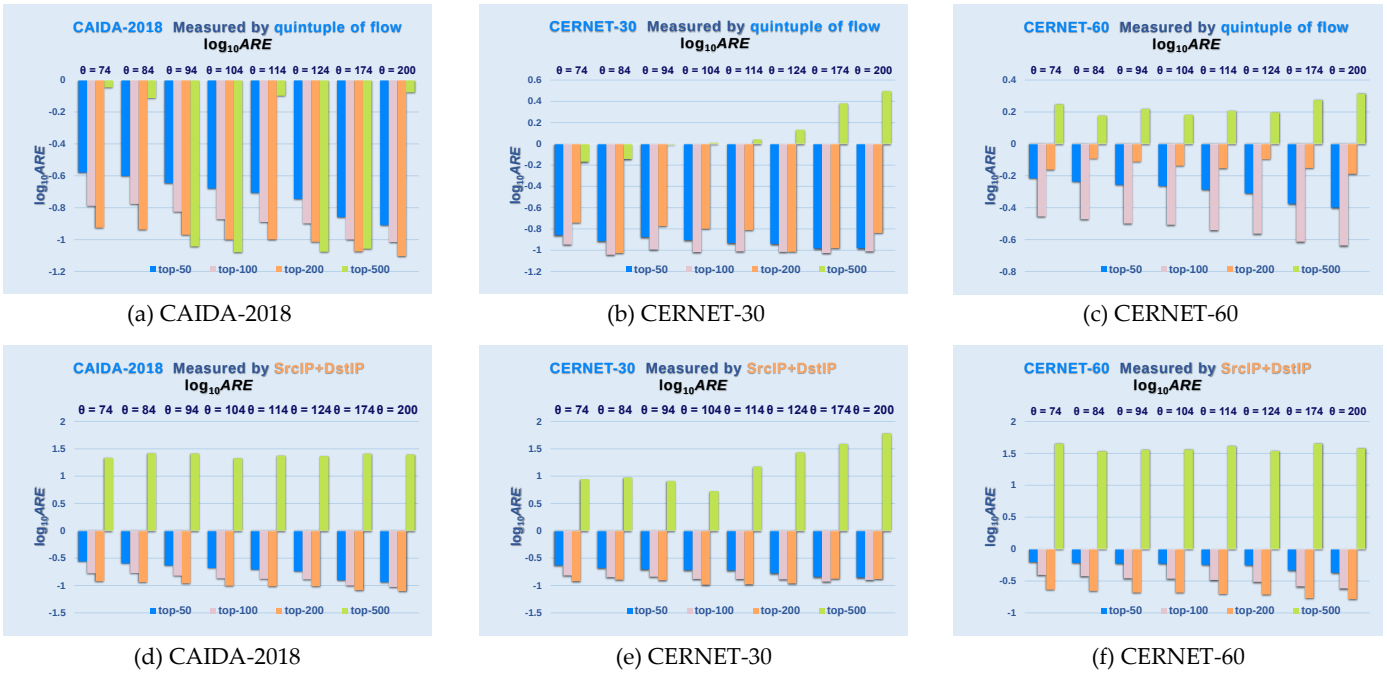
(f) Precision in CERNET-60 θ= 114/124/174/200

Figure 9. Experiments on precision *vs*. SrcIP/DstIP. (**a**) Precision in CAIDA-2018. (**b**) Precision in CAIDA-2018. (**c**) Precision in CERNET-30. (**d**) Precision in CERNET-30. (**e**) Precision in CERNET-60. (**f**) Precision in CERNET-60.

CERNET-60 with SrcIP/DstIP: The result of the CERNET-60 dataset is shown in Figure 9e,f. The measurement precision of top-50/100/200/500 is excellent, and the precision rate of 97%∼99% can be obtained when $\theta = 74 \sim 174$, which shows that our method can achieve the top-k measurement task with certain requirements under the condition of limited resources, and can ensure high precision.

4.3.3. The ARE Results

In order to make the results in the figure more intuitive, we performed an exponential operation (lg *ARE*) on the ordinate axis. All the results are shown in Figure 10. The average relative error is an important index to evaluate the simulation effect of the model. Because the method in this paper is based on the sampling strategy, which is different from the *count-all* strategy, there must be errors in sampling. Through this indicator, we can evaluate the reliability of our method.



(a) CAIDA-2018 (b) CERNET-30 (c) CERNET-60

(d) CAIDA-2018 (e) CERNET-30 (f) CERNET-60

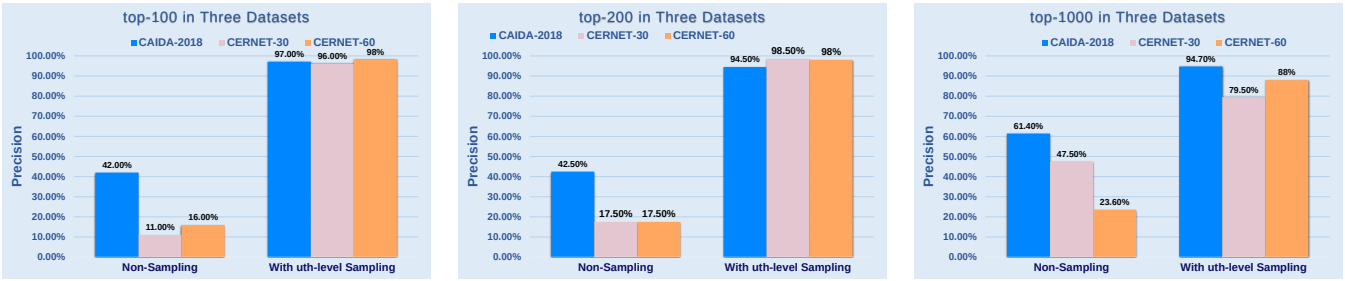
Figure 10. The ARE results of the evaluation. (**a**) The ARE results in CAIDA 2018 by quintuple of flows. (**b**) The ARE results in CERNET-30 by quintuple of flows. (**c**) tThe ARE results in CERNET-60 by quintuple of flows. (**d**) The ARE results in CAIDA 2018 by IP pairs. (**e**) The ARE results in CERNET-30 by IP pairs. (**f**) The ARE results in CERNET-60 by IP pairs.

Result of ARE vs. Quintuple of Flows: As shown in Figure 10a,b,c, for the CAIDA-2018 dataset in the measurement of quintuple of flow, we find that the ARE of MSLCFinder is less than 1. It shows that the difference between the predicted results obtained by our method and the precision results is slight. Although the task of top-500 made it challenging for this method to approach the limit under the condition of limited hardware resources, its ARE can still be less than 1. By the way, the performance of the CERNET-30 and CERNET-60 datasets is similar. There is a difference in the case of top-500. Due to the limited computing resources, the ARE is higher than 1. Nevertheless, the indicator of ARE is not very high. It still shows that the measurement results obtained by our method in this group of experiments are reliable.

Result of ARE vs. SrcIP/DstIP: As shown in Figure 10d,e,f, for all datasets, the ARE is acceptable. Since this measure is composed of IP address pairs, from the actual proportion, the proper top-*k* size is much larger than the flow size in the case of the quintuple, and the distribution is no longer extremely skewed. Therefore, although ARE is less than 1, it is not as good as the quintuple experiment. The reason for this problem is that the counter size is fixed. If the counter is complete, but the measurement task is not finished, although the flow is recorded as top-*k*, there will be a large gap between the estimated value and the actual value. In future work, based on this research, we can balance the compression of counting, i.e., to improve the precision of the estimation of the number of elephant flows.

4.4. Contribution of Key Technique

In order to verify the contribution of the key technique, which is the uth-level multi-sampling module, we additionally develop a comparison version of MSLCFinder. In this version, we remove the multi-sampling module and use only 8-bit counters to perform top-k measurement tasks on three datasets; specifically in Figure 11, we can intuitively see the gap and the importance of the uth-level multi-sampling module. Even in the case of the top-100 and top-200 with excellent results in the previous experiment, the finder that loses the sampling function will misjudge the mouse flows by a large margin. As the counter size of MSLCFinder is only n bits, it will be unable to cope with the measurement task of millions of flows without the multi-sampling module. If a bit counter with a long length is used, it will violate the original intention of the method proposed in this paper: to implement a top-k measurement task with a resource-limited environment. It will be similar to existing methods.



(a) Precision in Top-100 (b) Precision in Top-200 (c) Precision in Top-1000

Figure 11. Experiments verifying contribution of key technique. (**a**) Precision in top-100. (**b**) Precision in top-200. (**c**) Precision in top-1000.

4.5. Comparison with Existing Research Methods

In this subsection, we deploy the five usual methods in the existing research for the three datasets used in our previous evaluation to complete the comparative evaluations. We verify the feasibility and effectiveness of the method proposed in this paper in the previous evaluation. Notwithstanding, we prefer to verify the advantages of our proposed method based on MSLCFinder in the top-k measurement task.

4.5.1. Implementation

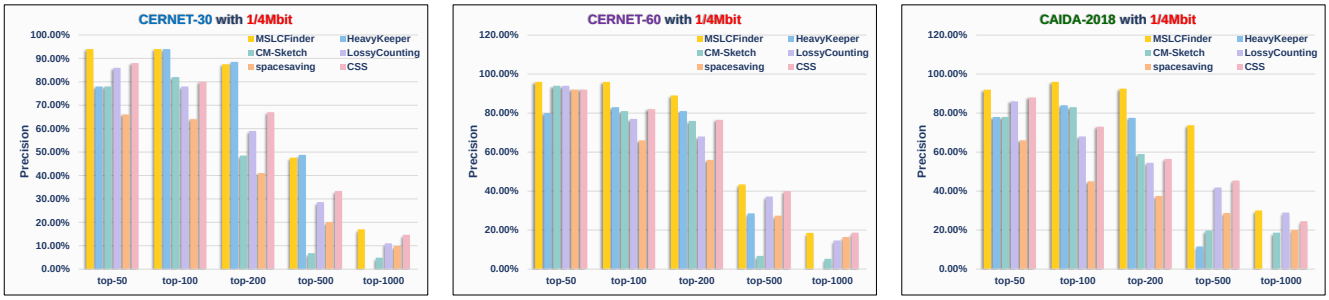
We choose five related methods, including HeavyKeeper [7], lossy counting [10], SpaceSaving [11], CSS [12], and count–min sketch [13]. We use C++ to complete the implementation of these methods. We determine the number of buckets w SpaceSaving, lossy counting, and CSS by the hardware resource (such as memory size) and the k of top-k. For HeavyKeeper and count–min sketch, the number of hash functions is 3, the heap size is k, and the width of each hash space is also determined by the hardware resource (such as memory size). We also choose to use the improved IPSX [15] algorithm for the hash function. The measure we choose is the quintuple information to identify the flow f_i.

In addition, we directly refer to the design and related parameters mentioned in [7] for HeavyKeeper implementation. The fingerprint field and the counter field of HeavyKeeper are both 16 bits long. For count–min sketch in our implementation, the counter is 32 bits long, equivalent to directly using the integer type in C++.

We set the memory size to be 1/4 Mbit, 1/2 Mbit, and 1 Mbit, and the varying k we set to 50, 100, 200, 500, and 1000. As with the previous evaluation methods, based on the idea of controlled experiments, we select an equal size of memory and use five related methods and MSLCFinder for experiments to complete the top-k measurement tasks with different k values. Additionally, in all the experimental results, the MSLCFinder experimental results are based on the multiple selections of threshold θ and the optimal results obtained after multiple experiments under the premise of keeping the memory size and k unchanged.

4.5.2. Precision vs. Three Datasets Using 1/4Mbit

Figure 12 shows the precision results of the top-*k* measurement experiments for three datasets using six methods when the memory size is 1/4 Mbit.



(a) CERNET30 with 1/4Mbit (b) CERNET60 with 1/4Mbit (c) CAIDA2018 with 1/4Mbit

Figure 12. Experiments on precision vs. six methods of three datasets with 1/4 Mbit. (**a**) CERNET-30 with 1/4 Mbit. (**b**) CERNET-60 with 1/4 Mbit. (**c**) CAIDA-2018 with 1/4 Mbit.

Due to the memory limitation of 1/4 Mbit, HeavyKeeper is not competent for this experiment's top-1000 measurement task of three datasets. This is mainly due to the lack of counters that can participate in the measurement.

For the CERNET-30 dataset, our method performs well in five measurement tasks, especially in evaluating top-50 to top-200, which can achieve a precision of about 90%. Although our method is slightly less accurate than HeavyKeeper in the top-200 and top-500 experiments, the overall effect is comparable to HeavyKeeper. On the other hand, compared with the other four methods, our method has a very intuitive advantage in 1/4 Mbit memory space. For the other two datasets, our method has better advantages.

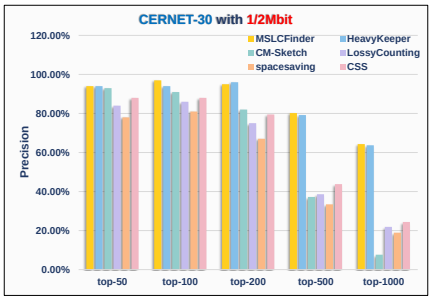
For the top-1000 measurement task, the information maintenance of 1000 stream IDs requires at least 104,000 bits, equivalent to 39.7% of the total memory in this group of experiments. Therefore, resources supporting various methods to complete the top-1000 measurement task are incredibly scarce. Nevertheless, our method can still achieve a precision of about 20%; especially in the CAIDA-2018 dataset, our method reached 30.1%. Moreover, they are better than the other four methods.

4.5.3. Precision vs. Three Datasets Using 1/2Mbit

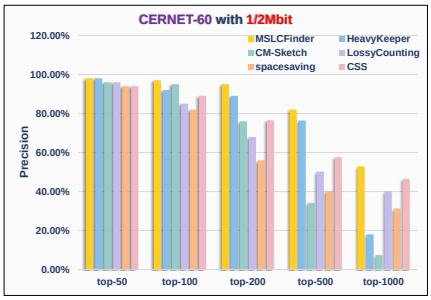
Figure 13 shows the precision results of the top-*k* measurement experiments for three datasets using six methods when the memory size is 1/2 Mbit.

Compared with the group of experiments in Section 4.5.2, the memory of this group is doubled. Under the memory size of 1/2 Mbit, the six methods have corresponding experimental results.

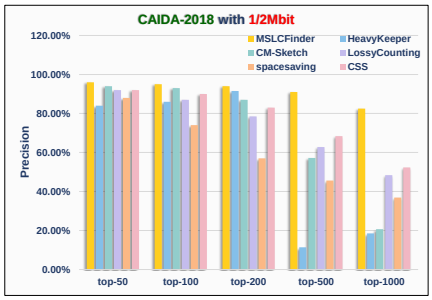
In the top-50/top-100/top-200 experiments, our method well matched experimental results of HeavyKeeper for the CERNET-30 dataset. Although our method is slightly inferior to HeavyKeeper in the top-200 experiment, the gap is tiny. It is superior to the other four methods and has better results than HeavyKeeper in CERNET-60 and CAIDA-2018 datasets. Moreover, our method achieved more than 95% precision in the three datasets. For the top-500 experiment, our method still has promising results on three datasets. It shows that our method can be further competent for the top-*k* measurement task in the case of a lack of resources.



(a) CERNET30 with 1/2Mbit



(b) CERNET60 with 1/2Mbit



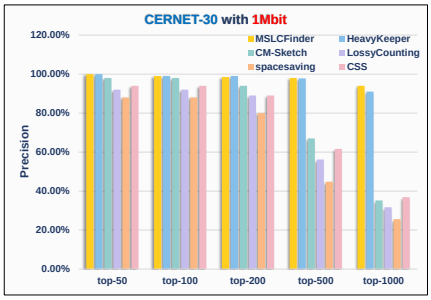
(c) CAIDA2018 with 1/2Mbit

Figure 13. Experiments on precision vs. six methods of three datasets with 1/2Mbit. (**a**) CERNET-30 with 1/2 Mbit. (**b**) CERNET-60 with 1/2 Mbit. (**c**) CAIDA-2018 with 1/2 Mbit.

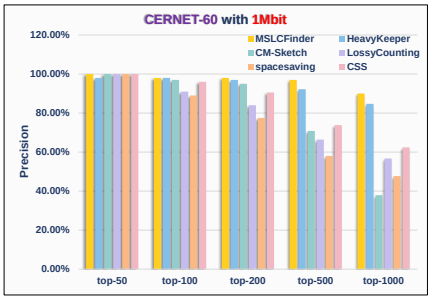
Although the memory is doubled, the measurement task of the top-1000 is still challenging. HeavyKeeper can output the measurement results in this group of experiments, but our method still has good advantages and is better than the other four methods. In addition, compared with the results in Section 4.5.2, 1/4 Mbit of memory is added to our method, which makes our method nearly four times more accurate in the measurement task of top-1000. Other methods involved in the comparison did not have such a good increase.

4.5.4. Precision vs. Three Datasets Using 1Mbit

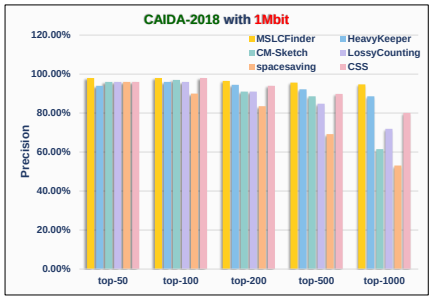
Figure 14 shows the precision results of the top-*k* measurement experiments for three datasets using six methods when the memory size is 1Mbit.



(a) CERNET30 with 1Mbit



(b) CERNET60 with 1Mbit



(c) CAIDA2018 with 1Mbit

Figure 14. Experiments on precision vs. six methods of three datasets with 1 Mbit. (**a**) CERNET-30 with 1 Mbit. (**b**) CERNET-60 with 1 Mbit. (**c**) CAIDA-2018 with 1 Mbit.

Compared with the first two groups of experiments, the 1Mbit memory is relatively abundant for the six methods involved in the experiment, especially in the top-50/top-100/top-200 measurement tasks. Although the six methods achieved similar precision, our method is still similar to HeavyKeeper in the three datasets and better than the other four methods.

For top-500 and top-1000, our method also has better results than the other four methods. Compared with HeavyKeeper, our method still outputs excellent measurement results.

4.6. Analysis and Discussion

4.6.1. Why is Finding Top-1000 Less Effective?

The results of top-1000 tasks are shown in Figures 8, 9, and 12–14, and the results of top-1000 are not optimistic. We think there are several reasons for this:

(1) Due to the strict limitation of 1 Mbit space, when the top-1000 measurement task is executed in this scheme, the size of the information storage table used to maintain the flow information will also increase with the increase of *k*. As mentioned earlier, when we

use 1/4 Mbit memory, the information of the top-1000 will consume at least 40% of the memory. It will reduce the number of finder counters used for measurement tasks and affect the measurement results. Similarly, the five methods involved in the comparative experiment will also make the number of counters involved in the measurement scarce. In addition, for MSLCFinder, it is not ruled out that the number of elephant flows that meet the measurement threshold θ in the dataset involved in the measurement is not 1000.

(2) The flow size distribution in the high-speed network is always highly skewed [46]. We summarized the flow size distribution based on a quintuple of three datasets in Table 3. Although they are not standard heavy-tailed distributions, they are still highly skewed, especially CERNET-30. Compared with the CERNET-30 dataset, the flow size distribution of the CAIDA-2018 public dataset is relatively uniform, with small differences, and there are relatively many streams with large flow sizes. In the experimental results, the top-1000 precision obtained from the CAIDA-2018 dataset is better than the other two. Although the precision is somewhat low, this does not affect the effectiveness of our method. The effect will be significantly improved if we increase the total memory size.

4.6.2. The Necessity of Using Different Measures to Complete the Experiment

We use different FlowID to show MSLCFinder's universality in network traffic header fields, i.e., MSLCFinder can be deployed on different header fields or other measures. Furthermore, the experimental results show that the performances of MSLCFinder on datasets with different header fields are similar.

In the same dataset, the distribution calculated by using a quintuple as the flow ID will be different to that when using an IP address pair. It is easy to understand that the communication between the same IP can be multiple ports and protocols.

When using an IP address pair as the flow ID, it is equivalent to using only the characteristics of the IP layer and generalizing the specific information contained in the transport layer. On the other hand, if the mouse flows between different ports of many different protocols are generated by the same pair of IP addresses, the pair of IP addresses is likely to become a member of the top-k flow when measured by the IP address pair.

Therefore, it is necessary to use different measures to verify the effectiveness of our method in the same dataset. We designed the experiment to show MSLCFinder's universality in terms of network traffic header fields.

4.6.3. The Energy Cost and Complexity of MSLCFinder

The method based on MSLCFinder works in a resource-constrained environment, including the CPU, memory, power, and storage. Our method benefits from the advantages of the data flow algorithm; MSLCFinder requires small memory. Due to the real-time, continuous, and unbounded characteristics of network flows in high-speed links, the algorithms for processing data flows can only perform one calculation on the network flows and can only use limited computing and memory resources.

Therefore, when designing the network data flow method, it is necessary to meet the following requirements: (1) the space used by the algorithm must be small enough; (2) processing and updating must be simple and rapid; (3) the query must be accurate. In other words, our method is based on the development and design of the network data flow method, which fundamentally limits energy consumption and time and space complexity. In addition, the total time complexity of our method is $O(n \log k)$, and the total space complexity is $O(wd)$.

4.6.4. The Threshold θ

In the experiment, the selection of threshold θ has a guiding influence on the experimental effect. Because the size of the flow is not known prior to measurement, the selected threshold θ will have different effects on different flows; however, the selection of threshold θ is also one of the critical parameters of the method proposed in this paper.

Unlike the static selection of security parameters of NetFlow [5], our goal is not to set sampling probability in the worst case to ensure that the network equipment can operate continuously under an adverse traffic environment. Our proposed scheme focuses on whether or not to use limited memory resources and limited counter units to complete the measurement of top-k elephant flow. We aim to not pursue a minor error between the estimated value and the actual value. In addition, our method has a fixed initial threshold θ to determine the sampling probability, but our sampling probability is recalculated with several counting thresholds in the counter.

In the open world, the flow distribution of unit measurement time period is different; only the mouse flow smaller than the decision condition likely exists in the entire measurement cycle, resulting in a decline in the effect of the top-k elephant flows. Therefore, in future work, we will improve the sampling threshold's adaptive generation and increase our scheme's applicability in the complex and changeable real network environment.

5. Conclusions

Finding the top-k elephant flows is critical for network traffic measurement, especially with resource constraints. As the network traffic overgrows, finding top-k flows with limited resources is challenging. In order to overcome the mutual restriction of hardware resources, massive data, and measurement precision, we adopt a novel strategy called *count-with-uth-level-sampling*. Moreover, we propose a novel core component named Multi-Sampled Lightweight Counting Finder (MSLCFinder) to achieve a measurement scheme for finding top-k flows based on hardware resource-constrained environments. The MSLCFinder has three primary modules: a lightweight counting module, a multi-sampling module, and a flow label recording module. Three modules complement and constrain each other to maximize the optimization of the MSLCFinder in the measurement task. The essential technique of MSLCFinder is that it uses a uth-level multi-sampling module to relieve the storage pressure of the MSLCFinder, and also reduces the possibility of the mouse flows being recorded by the flow label recording module. Our evaluation confirms that MSLCFinder can achieve more than 97% precision. With MSLCFinder that we proposed, we can effectively find top-k elephant flows with less memory overhead and counters under resource constraints.

Although the MSLCFinder can achieve good measurement results with small hardware resources, experiments show that this overhead is still high, especially in the flow label recording module. Therefore, in future research, we have the following issues that need to be further studied and optimized: (1) We first need to improve the efficiency of MSLCFinder, including updating the technology and methods at this stage, reconstructing core record data structure instead of min-heap to achieve a faster, more efficient, and more concise recording algorithm. (2) Our *count-with-uth-level-sampling* strategy can sample each flow for many continuous arriving flows in the network before the finder counts. Compared with the *count-all* strategy, it achieves a certain degree of filtering effect on mouse flows, but the accuracy of the strategy in filtering mouse flows needs further research and discussion. Therefore, we need to introduce effective filtering strategies in future work to improve the utilization of memory resources and the accuracy of identifying top-k elephant flows. (3) The initial sampling threshold θ has a decisive influence on the global measurement results. Determining how to generate sampling threshold θ based on flow adaptation or self-learning is the focus of our future research. (4) This paper only focuses on identifying elephant flow in network measurement, ignoring the identification of mouse flow. In the actual network environment, the traffic information of many attacks (such as DDoS attacks) in the network is composed of mouse flows. Based on the distribution characteristics of network traffic, it can be seen that many mouse flows in high-speed network links cannot be accurately monitored in real time. Therefore, in future work, we will continue to study how to identify mouse flow accurately. (5) In future research, we plan to design a machine learning or integrated learning method to adaptively calculate the initial sampling threshold for per-flow according to the traffic distribution in the current network environment to be

tested to replace the current single global threshold scheme. (6) We will continue developing parallel hardware versions of MSLCFinder to achieve more efficient measurement results.

Author Contributions: Conceptualization, X.D. and G.C.; methodology, X.D. and R.Z.; software, X.D. and Z.Y.; validation, X.D. and G.C.; formal analysis, X.D. and Z.Y.; investigation, X.D. and R.Z.; resources, Z.Y. and R.Z.; data processing, Z.Y. and X.D.; writing—original draft preparation, X.D.; writing—review and editing, X.D. and Y.Y.; visualization, X.D.; supervision, G.C.; project administration, G.C.; funding acquisition, G.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Joint Key Program of the National Natural Science Foundation of China under grant number U22B2025 and the General Program of the National Natural Science Foundation of China under grant number 62172093.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Three anonymized datasets in this paper are published at: https://github.com/ccie44899/MSCLFinder.git (accessed on 28 November 2022).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. We Are Social and Hootsuite. Digital 2022 October Global Statshot. Available online: https://datareportal.com/reports/digital-2022-october-global-statshot (accessed on 25 November 2022).
2. Google. *Encrypted Traffic in All Google Products and Services*; Google AI China Center: Beijing, China, 2022.
3. Yuan, Y.; Wang, W.; Wang, Y.; Adhatarao, S.S.; Ren, B.; Zheng, K.; Fu, X. VSiM: Improving QoE Fairness for Video Streaming in Mobile Environments. In Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications, London, UK, 2–5 May 2022; pp. 1309–1318.
4. Zhou, A.P.; Cheng, G.G.X. High-Speed network traffic measurement method. *Ruan Jian Xue Bao/J. Softw.* **2014**, *25*, 135–153. (In Chinese)
5. Estan, C.; Keys, K.; Moore, D.; Varghese, G. Building a better NetFlow. *ACM Sigcomm Comput. Commun. Rev.* **2004**, *34*, 245–256. [CrossRef]
6. Estan, C.; Varghese, G. New directions in traffic measurement and accounting. In Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Pittsburgh, PA, USA, 19–23 August 2002; pp. 323–336.
7. Yang, T.; Zhang, H.; Li, J.; Gong, J.; Uhlig, S.; Chen, S.; Li, X. HeavyKeeper: An Accurate Algorithm for Finding Top-*k* Elephant Flows. *IEEE/ACM Trans. Netw.* **2019**, *27*, 1845–1858. [CrossRef]
8. Demaine, E.D.; López-Ortiz, A.; Munro, J.I. Frequency estimation of internet packet streams with limited space. In Proceedings of the European Symposium on Algorithms, Rome, Italy, 17–21 September 2002; pp. 348–360.
9. Wang, W.P.; Li, J.Z.; Zhang, D.D.; Guo, L.J. Efficient algorithm for mining approximate frequent item over data streams. *Ruan Jian Xue Bao (J. Softw.)* **2007**, *18*, 884–892. [CrossRef]
10. Manku, G.S.; Motwani, R. Approximate frequency counts over data streams. *Proc. VLDB Endow.* **2012**, *5*, 1699. [CrossRef]
11. Metwally, A.; Agrawal, D.; Abbadi, A.E. Efficient computation of frequent and top-k elements in data streams. In Proceedings of the International Conference on Database Theory, Edinburgh, UK, 5–7 January 2005; pp. 398–412.
12. Ran, B.B.; Einziger, G.; Friedman, R.; Kassner, Y. Heavy hitters in streams and sliding windows. In Proceedings of the IEEE Infocom—The IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016.
13. Cormode, G.; Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms* **2005**, *55*, 58–75. [CrossRef]
14. Charikar, M.; Chen, K.; Farach-Colton, M. Finding frequent items in data streams. In Proceedings of the International Colloquium on Automata, Languages, and Programming, Malaga, Spain, 8–13 July 2002; pp. 693–703.
15. Guang, C.; Jian, G.; Wei, D.; Jialing, X.U. A Hash Algorithm for IP Flow Measurement. *Ruan Jian Xue Bao/J. Softw.* **2005**, *16*, 652–658. (In Chinese)
16. Zseby, T. Sampling and filtering techniques for IP packet selection. *RFC5475* 2009; pp. 1–46. Available online: https://www.rfc-editor.org/rfc/rfc5475 (accessed on 28 November 2022).

17. Jain, R. A Comparison of Hashing Schemes for Address Lookup in Computer Networks. *IEEE Trans. Commun.* **2002**, *40*, 1570–1573. [CrossRef]
18. Wegman, C. Universal classes of hash functions. *J. Comput. Syst. Sci.* **1979**, *18*, 143–154.
19. Cheng, G.; Gong, J.; Ding, W. Distributed sampling measurement model in a high speed network based on statistical analysis. *Chin. J. Comput.-Chin. Ed.* **2003**, *26*, 1266–1273.
20. Duffield, N.G.; Grossglauser, M. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Netw.* **2001**, *9*, 280–292. [CrossRef]
21. He, G.; Hou, J.C. On sampling self-similar Internet traffic. *Comput. Netw.* **2006**, *50*, 2919–2936. [CrossRef]
22. Raspall, F. Efficient packet sampling for accurate traffic measurements. *Comput. Netw.* **2012**, *56*, 1667–1684. [CrossRef]
23. Du, Y.; Huang, H.; Sun, Y.E.; Chen, S.; Gao, G. Self-Adaptive Sampling for Network Traffic Measurement. In Proceedings of the IEEE INFOCOM 2021—IEEE Conference on Computer Communications, Vancouver, BC, Canada, 10–13 May 2021.
24. Sanjuaas-Cuxart, J.; Barlet-Ros, P.; Duffield, N.; Kompella, R. Cuckoo sampling: Robust collection of flow aggregates under a fixed memory budget. In Proceedings of the IEEE Infocom, Orlando, FL, USA, 25–30 March 2012; pp. 2751–2755.
25. Hu, C.; Liu, B.; Wang, S.; Tian, J.; Cheng, Y.; Chen, Y. ANLS: Adaptive Non-Linear Sampling Method for Accurate Flow Size Measurement. *IEEE Trans. Commun.* **2012**, *60*, 789–798. [CrossRef]
26. Ma, X.; Hu, C.; Jiang, J.; Jing, W. S3: Smart selection of sampling function for passive network measurement. In Proceedings of the Local Computer Networks, Bonn, Germany, 4–7 October 2011.
27. Estan, C.; Varghese, G.; Fisk, M. Bitmap Algorithms for Counting Active Flows on High-Speed Links. *IEEE/ACM Trans. Netw.* **2006**, *14*, 925–937. [CrossRef]
28. Tarkoma, S.; Rothenberg, C.E.; Lagerspetz, E. Theory and Practice of Bloom Filters for Distributed Systems. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 131–155. [CrossRef]
29. Wang, P.; Guan, X.; Zhao, J.; Tao, J.; Qin, T. A New Sketch Method for Measuring Host Connection Degree Distribution. *Inf. Forensics Secur.* **2014**, *9*, 948–960. [CrossRef]
30. Gryaditskaya, Y.; Sypesteyn, M.; Hoftijzer, J.W.; Pont, S.C.; Durand, F.; Bousseau, A. OpenSketch: a richly-annotated dataset of product design sketches. *ACM Trans. Graph.* **2019**, *38*, 232–241. [CrossRef]
31. Huang, Q.; Xin, J.; Lee, P.; Li, R.; Gong, Z. SketchVisor: Robust Network Measurement for Software Packet Processing. In Proceedings of the Acm Sigcomm Conference, Los Angeles, CA, USA, 21–25 August 2017.
32. Huang, Q.; Lee, P.; Bao, Y. Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference. In Proceedings of the the 2018 Conference of the ACM Special Interest Group, Budapest, Hungary, 20–25 August 2018.
33. Liu, Z.; Ran, B.B.; Einziger, G.; Kassner, Y.; Sekar, V. Nitrosketch: robust and general sketch-based monitoring in software switches. In Proceedings of the the ACM Special Interest Group, Beijing, China, 19–23 August 2019.
34. Jiang, J.; Fu, F.; Yang, T.; Cui, B. SketchML: Accelerating distributed machine learning with data sketches. In Proceedings of the 2018 International Conference on Management of Data, Houston, TX, USA, 10–15 June 2018; pp. 1269–1284.
35. Zhou, Z.; Zhang, D.; Hong, X. RL-Sketch: Scaling Reinforcement Learning for Adaptive and Automate Anomaly Detection in Network Data Streams. In Proceedings of the 2019 IEEE 44th Conference on Local Computer Networks (LCN), Osnabrueck, Germany, 14–17 October 2019; pp. 340–347.
36. Fu, Y.; Li, D.; Shen, S.; Zhang, Y.; Chen, K. Locality-sensitive sketching for resilient network flow monitoring. *arXiv* **2019**, arXiv:1905.03113.
37. Cormode, G. Sketch techniques for approximate query processing. *Foundations and Trends in Databases*; NOW Publishers: Hanover, MA, USA, 2011; p. 15.
38. Schweller, R.; Gupta, A.; Parsons, E.; Chen, Y. Reversible sketches for efficient and accurate change detection over network data streams. In Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference (IMC), Taormina Sicily, Italy, 25–27 October 2004; pp. 207–212.
39. Yang, Z.; Tong, Y.; Jie, J.; Cui, B.; Uhlig, S. Cold Filter: A Meta-Framework for Faster and More Accurate Stream Processing. In Proceedings of the the 2018 International Conference, Houston, TX, USA, 10–15 June 2018; pp. 741–756.
40. Tong, Y.; Gong, J.; Zhang, H.; Lei, Z.; Li, X. HeavyGuardian: Separate and Guard Hot Items in Data Streams. In Proceedings of the the 24th ACM SIGKDD International Conference, London, UK, 19–23 August 2018.
41. Wu, M.; Huang, H.; Sun, Y.E.; Du, Y.; Chen, S.; Gao, G. Activekeeper: An accurate and efficient algorithm for finding top-k elephant flows. *IEEE Commun. Lett.* **2021**, *25*, 2545–2549. [CrossRef]
42. Wang, Y.; Li, D.; Wu, J. FastKeeper: A Fast Algorithm for Identifying Top-k Real-time Large Flows. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–7.
43. Xiao, Q.; Tang, Z.; Chen, S. Universal online sketch for tracking heavy hitters and estimating moments of data streams. In Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; pp. 974–983.
44. CAIDA. The CAIDA UCSD Anonymized Internet Traces. 2018. Available online: http://www.caida.org/data/overview/ (accessed on 25 November 2022).

45. CERNET. CERNET East China and North China Node Network Center. Available online: https://www.njnet6.edu.cn/ (accessed on 28 November 2022).
46. Li, Z.; Xiao, F.; Wang, S.; Pei, T.; Li, J. Achievable rate maximization for cognitive hybrid satellite-terrestrial networks with AF-relays. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 304–313. [CrossRef]