Spread Estimation With Non-Duplicate Sampling in High-Speed Networks

He Huang, *Member, IEEE, ACM, Yu-E Sun, Member, IEEE, ACM, Chaoyi Ma, Graduate Student Member, IEEE, Shigang Chen, Fellow, IEEE, Yang Du, Member, IEEE, Haibo Wang, Graduate Student Member, IEEE, and Qingjun Xiao, Member, IEEE*

***Abstract*— Per-flow spread measurement in high-speed networks has many practical applications. It is a more difficult problem than the traditional per-flow size measurement. Most prior work is based on sketches, focusing on reducing their space requirements in order to fit in on-chip cache memory. This design allows the measurement to be performed at the line rate, but it suffers from expensive computation for spread queries (unsuitable for online operations) and large errors in spread estimation for small flows. This paper complements the prior art with a new spread estimator design based on an on-chip/off-chip model. By storing traffic statistics in off-chip memory, our new design faces a key technical challenge to design an efficient online module of non-duplicate sampling that cuts down the off-chip memory access. We first propose a two-stage solution for non-duplicate sampling, which is efficient but cannot handle well a sampling probability that is either too small or too big. We then address this limitation through a three-stage solution that is more space-efficient. Our analysis shows that the proposed spread estimator is highly configurable for a variety of probabilistic performance guarantees. We implement our spread estimator in hardware using FPGA. The experiment results based on real Internet traffic traces show that our estimator produces spread estimation with much better accuracy than the prior art, reducing the mean relative (absolute) error by about one order of magnitude. Moreover, it increases the query throughput by around three orders of magnitude, making it suitable for supporting online queries in real time.**

Manuscript received February 15, 2020; revised October 16, 2020 and January 30, 2021; accepted April 25, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Kuzmanovic. This work was supported in part by the NSF under Grant STC-1562485 and Grant CNS-1719222; in part by a grant from the Florida Cybersecurity Center; and in part by the National Natural Science Foundation of China (NSFC) under Grant 62072322, Grant 61873177, Grant U20A20182, and Grant 61872080. The preliminary version of this paper appeared in IEEE INFOCOM 2020 [1]. *(Corresponding author: Yu-E Sun.)*

He Huang is with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China, and also with the Department of Computer and Information of Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: huangh@suda.edu.cn).

Yu-E Sun is with the School of Rail Transportation, Soochow University, Suzhou 215131, China, and also with the Department of Computer and Information of Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: sunye12@suda.edu.cn).

Chaoyi Ma, Shigang Chen, and Haibo Wang are with the Department of Computer and Information of Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: ch.ma@ufl.edu; sgchen@cise.ufl.edu; wanghaibo@ufl.edu).

Yang Du is with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China (e-mail: duyang@suda.edu.cn).

Qingjun Xiao is with the School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China (e-mail: csqjxiao@seu.edu.cn).

Digital Object Identifier 10.1109/TNET.2021.3078725

***Index Terms*— Traffic measurement, sampling, spread estimation, high-speed network.**

I. INTRODUCTION

PER-FLOW spread measurement on a packet stream in high-speed networks is a fundamental problem with many practical applications [2]–[5]. Different from counting the number of packets (*i.e.*, flow-size measurement) [6]–[9], a spread measurement task estimates the number of distinct elements in each flow, where flows can be TCP flows, P2P flows, or any other types defined based on application-specific interests, and elements may be destination/source addresses, ports, other header fields, or payload content. For instance, we may consider all of the packets sent from the same source as a per-source flow, and elements can be distinct destination addresses carried in the packets of the flow. As an application, measuring the number of distinct destinations from each external source at the gateway helps identify scanners. By defining flows and elements differently, spread measurement can be used in many other applications, including worm monitoring, DDoS detection, and content access profiling [10], [11].

The function of traffic measurement can be implemented either entirely on the network processor chip (where the packet stream is forwarded) or using off-chip memory to record traffic data. The former has the advantage of keeping up with the line rate, while the latter has the advantage of leaving precious on-chip resources to key network functions, such as routing-table lookup, access control, and traffic engineering. Most sketch-based prior work chooses the on-chip approach [9], [12]–[17]. Their designs focus on how to reduce their space requirements in order to fit in on-chip memory. To do so, they have to make a tradeoff to sacrifice in other regards. For example, their compact data structures make it harder to query for a flow's spread, which will require scanning hundreds or thousands of bits or registers [12]–[14], [18]–[20]. Therefore, they are more suitable for offline queries instead of online queries that are triggered by packet inspection in live traffic. Moreover, their designs make sure that the accuracy of spread estimation is good for large flows, but not necessarily for small flows due to space-accuracy compromise. We want to stress that small-spread flows are sometimes important to certain applications. For example, some denial-of-service attackers may stay low-key by slowing down its attack rate, avoiding detection that focuses on large flows [20]. Thus, accurate estimations for the small flows' spreads are indispensable to detect those stealthy attackers.

This paper adopts an on-chip/off-chip design, which utilizes both on-chip memory for speed and off-chip memory for space. On the one hand, on-chip resources are often limited due to speed, power, and price considerations [21]. On the other hand, unlike per-flow size measurement, which needs only a counter for each flow, per-flow spread measurement requires much more resources because we have to remember which elements are new and which are duplicates that have already been carried in the previous packets — measuring the spread of a flow only counts the new elements. Moreover, modern network traffic is huge, which aggravates the resource demand. For instance, we observe that one-hour traffic trace downloaded from CAIDA [22] can easily include over multiple millions of per-source flows, tens of millions of distinct elements (*e.g.*, destinations), and billions of packets where duplicate elements are numerous. Therefore, it makes sense to utilize both on-chip memory and off-chip memory in a design that exploits the former's speed and the latter's space.

Our choice of using off-chip memory for traffic measurement is typical in practice (NetFlow [23], [24]) and has been studied for flow-size and heavy-hitter measurements [25]–[29]. But there is no prior work on per-flow spread measurement under this model. To compensate the difference between on-chip speed and off-chip access, a sampling module will be needed on-chip to select a subset of packets for recording randomly. Sampling is simple for flow-size measurement because each packet is treated independently with the same sampling probability. It is, however, tricky for flow-spread measurement because duplicates should not be sampled and on-chip operation must be efficient, which prevents us from using conventional methods to search for duplicates.

In this paper, we formally define a new problem of non-duplicate sampling, which requires that any element should be sampled only once with a given probability, regardless of how many times that element shows up or where it shows up in the packet stream. Our first solution to this problem consists of two stages of processing, where one stage is to check whether an arrival element appears for the first time (with the help of a bitmap) and the other stage is to control the sampling probability of each element to a fixed value. This solution works well for a range of sampling probabilities, but not when the sampling probability is too small or too large. We then propose a more sophisticated three-stage solution (with the help of a bloom filter) for non-duplicate sampling, which is more space-efficient and can work for all sampling probabilities. We mathematically derive the optimal system parameters that minimize the space overhead and achieve probabilistic performance guarantees, including bounded spread-estimation error and probabilistic assurance on threshold-based classification. We implement the proposed non-duplicate sampling module and the on-chip/off-chip spread estimator in both software and hardware. We perform extensive experiments based on real Internet traces from CAIDA [22]. The experimental results show that our spread estimator increases query throughput by around three orders of magnitude, reduces spread estimation error by about an order of magnitude, and incurs a much smaller on-chip footprint than the prior art.

The rest of the paper is organized as follows. We first formulate the problem we studied in this work, the limitations of the prior art, and our design goals in Section II. Then, we propose two solutions for spread estimation by non-duplicate sampling in Section III and further analyze the performance of our estimator in Section VI. To evaluate the performance of the proposed estimator, we use the real network traffic traces to perform simulations in Section VII. Section VIII describes the related work. At last, we conclude the paper in Section IX.

II. Preliminaries

A. System Model

A flow under measurement in high-speed networks consists of all packets that share the same values in a pre-defined subset of the header fields, which together form the flow label. Flow spread is defined as the number of distinct elements in each flow, where elements are defined based on application requirements. The problem is to estimate the spread of each flow.

We adopt an on-chip/off-chip model. A sampling module is placed on the network processor chip. We use a bit array, denoted as B, in the sampling module to help select new elements carried in the packet stream and filter out duplicates. Time is divided into epochs. All bits in B are reset to zeros at the beginning of each epoch. A recording/estimation module is implemented in off-chip memory to record the sampled elements and answer online queries in real-time.

B. Prior Art and Limitations

Most existing spread estimators [12], [13], [18]–[20] are sketch-based under a different system model, where the whole sketch data structure is placed in on-chip memory. This model choice results in three limitations that the proposed work in this paper will address. First, the whole data structure has to be very compact. Consequently, such spread estimators do not keep track of flow labels. Given a flow label, they can answer the flow's spread, but they have to rely on external means to record the flow labels. Second, their estimation formulas are expensive to compute. Hence, the periodically recorded data structures will be sent to an offline server, which answers queries on flow spread. This is ok for many applications based on offline traffic analysis, but it is also restrictive for online applications. Third, due to compactness in their data structures, the accuracy in spread estimation has to give, particularly for small flows. Recently, SketchVisor [30] tries to augment sketch-based methods by using a fast path when traffic load is high. However, it adopts the same on-chip model as existing sketch-based methods and has the same limitations. Besides, SketchVisor is more concerned about system implementation. It does not provide a new solution either for non-duplicate sampling or for spread measurement.

Take the spread estimator in [20] as an example. It shares a single physical bitmap among all of the flows to record their elements statistically under a novel concept of virtual bitmaps. However, on-chip compactness requires aggressive space sharing, which results in significant errors for small/medium flows. We use one-minute traffic downloaded from CAIDA as our test dataset, which contains 589740 per-source flows and 907463 distinct destinations (elements). By setting the packet sampling rates at 1 and 0.1, respectively, the experimental results are as shown in Fig. 1 in log scale, where each point represents a flow with its x coordinate being the actual spread and its y coordinate being the estimated spread. The more a point deviates from the equality line $y = x$, the less accurate the estimation is. Clearly, small flows suffer very large (relative) errors. Moreover, estimating the spread of each flow requires examining thousands of bits.

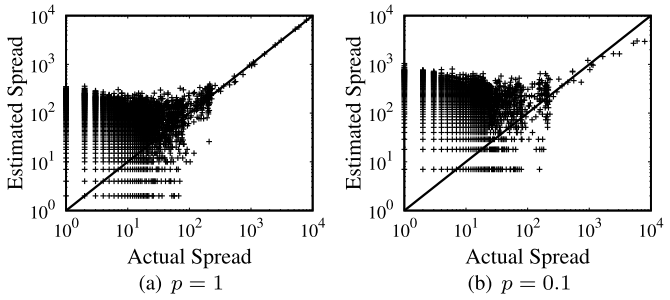


Fig. 1. Spread estimation by the estimator (ESD) proposed in [20] with $0.11MB$ on-chip memory.

C. Our Goal

We aim to fill the gap left by the prior art by addressing their limitations. Our goal is to design an efficient on-chip/off-chip spread estimator, which works with a small on-chip space and a larger off-chip memory to provide online accurate per-flow spread estimation that records flow labels, incurs very low query overhead, and achieves good performance for both large and small flows. Our estimator design should have the following performance guarantees.

Missed-Sampling

- *Bound:* Due to the probabilistic nature of sampling, some small flows may not be sampled. We want to ensure that the probability for a flow with spread greater than n not to be sampled is bounded by ϵ, where n and ϵ are user-specified parameters.
- *Relative and Absolute Error Bounds:* We want to ensure that the relative error of the flows with spreads greater than n is bounded by δ with probability $1-\epsilon$, and that the absolute error of the flows with spreads smaller than n is bounded by δ' with probability $1-\epsilon$, where n, δ, δ', and $\epsilon \in (0,1)$ are user-specified parameters.
- *Probabilistic Guarantee in Flow Classification:* We want to identify the flows whose spreads are greater than a threshold T with the following probabilistic guarantee as defined in [31]: Given a lower bound l and an upper bound h with $l < T < h$, the probability for a flow with spread greater than h not to be identified is bounded by $1-\alpha$, and probability for a flow with spread smaller than l to be mis-identified is bounded by β, where T, l, h, α and β are user-specified parameters.

III. Spread Estimation by Non-Duplicate Sampling

A. NetFlow and Packet Sampling

NetFlow [23], [24] and its non-Cisco equivalents enable routers to measure per-flow statistics (such as the number of packets and the number of bytes for every TCP flow during each epoch). Modern routers process packets at high speed through network processor chips and switching fabrics, whereas per-flow statistics are typically stored in off-chip memory. There is a mismatch between the rate at which packets are forwarded on-chip and the rate at which per-flow statistics are updated off-chip. Sampling is a common technique to compensate for such a mismatch. The sampling probability p may be determined based on the ratio of off-chip memory throughput and packet forwarding rate. Let r be the highest line rate and t be the achievable off-chip throughput for traffic measurement considering both on-chip processing and off-chip memory access/processing. We can set p to a constant $\frac{t}{r}$ and perform per-packet sampling, where each packet is selected independently with a probability of p for updating the statistics of the flow that the packet belongs to.

B. Packet Sampling and Spread Measurement

Packet sampling is fine for *per-flow size statistics* in terms of the number of packets or the number of bytes. For such statistics, every packet (byte) worths the same as any other. However, it is not suitable for *per-flow spread statistics*, where each packet carries an element, which may be new or a duplicate that has appeared in earlier packets and thus should not be sampled or recorded.

We use an example to show the difference between size measurement and spread measurement. Consider a flow of 1000 packets. Suppose we measure the flow size in the number of packets and perform packet sampling with $p = 0.1$. We use an off-chip counter r to record the number of sampled packets in this flow and estimate the flow size to be $\frac{r}{p}$. Now consider a flow of 1000 packets, each carrying an element. We want to measure the flow spread, *i.e.*, the number of distinct elements in the flow. Per-packet sampling with $p = 0.1$ will select about 100 packets for off-chip recording. However, in case that all 1000 packets carry the same element, doing so is completely unnecessary. We should instead sample at most one packet of the flow and perform at most a single off-chip recording because all other packets are duplicates. This will save off-chip memory throughput for other tasks such as higher sampling of other flows with more distinct elements.

For spread measurement, a flow of 1000 packets carrying the same element is equivalent to a flow of 1 packet carrying that element. The actual packet sampling rates of different flows should be different, depending on their densities of distinct elements, which are, however, unknown when sampling is performed on each individual packet.

C. Non-Duplicate Sampling

We introduce a new concept called *non-duplicate sampling*, which is to select each distinct element in any flow with a given probability p at its first appearance only, in contrast to traditional sampling that selects each packet with probability p.

Consider the previous flow of 1000 packets carrying the same element e. Still let $p = 0.1$. Traditional packet sampling will select about 100 packets and record the same element that many times, whereas non-duplicate sampling will select e with probability 0.1 when it appears for the first time — if it is selected, we record it; otherwise, we do not record it. All subsequent 999 appearances will be ignored.

Clearly, the implementation of non-duplicate sampling over a packet stream will still require sampling decision on a per-packet basis, but the method of traditional sampling will need to be replaced with something that will not only remember what we have seen so far (to avoid duplicates), but also possess the precision of ensuring that each distinct element has exactly a chance of p being selected, no matter how early (or late) the element appears in the packet stream and how many times it appears.

IV. Solutions for Non-Duplicate Sampling

Before we present our solution for non-duplicate sampling, we give a list of desired properties for such a solution.

TABLE I
NOTATIONS

Symbols	*Meaning*
B, m	bit array and its length
N	number of distinct elements during the measurement epoch
c	the counter recording the number of bits with value 1 in bit array B
p	the probability of recording a new element at its first appearance
p_*	first-stage sampling probability of the two-stage solution
k	number of hash functions for bloom filter in three-stage solution
p', p'', p'''	first-stage sampling probability, second-stage false positive rate, and third-stage sampling probability of the three-stage solution

First, because sampling operations are performed online on a per-packet basis, we want them to be simple. The simpler, the better. Second, existing sketch-based spread estimators (such as Bitmap [5], [32], [33], FM [34], HLL [3], [35], and virtual sketches [9], [36]) do not support efficient online queries. We will not use them in this paper, but prefer a solution that supports real-time access to the spread of any flow. Third, while the online operations are kept simple, they should still provide great flexibility in quantitatively controlling spread measurement in terms of missed-sampling bound, absolute error bound, relative error bound, and probabilistic guarantee in flow classification, as defined in Section II-C. Also, we list the variables and parameters used in this paper with Table I.

A. A Two-Stage Solution for Non-Duplicate Sampling

We first propose a two-stage solution for non-duplicate sampling. Its on-chip data structures include a bit array B of m bits and a counter c, which are all initialized to zeros at the beginning of each measurement epoch. The purpose of the bit array is two-fold: One is to filter out duplicates, and the other is to serve for the second-stage sampling.

The first stage is *element sampling* with a variable probability p_* whose value increases over time. Shortly, we will discuss how to adjust the value of p_* dynamically. Element sampling is performed as follows: From each arrival packet, the router extracts the flow label f and the element ID e. It performs a hash $h = H(e \oplus f)$, where H is a hash function whose range is $[0, X)$, and $\oplus$ is the XOR operator. If and only if $h < p_* X$, the element is selected (sampled) by the first stage. Regardless of whether an element is selected or not, it needs to be processed by the second stage.

The second stage is *element filtering*. The router hashes the element pseudo-randomly to a bit in B by computing $h_* = H_*(e \oplus f) \mod m$, where H_* is another independent hash function. There are two cases to consider: (1) If $B[h_*] = 0$, it means that the router never sees this element before. In this case, it sets $B[h_*]$ to 1, increases c by 1, and sends the flow label f to off-chip memory for recording if e is selected by the first stage. (2) If $B[h_*] = 1$, it means that the router has seen an element hashed to the bit. The element may be e from f or another one setting the same bit due to hash collision. We nonetheless filter out e of f and thus take no further action. Note that our goal is not to record each element at its first appearance, but to record it with a pre-set probability, which we will show the details below.

Consider an arbitrary element e from an arbitrary flow f. When it first appears in the packet stream, element sampling (first stage) has a probability of p_* to select the element, and element filtering (second stage) has a probability of $\frac{m-c}{m}$ to hash into a bit of zero, which will trigger off-chip recording if e is selected at the first stage. Combining the above two stages, the probability of recording a new element at its first appearance is $p_* \frac{m-c}{m}$. We want to set it to a constant value $p = \frac{t}{r}$, which is pre-determined based on the line rate r and the achievable off-chip throughput t as explained in Section III-B. Hence,

$$p_* \frac{m-c}{m} = p$$
$$p_* = \frac{mp}{m-c}. \quad (1)$$

The sampling probability p_* at the first stage increases as the number of recorded elements (i.e., c) increases. That is the reason for all elements to be processed by the second stage regardless of whether they are selected by the first stage. Otherwise, an element that was not selected for its first appearance will have chance to be selected (sampled) in its subsequent appearances as p_* increases, which is against the requirement of non-duplicate sampling that an element can only be sampled at its first appearance.

The maximum value of c should be limited to $m(1-p)$ when p_* becomes 1. The current measurement epoch will terminate after c reaches its maximum value. To avoid persistent premature epoch termination, we may double the bit array size m in the subsequent epoches until it is large enough to prevent c from reaching its maximum.

Note that all subsequent appearances of the same element e in flow f will be hashed to the same bit in B (which is already set to 1) and thus automatically be filtered out.

Consider the first appearance of a new element again. The probability for it to not be selected at the first stage is $1 - p_*$. The probability for it to be selected at the first stage but hashed to a bit of 1 is $p_* \frac{c}{m}$. Combining these two cases with (1), the probability for the element to not be recorded is

$$(1 - p_*) + p_* \frac{c}{m} = 1 - p, \quad (2)$$

which matches the expectation of non-duplicate sampling.

In the worst case, when packets arrive at the highest rate r and they all carry different elements, each having a probability of p being recorded off-chip (under the two-stage solution), the off-chip throughput will be $rp = t$, which is achievable.

B. An Extended Three-Stage Solution for Non-Duplicate Sampling

The proposed two-stage solution has two limitations. First, it needs to record all the elements in the packet stream since we want to ensure that elements can only be sampled at their first appearance. Actually, most elements will not be sampled when p is very small. If some elements can be removed from the packet stream before the element filtering stage, we will be more space-efficient by recording fewer in the bit array B. Second, recording each element by a single bit

in B is not the best way to separate new elements from duplicates, particularly when p is large. Bloom filter is a space-efficient probabilistic data structure for checking whether an element has been seen before or not. It is more suitable for filtering out the duplicates. To address the above limitations, we propose a new three-stage solution for non-duplicate sampling in the following.

The first stage is element pre-sampling with probability p', where p' is a pre-set stable value that will not be changed over time. We will discuss how to determine the value of p' shortly. Element pre-sampling is performed as follows: For each arrival packet, the router performs a hash $h' = H'(e \oplus f)$ where H' is a hash function whose range is $[0, X)$ and $\oplus$ is the XOR operator. If and only if $h' \leq p'X$, the element is selected (sampled) and enters into the second stage.

The second stage is duplicate filtering, which is used to find out the first appearance of the elements selected by the first stage. We use bloom filter to record the passing elements and filter out the duplicates in the three-stage solution. Suppose there are k independent hash functions H_j, $1 \leq j \leq k$, whose range is $[0, m-1)$. We perform the duplicate filtering as follows: For each element selected by the first stage, the router hashes this element pseudo-randomly to k bits from B through $H_j(e \oplus f)$, $1 \leq j \leq k$. There are two cases to consider: (1) If at least one of these k bits is zero, it means that the router never sees this element before. In this case, the router sets all the k bits to one and transfers this element to the third stage. (2) If all the k bits are ones, the router regards this element as one that has been already recorded in the bloom filter. In this case, router regards this element as a duplicate and drops it. However, bloom filter can make false positive claims. It is always true if it says an element has not been recorded, but it can be wrong if it says an element has been recorded. We use p'' to denote the false positive ratio of bloom filter B, which is the probability for an element to be falsely reported as a duplicate on its first arrival. We also use a counter c to record the number of ones in the bloom filter. When a bit of B is set from zero to one, we increase c by 1.

The third stage is element sampling with a variable probability p''', which is used to control the overall element sampling probability to a pre-set value p. For each element selected by the second stage, the router performs a hash $h'' = H''(e \oplus f)$, where H'' is a hash function whose range is $[0, X]$. If and only if $h'' \leq p'''X$, the element is selected (sampled) by the third stage, and the router sends the flow label f to the off-chip memory.

Consider an arbitrary element e from an arbitrary flow f. It has a probability of p' to be selected in the first stage, further has a probability of $(1-p'')$ to be selected in the second stage at its first appearance, and finally has a probability of p''' to be sampled and trigger off-chip recording at the third stage. Combining the above three stages, the probability of recording an element at off-chip memory is $p'(1-p'')p'''$. Our goal is to ensure that each element has a pre-set probability to be sampled at its first appearance. Hence, we have

$$p'(1-p'')p''' = p. \tag{3}$$

The space efficiency of the three-stage solution is controlled by two parameters, *i.e.*, p' and k. In the following, we will show how to get the optimal value of p' and k for different p.

Consider an arbitrary element e from an arbitrary flow f, where e is selected from the first stage. When it first appears in the packet stream, each hash function H_j has a probability of $\frac{c}{m}$ to hash it to a bit of one. The probability for all the k hash functions hash e to the bits of one is $(\frac{c}{m})^k$. If an element is not mis-reported as a duplicate, it will be selected by the second stage. Then, we have the probability for each element to be selected by the second stage when it appears at the first time is

$$1 - p'' = 1 - (\frac{c}{m})^k. \tag{4}$$

Since c increases over time, the value of $(1-p'')$ decreases over time. By applying (4) to (1), we have

$$p'(1-(\frac{c}{m})^k)p''' = p. \tag{5}$$

The sampling probability in the third stage p''' can be most 1, *i.e.*, $0 \leq p''' \leq 1$. Therefore, we have

$$\begin{aligned} p'(1-(\frac{c}{m})^k) &\geq p, \\ 1-(\frac{c}{m})^k &\geq \frac{p}{p'}. \end{aligned} \tag{6}$$

Suppose there are N distinct elements in the packet stream. After the first stage, the number of remaining distinct elements is Np'. When the number of recorded elements grows, c and $p'' = (\frac{c}{m})^k$ will also increase. Hence, the value of p'' is maximized after all Np' elements are recorded. Consider an arbitrary bit $B[i]$ in B, each recorded element has a probability of $(1-\frac{1}{m})^k$ not set $B[i]$ to one. Thus, the probability for $B[i]=0$ after recording Np' elements is $(1-\frac{1}{m})^{kNp'}$, and the probability that $B[i]=1$ after recording Np' elements is $(1-(1-\frac{1}{m})^{kNp'})$. Then, we have the false positive ratio p''

$$p'' = (\frac{c}{m})^k \leq (1-(1-\frac{1}{m})^{kNp'})^k. \tag{7}$$

Note that p and p' are pre-set values. Thus, we have the following inequality should hold through combining (6) and (7).

$$1 - \frac{p}{p'} \geq (1-(1-\frac{1}{m})^{kNp'})^k. \tag{8}$$

Since $(1-\frac{1}{m})^{kNp'}$ approximates to $e^{\frac{-kNp'}{m}}$, we further have

$$1 - \frac{p}{p'} \geq (1-e^{\frac{-kNp'}{m}})^k. \tag{9}$$

The smaller the $\frac{m}{N}$, the more space-efficient of the proposed mechanism. Our goal is to optimize the system parameters (p' and k) such that $\frac{m}{N}$ is minimized under constraints. The problem is formally defined as follows.

$$\begin{aligned} &Minimize \quad \frac{m}{N} \\ &Subject\ to \begin{cases} 1-\frac{p}{p'} \geq (1-e^{\frac{-kNp'}{m}})^k, \\ p < p' \leq 1, \\ k = \{1, 2, \ldots, m\}. \end{cases} \end{aligned} \tag{10}$$

The parameter p is determined by the performance objective. In Section VI, we will show how to get the optimal value of p based on different performance requirements. For any given value of p, we can get the optimal value of k and p' by solving the constrained optimization problem (10). The relationship between p and the optimal value of k and p' are shown in Fig. 2.

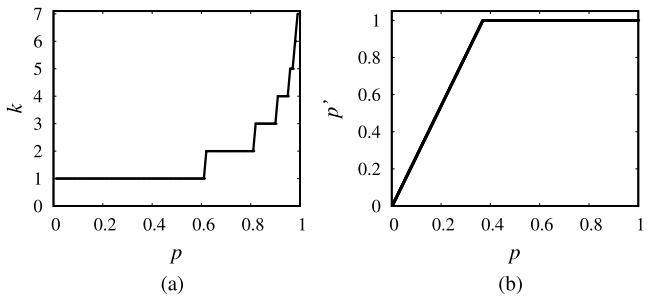


Fig. 2. The relationship between p and the optimal value of k and p'.

From Fig. 2, we found that the optimal value of k is equal to one when p is not large enough. In this case, the bloom filter in the second stage is a typical bitmap. By substituting $k = 1$ to (9), we have

$$\frac{p}{p'} \leq e^{\frac{-Np'}{m}}. \tag{11}$$

We further take logarithm on both sides of (11) and have

$$\ln(\frac{p}{p'}) \leq \frac{-Np'}{m},$$
$$\frac{m}{N} \geq \frac{1}{p'}(\ln p' - \ln p). \tag{12}$$

Through taking the first-order derivative on variable p', we find the minimum result of $\frac{m}{N}$ is obtained when $p' = pe$, where e is the natural logarithm with the value of 2.718. Then, the optimal value of $p' = \min\{1, pe\}$ when $k = 1$. Moreover, the optimal value of $p' = 1$ when $k > 1$ (as shown in Fig. 2(b)). Thus, we have

$$p' = \begin{cases} pe, & \text{if } p \leq \frac{1}{e}; \\ 1, & \text{otherwise.} \end{cases} \tag{13}$$

Next, we discuss the optimal value of k. According to the optimal theory of bloom filter, the optimal number of hash functions k (ignoring integrality) is

$$k = -\frac{\ln p''}{\ln 2}. \tag{14}$$

Notice that k should be an integer that no less than 1. Then, we have the optimal value of $k = 1$ when $p'' \leq 0.5$ based on (14). Combine with $p'' \leq p$, we can easily get that the optimal value of $k = 1$ when $p \leq 0.5$. When $p > 0.5$, the optimal value of $p' = 1$ and the maximum value of p'' is limited to $1 - p$ when p''' becomes 1. Thus, the problem of finding the optimal value of k is equal to the problem of finding the value of k that minimizes $\frac{m}{N}$ for a given $p'' = 1-p$. Then, we have

$$k = -\frac{\ln(1-p)}{\ln 2}. \tag{15}$$

Substituting $p' = 1$ to (9), we have

$$\text{1-p} \geq (1 - e^{\frac{-kN}{m}})^k, \tag{16}$$

and further have

$$\frac{m}{N} \geq -\frac{k}{\ln(1-(1-p)^{\frac{1}{k}})}. \tag{17}$$

Let $A_1 = -\frac{k}{\ln(1-(1-p)^{\frac{1}{k}})}$ when $k = \lceil -\frac{\ln(1-p)}{\ln 2} \rceil$, and $A_2 = -\frac{k}{\ln(1-(1-p)^{\frac{1}{k}})}$ when $k = \lfloor -\frac{\ln(1-p)}{\ln 2} \rfloor$. Actually, the minimum value of $\frac{m}{N}$ that our estimator can achieve is $\min\{A_1, A_2\}$. Denote the optimal value of k by k^*, which can be computed by

$$k^* = \begin{cases} 1, & \text{if } p \leq 0.5; \\ \lceil -\frac{\ln(1-p)}{\ln 2} \rceil, & \text{if } p > 0.5 \text{ and } A_1 < A_2; \\ \lfloor -\frac{\ln(1-p)}{\ln 2} \rfloor, & \text{if } p > 0.5 \text{ and } A_2 \leq A_1. \end{cases} \tag{18}$$

Let $\left(\frac{m}{N}\right)^*$ be the optimal value of $\frac{m}{N}$. With the optimal value of k and p', we have

$$\left(\frac{m}{N}\right)^* = \begin{cases} ep, & \text{if } p \leq \frac{1}{e}; \\ -\frac{1}{\ln p}, & \text{if } \frac{1}{e} < p \leq 0.5; \\ -\frac{k^*}{\ln(1-(1-p)^{\frac{1}{k^*}})}, & \text{if } p > 0.5. \end{cases} \tag{19}$$

As p''' can be most 1, $p'(1 - (\frac{c}{m})^k) \geq p$ should always stands. Thus, the maximum value of c should be limited to $m \cdot \sqrt[k]{(1 - \frac{p}{p'})}$ when p''' becomes 1. The current measurement epoch has to terminate after c reaches its maximum value. To avoid premature epoch termination, we may double the bit array size m until it is large enough to prevent c from reaching its maximum.

Consider the first appearance of a new element again. The probability for it to not be selected at the first stage is $1 - p'$. The probability for it to be selected at the first stage but misreported as a duplicate at the second stage is $p'p''$. The probability for it to be selected both at the first and the second stages but not selected at the third stage is $p'(1-p'')(1-p''')$. Combining these three cases with (1), the probability for the element to not be recorded is

$$(1 - p') + p'p'' + p'(1 - p'')(1 - p''') = 1 - p, \tag{20}$$

which matches the expectation of non-duplicate sampling.

V. Our Design With Non-Duplicate Sampling

With the solutions for non-duplicate sampling, we propose two spread estimators that can answer the online query in real time. The details of our design are as shown in Algorithm 1.

We want to stress that our bit-array filter B serves a different purpose from the bitmaps in [2], [5]. Our filter is used to assist sampling, whereas those bitmaps are used as per-flow data structures in spread estimation.

In fact, they could be complementary to each other, with ours for on-chip duplicate removal and theirs as off-chip data structures for recording flow elements. However, bitmaps have limited estimation ranges. More sophisticated sketches, such as FM [34], HLL [3], [35], and their virtualized versions [9], [36], have very large ranges. But all these sketches (including bitmaps) do not support online queries because they are expensive to compute, having to synthesize data from hundreds or thousands of bits or registers.

In order to support efficient online queries, we opt not to use these sketches as our off-chip data structures. We observe that, after non-duplicate sampling, each time an element from f is recorded, it must be a new one that is not seen before, which

Algorithm 1 Non-Duplicate Sampling

```
for each arrival packet in packet stream do
    Extract flow label f and element label e;
    Using two-stage or three-stage solution to sample
    each element < f, e > at its first appearance;
    if an element < f, e > is sampled then
        Send flow label f to off-chip recording;
        if label f is downloaded for the first time then
            Initialize flow f's table entry c_f to 0;
        end
        Set c_f = c_f + 1;
    end
end
for each flow f do
    Estimate its flow spread by n̂_f = c_f / p;
end
```

is why we only send the flow label f off-chip for recording. Our off-chip data structures include a hash table to store the flow labels, each with a counter. When we record f for the first time, it is inserted into the hash table with its counter value c_f set to 1. After that, when f is recorded again (because its other elements are sampled), we find its entry in the hash table and increase the counter by 1.

For an online query on the spread of flow f, we only need to hash f to find its table entry and return its current counter value c_f divided by the sampling probability p.

The simple operations of non-duplicate sampling have an immediate benefit of online efficiency, both in spread measurement and real-time queries. Nevertheless, simplicity does not necessarily limit functionality. We show in the next section that the proposed spread estimator can be flexibly configured for various probabilistic performance guarantees.

VI. Performance Analysis

In this section, we show how to configure the system parameters of the proposed estimator for different probabilistic performance guarantees, such as the flow miss-sampling bound, relative and absolute error bound, and probabilistic guarantee in flow classification.

A. Miss-Sampling Bound

A flow will miss if none of its elements are sampled. Define the miss-sampling probability as the probability for a flow miss. Given two values n and ϵ, our design can ensure that the miss-sampling probability for a flow with spread greater than n is bounded by ϵ through proper system parameter configuration.

Consider an arbitrary flow f with spread n_f. Each element of f has a probability of p to be sampled, and then the probability for none of f's elements to be sampled is $(1-p)^{n_f}$. Since $(1-p)^n \geq (1-p)^{n_f}$ when $n_f \geq n$, our design can bound the miss-sampling probability within ϵ for flows with spread greater than n if $(1-p)^n \leq \epsilon$, i.e.,

$$p \geq 1 - \epsilon^{1/n}. \tag{21}$$

We want to stress that our solutions ensure that all the distinct elements are sampled with the same probability, no matter how early (or late) the element appears in the packet stream and how many times it appears. Thus, the miss-sampling ratio of a flow has no relationship with the arriving time of the flow (or burst). It will diminish exponentially as n_f increases.

Let N be the total number of distinct elements in the current epoch. For our two-stage solution, all of them will have set their hashed bits in the filter at the end of the epoch, and from Section IV-A we know that the number c of bits that are ones in the filter reaches its maximum value of $m(1-p)$. The percentage of bits in the filter that are zeros is thus $\frac{m-c}{m} = p$. According to [12], [32], N can be approximated as $-m \ln p$. From (21), we should set m as

$$m = -\frac{N}{\ln p} \geq -\frac{N}{\ln(1-\epsilon^{1/n})}. \tag{22}$$

The value of N can be estimated based on the measurements in prior epoches, for example, as the moving average of the prior measurements, each being the total number of sampled elements in an epoch divided by p. Let $\hat{N}$ be such an estimate. Substituting N with its estimate, we can practically set m as follows:

$$m \geq -\frac{\hat{N}}{\ln(1-\epsilon^{1/n})}. \tag{23}$$

For our three-stage solution, we can get the minimum value of m by substituting N with its estimate to (19),

$$m = \begin{cases} \hat{N}ep, & \text{if } p \leq \dfrac{1}{e}; \\ -\dfrac{\hat{N}}{\ln p}, & \text{if } \dfrac{1}{e} < p \leq 0.5; \\ -\dfrac{\hat{N}k^*}{\ln(1-(1-p)^{\frac{1}{k^*}})}, & \text{if } p > 0.5, \end{cases} \tag{24}$$

where $p = 1 - \epsilon^{1/n}$, k^* is the optimal value of k obtained by (18).

B. Relative and Absolute Error Bounds

We now show how to configure the system parameters to bound the relative and absolute errors of the proposed estimator.

Consider an arbitrary flow f. Let c_f be the counter value of f's table entry and $Pr\{c_f = k\}$ be the probability for $c_f = k$. We further consider an arbitrary subset $S_{f,k}$ of k elements of flow f; there are $C_{n_f}^k$ ways to form such a subset. Denote the complement of the subset as $S_{f,-k}$, which consists of all elements of flow f that are not included in $S_{f,k}$. The probability for all of the elements in $S_{f,k}$ to be sampled is p^k, and the probability for no element in $S_{f,-k}$ to be sampled is $(1-p)^{n_f-k}$. Then, we have the probability for all of the elements in $S_{f,k}$ to be sampled and all of elements in $S_{f,-k}$ that are not to be sampled, namely $p^k(1-p)^{n_f-k}$. Note that there are $C_{n_f}^k$ ways to form a subset $S_{f,k}$. Thus, we have

$$Pr\{c_f = k\} = C_{n_f}^k p^k (1-p)^{n_f-k}. \tag{25}$$

Suppose we want to ensure the relative (absolute) error of flows whose spreads are greater (smaller) than n is bounded by δ (δ') with probability $1-\epsilon$.

Since each sampled element will be estimated as $\frac{1}{p}$ elements, $\lceil (n_f - \delta')p \rceil \leq c_f \leq \lfloor (n_f + \delta')p \rfloor$ should stand if we want to bound the absolute error of flow f within δ'.

Based on (25), the probability for the estimated spread of flow f not distributed in $[n_f - \delta', n_f + \delta']$ is

$$p_2 = 1 - \sum_{j=\lceil (n_f-\delta')p \rceil}^{\lfloor (n_f+\delta')p \rfloor} C_{n_f}^j p^j (1-p)^{n_f-j}. \quad (26)$$

Therefore, the absolute error of flow f is bounded by δ' with probability $1-\epsilon$ if the following inequality stands:

$$1 - \sum_{j=\lceil (n_f-\delta')p \rceil}^{\lfloor (n_f+\delta')p \rfloor} C_{n_f}^j p^j (1-p)^{n_f-j} \leq \epsilon. \quad (27)$$

Note that p_2 is increased with increasing n_f, which means the flow with large spread has a high probability to have a greater absolute error than the flow with small spreads. Thus, we can obtain the optimal value of p by solving (28), which ensures that the absolute error of the flows whose spreads are smaller than n is bounded by δ' with probability $1-\epsilon$:

$$1 - \sum_{j=\lceil (n-\delta')p \rceil}^{\lfloor (n+\delta')p \rfloor} C_{n}^j p^j (1-p)^{n-j} = \epsilon. \quad (28)$$

Similar to the analysis for the absolute error bound, we can obtain the optimal value of p by solving (29), which ensures the relative error of the flows whose spreads are greater than n is bounded by δ with probability $1-\epsilon$:

$$1 - \sum_{j=\lceil (1-\delta)np \rceil}^{\lfloor (1+\delta)np \rfloor} C_{n}^j p^j (1-p)^{n-j} = \epsilon. \quad (29)$$

C. Upper Bounds of Relative and Absolute Errors

Next, we analyze the upper bounds of the relative and absolute errors.

Let E_a^f be the absolute error of the proposed estimator for flow f. Let A_1 be the event that all of the elements of f are sampled, and A_2 be the event that none element of f is sampled. Then, we have that the upper bound of E_a^f is equal to $\max\{E_{a,A_1}^f, E_{a,A_2}^f\}$, where $E_{a,A_1}^f (E_{a,A_2}^f)$ is the value of E_a^f when event $A_1 (A_2)$ happens. The probability for A_1 happens is

$$P_{A_1,f} = p^{n_f}. \quad (30)$$

When A_1 happens, the absolute error of each element of flow f that will bring is $\frac{1}{p} - 1$. Then, we have

$$E_{a,A_1}^f = n_f(\frac{1}{p} - 1). \quad (31)$$

For a given flow f, the probability that A_2 happens is

$$P_{A_2,f} = (1-p)^{n_f}. \quad (32)$$

When A_2 happens, the spread of f will be estimated as zero. Thus, we have $E_{a,A_2}^f = n_f$.

Combined with the analysis above, we have the upper bound of the absolute error of the proposed estimator for flow f,

$$\max\{E_{a,A_1}^f, E_{a,A_2}^f\} = \max\{n_f(\frac{1}{p} - 1), n_f\}, \quad (33)$$

and, further, we have the upper bound of the relative error of the proposed estimator for flow f,

$$\max\{\frac{E_{a,A_1}^f}{n_f}, \frac{E_{a,A_2}^f}{n_f}\} = \max\{\frac{1}{p} - 1, 1\}. \quad (34)$$

Note that the upper bound of the absolute and relative errors for flow f are, respectively, equal to n_f and 1 when $p \geq 0.5$.

D. Probabilistic Guarantee in Flow Classification

In some applications, such as DDoS detection and scanner detection, we must monitor the flows with abnormal spreads, *i.e.*, identify all of the flows whose spreads exceed a certain threshold in each measurement period, where the threshold is a system parameter. In other words, we want to classify flows into two types based on whether their spreads are abnormally large or not. Since the limited SRAM only allows us to record part of the information of each flow, a precise flow classification is not feasible [5], [31], [33]. Thus, we adopt the probabilistic performance objective from [31].

Let h and l be two positive integers, and $\hat{n}_f$ be the estimated value of n_f. The objective is to identify the flows whose spreads are greater than a threshold T with the following probabilistic guarantees: identify any flow whose spread is h or larger with a probability no less than α and identify any flow whose spread is l or smaller with a probability no more than α, where $l < T < h$. There are two kinds of false identification. The first one identifies flow f if $n_f \leq l$, which is defined as a false positive. The second one is non-identification when $n_f \geq h$, which is treated as a false negative. Then, the objective can be expressed in terms of conditional probabilities:

$$\begin{aligned} &\Pr\{\text{identify} f \text{ as an abnormal flow } | n_f \leq l\} \leq \beta, \\ &\Pr\{\text{mis-identify} f \text{ as an abnormal flow } | n_f \geq h\} \geq \alpha, \end{aligned} \quad (35)$$

where β is the false positive probability and $1-\alpha$ is the false negative probability. The above objective is to bound the false positive ratio by β and the false negative ratio by $1-\alpha$. In the following, we show that the proposed estimator can achieve the above objective by proper parameter settings.

Let $Pr\{c_f \leq k\}$ ($Pr\{c_f \geq k\}$) be the probability for $c_f \leq k$ ($c_f \geq k$). Based on (25), we have:

$$\begin{aligned} Pr\{c_f \leq k\} &= \sum_{j=0}^{k} C_{n_f}^j p^j (1-p)^{n_f-j}, \\ Pr\{c_f \geq k\} &= \sum_{j=k}^{n_f} C_{n_f}^j p^j (1-p)^{n_f-j}. \end{aligned} \quad (36)$$

Given a threshold T, the flow with an estimated spread no less than T will be identified as an abnormal one. Since the spread of a flow will be estimated as no less than T when $c_f \geq \lceil Tp \rceil$, the probability for a flow f to be identified as an abnormal one is:

$$Pr\{c_f \geq \lceil Tp \rceil\} = \sum_{j=\lceil Tp \rceil}^{n_f} C_{n_f}^j p^j (1-p)^{n_f-j}. \quad (37)$$

TABLE II
MISS-SAMPLING RATIO OF OUR DESIGN

m (MB)	solution	spread						
		$1 \sim 3$	$4 \sim 6$	$7 \sim 10$	$11 \sim 20$	$21 \sim 50$	$51 \sim 100$	$101 \sim$
0.1	two-stage ($p = 0.02$)	0.9714	0.8952	0.8259	0.7149	0.5064	0.2184	0.0241
	three-stage ($p = 0.10$)	0.8822	0.6213	0.4415	0.2428	0.0554	0.0024	0.0000
0.5	two-stage ($p = 0.47$)	0.4758	0.0585	0.0065	0.0005	0.0000	0.0000	0.0000
	three-stage ($p = 0.47$)	0.4752	0.0571	0.0073	0.0004	0.0000	0.0000	0.0000
2	two-stage ($p = 0.82$)	0.1434	0.0005	0.0000	0.0000	0.0000	0.0000	0.0000
	three-stage ($p = 0.92$)	0.0642	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

To achieve the above objectives, the following should be satisfied:

$$\begin{cases} \sum\limits_{j=\lceil Tp \rceil}^{n_f} C_{n_f}^j p^j (1-p)^{n_f - j} \leq \beta, \quad \forall n_f \leq l; \\ \sum\limits_{j=\lceil Tp \rceil}^{n_f} C_{n_f}^j p^j (1-p)^{n_f - j} \geq \alpha, \quad \forall n_f \geq h. \end{cases} \tag{38}$$

By solving (38), we can obtain the minimum value of p that satisfies the above constraints. Note that the upper bound of absolute error for a flow with a spread less than l is $l(\frac{1}{p} - 1)$ if the spread of this flow is overestimated. Then, the probability for f to be identified as an abnormal flow is zero if we set $T \geq \frac{l}{p}$, *i.e.*, $Pr\{\text{identify } f \text{ as an abnormal flow } | n_f \leq l\} = 0$ when $h \geq \frac{l}{p}$.

VII. EXPERIMENTAL RESULTS

We use five minutes of data downloaded from CAIDA [22] as our dataset. This dataset has 1689780 distinct per-source flows, 3150740 distinct elements, and 152163629 packets. Our goal is to estimate the spread of per-source flows in this dataset. In the experiments, we use ESD [20] and KPSE [13] as baselines, both of which are sketch-based methods. Their core idea is to allocate each flow a virtual bitmap and let different flows' virtual bitmaps share their bits uniformly.

A. Implementation

We implement the proposed two-stage solution and three-stage solution on an XLINX ZYBO-7020 SOC development board, with 53200 logic units, 5040 Kbits Block RAM, and a clock rate of 100MHz. The Arm part of this board contains a 667MHz dual-core Cortex-A9 processor. We use the FPGA part as the hardware and the Arm part as the software. The hardware processing speed is 100Mpps, and the communication speed among these two parts is about 1.6Mpps.

In our implementation, the throughput threshold of the on-chip sampling module and the off-chip memory are separately 100MHz and 1.6MHz. The distinct element arriving rate of our implementation is 2.86MHz. To keep up with the line speed, the element offloading speed should be no more than the threshold of the off-chip throughput, *i.e.*, multiplying 2.86MHz with p should be no more than 1.6MHz. Then, we have $p \leq 0.55$.

Further, we test the query throughput of our solutions, ESD, and KPSE, where query throughput refers to the number of queries an estimator can answer within one second. The evaluations are executed by running the off-chip spread estimation of these estimators on a machine with an Intel(R) Core(TM) i5-4590 @3.3GHz CPU and 16 GB memory. Notice that the off-chip recording and estimation process of our solutions are the same. Thus, our two-stage solution and three-stage solution have the same query throughput. The experimental results are presented in Fig. 3, which shows that our solutions increase the query throughput by around three orders of magnitude greater than the existing ones. Actually, our solutions can answer the query for the spreads of all 9090784 flows in 3 seconds (less than 1μ s per-flow), which is efficient enough for any online queries.

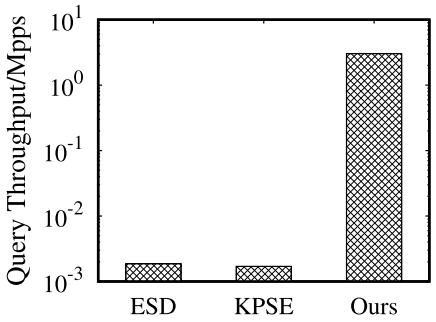


Fig. 3. Query throughput.

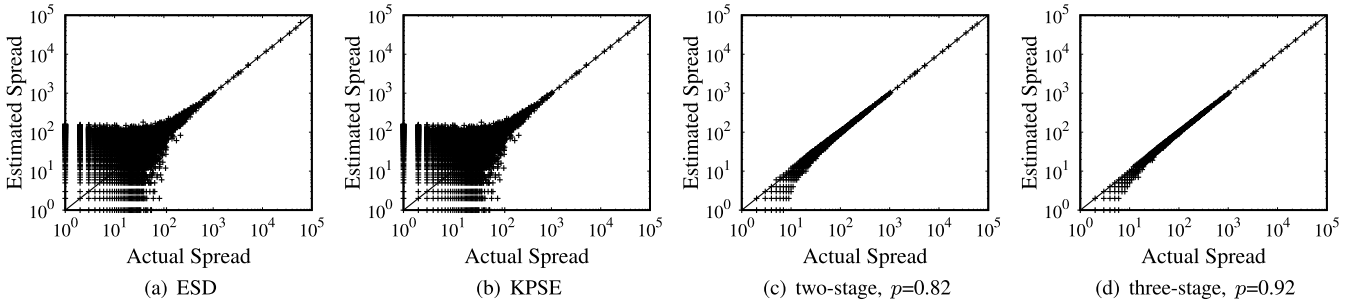
B. Miss-Sampling Ratio

TABLE II shows the performance of our solutions on miss-sampling ratio for the flows with different spreads when m is equal to 0.1MB, 0.5MB, and 2MB. As p or the spread of flow increases, the miss-sampling ratio decreases quickly, which verifies our theoretical analysis. Consider a case in which we want to bound the miss-sampling probability for flows with spread larger than 50 within 0.01. Based on (21), we have that p should be no less than 0.09. The actual miss-sampling rate of our two-stage solution and three-stage solution are 0.0012 and 0.0024 respectively when $p = 0.09$, which is much smaller than the theoretical bound.

C. Estimation Accuracy

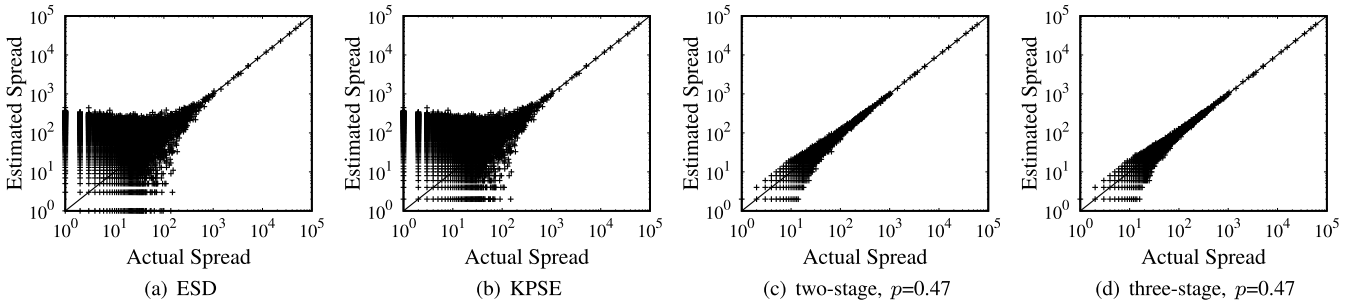
In this part of the experiments, we evaluate the performance of our solutions and compare them with ESD and KPSE under different sizes of on-chip memory.

Fig. 4 - Fig. 6, TABLE IV and TABLE V compare our solutions, ESD, and KPSE on spread estimation accuracy. In this set of experiments, we set the parameter p of our solutions to the optimal value under the given m, and the size of the virtual bitmap of ESD and KPSE to the minimum value that can supply a large enough estimation range for all



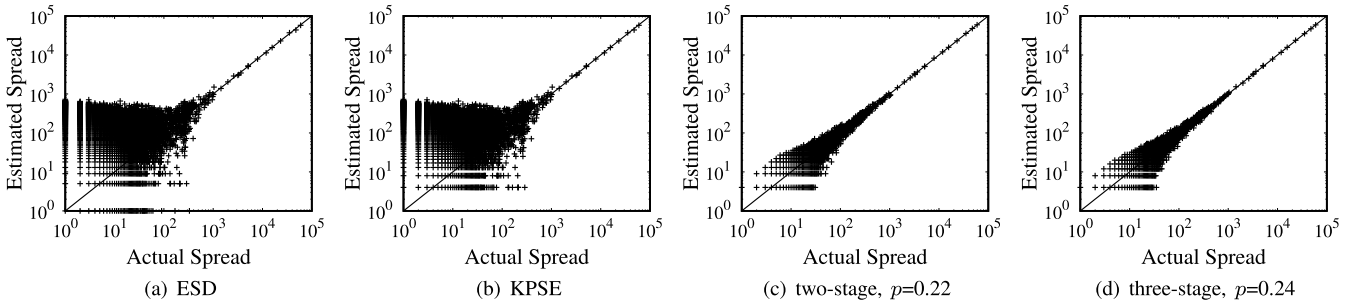
(a) ESD (b) KPSE (c) two-stage, p=0.82 (d) three-stage, p=0.92

Fig. 4. Spread estimation accuracy of ESD, KPSE and our estimator when $m = 2$MB.



(a) ESD (b) KPSE (c) two-stage, p=0.47 (d) three-stage, p=0.47

Fig. 5. Spread estimation accuracy of ESD, KPSE and our estimator when $m = 0.5$MB.



(a) ESD (b) KPSE (c) two-stage, p=0.22 (d) three-stage, p=0.24

Fig. 6. Spread estimation accuracy of ESD, KPSE and our estimator when $m = 0.25$MB.

TABLE III

MEMORY REQUIREMENTS OF TWO-STAGE ESTIMATOR AND THREE-STAGE ESTIMATOR (MB)

p	0.01	0.05	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.99
two-stage	0.09	0.13	0.17	0.24	0.32	0.41	0.55	0.74	1.06	1.69	3.57	37.38
three-stage	0.02	0.06	0.11	0.21	0.31	0.41	0.55	0.74	0.95	1.27	1.81	3.61

of the flows. The experimental results show that our solutions work much better than the existing ones. In detail, when flow spread is less than 100 or larger than 1000, our methods show significant advantages in estimation accuracy compared to ESD and KPSE. We want to stress that, due to the properties of probabilistic counting algorithm, ESD and KPSE show better performance on the flows whose spreads are within $[100, 1000]$. Even for those flows, our algorithm can still outperform the baselines in estimation accuracy. This is because the existing studies use on-chip memory to store the traffic data for estimation, which requires aggressive space sharing and further results in significant errors for small/medium flows. However, our solutions only use on-chip memory to filter out the duplicates and help sample the passing element, but stores the traffic data in off-chip memory. Thus, our solutions can work in a much smaller on-chip memory while achieving higher estimation accuracy than the existing studies.

From the experimental results, we also found that ESD and KPSE fail to obtain an accurate estimation for flows with spreads less than 100 when the allocated on-chip memory is 5.32 bit per element ($m = 2$MB), for flows with spreads less than 200 when the allocated on-chip memory is 1.33 bit per element ($m = 0.5$MB), and for flows with spreads less than 1000 when the allocated on-chip memory is 0.67 bit per element ($m = 0.25$MB). Our solutions are clearly the winner, especially for the flows with small or middle spread. Moreover, the experimental results also show that the three-stage solution performs better than the two-stage solution. When p is large or small enough, our three-stage solution needs less on-chip memory than our two-stage solution (as shown in TABLE III). That is because bloom filter is more efficient than bitmap when p is large enough, *i.e.*, the threshold of the false positive ratio is small, and the first stage of our three-stage solution can reduce the on-chip recorded elements, which can help us to

TABLE IV

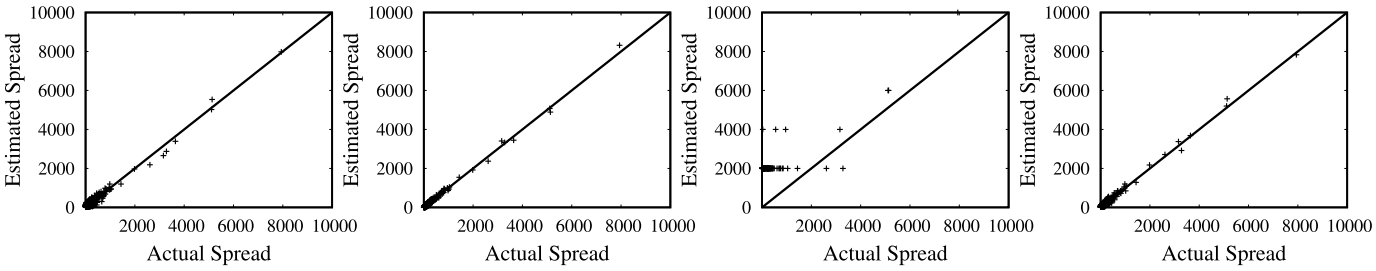
MEAN ABSOLUTE ESTIMATION ERROR

spread	all flows				$1 \sim 100$				$101 \sim 1000$				$1001 \sim$			
algorithm / p	ESD	KPSE	two-stage	three-stage	ESD	KPSE	two-stage	three-stage	ESD	KPSE	two-stage	three-stage	ESD	KPSE	two-stage	three-stage
0.05	23.32	22.86	2.74	2.30	23.17	22.71	2.72	2.29	78.11	78.37	51.25	32.89	13024.15	13025.24	339.00	214.26
0.1	20.40	19.97	2.41	2.13	20.26	19.83	2.40	2.12	57.12	57.32	33.57	25.77	12845.67	12846.50	215.12	162.02
0.3	11.17	10.90	1.62	1.62	11.05	10.77	1.61	1.61	31.25	31.28	17.94	18.08	11622.36	11622.68	153.03	87.44
0.5	7.57	7.42	1.09	1.09	7.45	7.30	1.09	1.09	21.41	21.39	12.72	12.08	11538.05	11538.25	65.71	65.24
0.7	5.22	5.14	0.68	0.64	5.10	5.02	0.68	0.64	15.86	15.86	7.87	6.65	11479.76	11479.90	40.94	59.85
0.9	3.64	3.61	0.26	0.02	3.53	3.51	0.26	0.02	9.78	9.78	3.98	1.23	10328.59	10328.65	23.84	8.46

TABLE V

MEAN RELATIVE ESTIMATION ERROR

spread	all flows				$1 \sim 100$				$101 \sim 1000$				$1001 \sim$			
algorithm / p	ESD	KPSE	two-stage	three-stage	ESD	KPSE	two-stage	three-stage	ESD	KPSE	two-stage	three-stage	ESD	KPSE	two-stage	three-stage
0.05	19.60	19.20	1.85	1.66	19.60	19.20	1.85	1.66	0.34	0.34	0.25	0.16	0.88	0.88	0.05	0.03
0.1	17.12	16.75	1.71	1.57	17.13	16.76	1.71	1.57	0.27	0.27	0.17	0.13	0.82	0.82	0.03	0.03
0.3	9.28	9.04	1.25	1.25	9.28	9.04	1.25	1.26	0.15	0.15	0.09	0.09	0.55	0.55	0.02	0.01
0.5	6.22	6.09	0.87	0.87	6.22	6.09	0.87	0.87	0.10	0.10	0.06	0.06	0.54	0.54	0.01	0.01
0.7	4.23	4.16	0.53	0.50	4.23	4.16	0.53	0.50	0.07	0.07	0.04	0.03	0.54	0.54	0.01	0.01
0.9	2.91	2.89	0.18	0.01	2.91	2.89	0.18	0.01	0.05	0.05	0.02	0.01	0.39	0.39	0.01	0.00



(a) two-stage, m=0.1MB, (b) three-stage, m=0.1MB, (c) two-stage, m=0.05MB, (d) three-stage, m=0.05MB,

Fig. 7. Spread estimation accuracy two-stage estimator and three-stage estimator when $m = 0.1$MB or 0.05MB.

get a more space-efficient spread estimator when $p' < 1$. The experimental results as shown in Fig. 7 shows that, the estimation error of our three-stage solution is much smaller than our two-stage solution when p is small enough, which also verifies our theoretical analysis. Specially, the element sampling rates of our two-stage solution and three-stage solution are 0.0005 and 0.05 respectively when $m = 0.05$MB. In such a small on-chip memory, our two-stage solution cannot estimate the spreads of flows accurately. However, our three-stage solution still works well in this case (see Fig. 7(d)), which further verify the space-efficiency of our three-stage solution.

The mean absolute estimation error and the mean relative estimation error for the flows under different distribution when $p = 0.05, 0.1, 0.3, 0.5, 0.7$, and 0.9 are separately shown in Table. IV and Table. V. Both of our solutions perform much better than ESD and KPSE. In this set of experiments, we set m to the optimal value of our two-stage solution for all the comparison estimators. They reduce the mean absolute error and relative error of all flows by around one order of magnitude compared to the prior art. It is worth pointing out that they reduce the mean absolute error of flows with spread no less than 1000 by around two orders of magnitude compared to the prior art.

We then show the performance of our solutions on bounding the absolute and relative errors. Given a set of bounds, we can obtain the optimal value of p based on (27) and (29). TABLE VI shows the optimal values of p under different settings. The second to sixth columns present the optimal value of p that can ensure that the absolute errors of the flows with a spread smaller than n are bounded by δ' with probability 99%, and from the seventh to eleventh columns present the optimal value of p that can ensure that the relative errors of the flows with spreads greater than n are bounded by δ with probability 99%. From this table, we found that our solutions can bound the relative error (absolute error) in 25% (250) for the flows with spreads greater (smaller) than 1000 with probability 99% when $p = 0.1$. However, the mean relative error of ESD is 82% as shown in TABLE V, which indicates that our solutions not only has a higher accuracy than the existing ones in the small/medium flow spread estimation, but also in large flow spread estimation.

D. Flow Mis-Classification Probability

Finally, we compare the flow mis-classification probability of our solutions and ESD, which are what ESD was designed for. The first set of experiments compares our solutions and ESD for the amount of memory that they need to satisfy the constraints given in (38). We set $T = (h + l)/2$. TABLE VII shows the memory requirements of our solutions and ESD with respect to α, β, h, and l, which were computed according to the methods proposed in [20] and this work. However, ESD has a limited estimation accuracy for the flows with small spreads. We cannot obtain the required memory space of ESD when h is too small or the gap between h and l is too small. Hence, we use a short bar to indicate the required memory space of ESD in this case.

TABLE VI
OPTIMAL VALUE OF p UNDER DIFFERENT SETTINGS ($\epsilon = 0.01$)

n	absolute error					relative error				
	$\delta' = 50$	$\delta' = 100$	$\delta' = 150$	$\delta' = 200$	$\delta' = 250$	$\delta = 0.05$	$\delta = 0.1$	$\delta = 0.15$	$\delta = 0.2$	$\delta = 0.25$
200	0.34	0.11	0.06	0.03	0.02	0.92	0.76	0.58	0.45	0.34
500	0.57	0.25	0.13	0.08	0.05	0.84	0.57	0.37	0.25	0.17
1000	0.73	0.40	0.23	0.14	0.10	0.73	0.40	0.23	0.14	0.10
1500	0.80	0.50	0.31	0.20	0.14	0.64	0.31	0.17	0.10	0.07
2000	0.84	0.57	0.37	0.25	0.18	0.57	0.25	0.13	0.08	0.05

TABLE VII
MEMORY REQUIREMENTS OF OUR ESTIMATOR AND ESD (MB)

	h	$l = 0.5h$			$l = 0.7h$			$l = 0.8h$		
		ESD	two-stage	three-stage	ESD	two-stage	three-stage	ESD	two-stage	three-stage
$\alpha = 0.9, \beta = 0.1$	20	–	0.60	0.60	–	1.94	1.37	–	6.58	2.26
	50	1.11	0.31	0.30	–	0.78	0.78	–	1.49	1.18
	100	0.45	0.21	0.16	3.76	0.40	0.40	–	0.76	0.76
	200	0.23	0.15	0.09	0.78	0.26	0.24	10.87	0.44	0.44
	300	0.16	0.14	0.06	0.48	0.21	0.18	1.89	0.34	0.34
	500	0.10	0.11	0.04	0.29	0.17	0.11	0.77	0.25	0.23
	1000	0.05	0.09	0.02	0.15	0.13	0.06	0.38	0.18	0.13
	2000	0.03	0.08	0.01	0.08	0.11	0.03	0.19	0.14	0.07
$\alpha = 0.95, \beta = 0.05$	20	–	0.93	0.87	–	3.00	1.67	–	6.58	2.26
	50	4.14	0.42	0.42	–	1.12	0.98	–	2.22	1.46
	100	0.82	0.27	0.25	–	0.55	0.55	–	1.10	0.97
	200	0.37	0.19	0.14	2.04	0.35	0.34	–	0.63	0.63
	300	0.25	0.16	0.10	0.90	0.28	0.26	–	0.47	0.47
	500	0.15	0.13	0.06	0.48	0.21	0.17	2.80	0.34	0.33
	1000	0.08	0.11	0.03	0.24	0.16	0.10	0.99	0.23	0.20
	2000	0.04	0.09	0.02	0.12	0.12	0.05	0.49	0.17	0.11

TABLE VIII
FPR AND FNR OF OUR ESTIMATOR AND ESD WHEN $m = 0.2MB$

	h	$l = 0.5h$			$l = 0.7h$			$l = 0.8h$		
		ESD	two-stage	three-stage	ESD	two-stage	three-stage	ESD	two-stage	three-stage
FPR	20	3.53e-01	2.54e-03	4.49e-03	3.54e-01	3.63e-03	1.70e-03	3.54e-01	4.05e-03	2.02e-03
	50	1.81e-01	2.90e-04	9.91e-05	1.48e-01	3.34e-04	2.02e-04	1.48e-01	4.34e-04	3.02e-04
	100	3.12e-02	2.19e-05	8.29e-06	2.26e-02	5.33e-05	2.72e-05	1.61e-02	4.68e-05	2.66e-05
	200	3.34e-04	5.92e-07	1.18e-06	6.93e-05	4.14e-06	3.55e-06	4.56e-05	4.74e-06	2.96e-06
	300	4.74e-06	0.00e+00	0.00e+00	4.14e-06	2.37e-06	1.18e-06	3.55e-06	2.37e-06	2.37e-06
	500	0.00e+00	0.00e+00	0.00e+00	5.92e-07	0.00e+00	0.00e+00	1.18e-06	1.78e-06	0.00e+00
FNR	20	3.11e-01	1.64e-01	7.82e-02	3.11e-01	1.64e-01	1.75e-01	3.11e-01	1.64e-01	1.75e-01
	50	1.51e-01	5.00e-02	4.29e-02	1.82e-01	8.21e-02	8.07e-02	1.82e-01	8.21e-02	8.07e-02
	100	6.85e-02	1.44e-02	9.01e-03	7.39e-02	2.70e-02	2.34e-02	8.29e-02	4.50e-02	3.96e-02
	200	5.57e-02	0.00e+00	3.48e-03	1.11e-01	1.39e-02	1.39e-02	1.50e-01	2.79e-02	2.44e-02
	300	2.73e-02	0.00e+00	9.09e-03	1.18e-01	0.00e+00	1.82e-02	1.82e-01	9.09e-03	2.73e-02
	500	4.76e-02	0.00e+00	0.00e+00	1.90e-01	0.00e+00	2.38e-02	3.33e-01	0.00e+00	2.38e-02

For the setting of $\alpha = 0.9$, $\beta = 0.1$, and $\alpha = 0.95$, $\beta = 0.05$, we found that ESD requires more on-chip memory than our solutions require when h is small, which indicates the space-efficiency of our solutions for classifying flows with small spreads. Then, we define the false positive ratio (FPR) as the fraction of all of the flows with a spread smaller than l that are mistakenly identified. The false negative ratio (FNR) is defined as the fraction of all of the flows with a spread greater than h that are mis-identified. The second set of experiments compares our solutions and ESD for FPR and FNR. We set $m = 0.2MB$. The values of p of two-stage estimator and three-stage estimator are set to 0.15 and 0.19, respectively. The experimental results are shown in TABLE VIII. The values of FPR and FNR decrease quickly as h increases, and our solutions always work much better than ESD. For example, when $h = 200$, $l = 160$ ($l = 0.8h$), the FPR and FNR of ESD are 4.56×10^{-5} and 0.150, respectively. There are 1689428 flows with spreads less than 160 and 287 flows with spreads larger than 200 in our database. This means 77 (43) flows are mis-identified (are not identified) by ESD, which is too many for the applications like scanner detection. However, only 8 (8) flows are mis-identified (are not identified) by two-stage estimator, and only 5 (7) flows are mis-identified by three-stage estimator in the same setting.

VIII. RELATED WORK

To the best of our knowledge, there is no prior work on non-duplicate sampling. However, a large body of studies has been devoted to per-flow spread measurement and packet sampling for per-flow size measurement in high-speed networks. Next, we will briefly summarize the prior work on per-flow traffic measurement in high-speed networks and packet sampling.

Per-flow size measurement is to count the number of packets in each flow, which can be easily solved by using counters. However, there are numerous flows in the packet stream in high-speed networks. It will incur tremendous memory

overhead if we allocate a separate counter to each flow. Therefore, many space-efficient mechanisms were proposed, such as counting Bloom filter [10], Count-Min [15], Counter Braids [6], [37] and Virtual sketch [7], [9], [36], which let flows share counters or registers to reduce the memory overhead.

Per-flow spread measurement is a more difficult problem than flow size measurement since we need to record all the elements for removing duplicates. To reduce the memory overhead, most existing spread estimators are sketch-based and let elements from different flows share a common bits pool. For example, [21] divides bits pool into bitmaps and assigns sources to bitmaps through hash functions. References [12], [13], [19], [20], [38] use virtual bitmaps to share bits uniformly and store the contact information, where the uniform noise introduced by bit sharing can be measured and removed during spread estimation. VI-HLL [18] proposed a virtual HyperLogLog sketch by sharing a common bits pool and achieved better memory efficiency than the virtual bitmap methods. However, these solutions are efficient for the spread estimation of large flows and off-line queries, but have a low estimation accuracy for small or middle flows and incur a heavy on-chip memory consumption since the whole sketch data structure is placed in on-chip memory. Moreover, their compact data structures make it harder to answer online spread queries for flows, which needs thousands of hash operations. Packet sampling can be used to match the rate at which packets are forwarded on-chip and the rate at which per-flow statistics are updated off-chip, reduce on-chip operation and memory consumption. For example, Sampled Netflow [39] uses packet sampling to keep up with the line speed, [2], [7], [9], [36] use packet sampling to filter out most of the small flows, which can reduce the on-chip operation and memory. However, the existing studies can only sample the packets with a pre-set probability, which can be easily achieved. To design a spread estimator based on sampling needs to sample each distinct element with a pre-set probability, which is a much harder problem since the duplicates need to be filtered out. Therefore, we study the non-duplicate element sampling problem in this paper, which can return an efficient spread estimator.

IX. Conclusion

This paper proposes an efficient spread estimator that can answer online spread queries for any flow. Based on a new concept of non-duplicate element sampling, our estimator can achieve both space-efficiency and accuracy-efficiency. The experimental results based on real Internet traffic traces demonstrate that our new estimator provides great flexibility in quantitatively controlling spread measurement, and can work efficiently in a very small on-chip memory space, such as 0.43 bit per element, while the best existing work will fail.

Our future work is to extend research on other network-wide spread estimation functions, such as persistent spread estimation and per-flow spread estimation over multiple periods.

References

[1] Y.-E. Sun, H. Huang, C. Ma, S. Chen, Y. Du, and Q. Xiao, "Online spread estimation with non-duplicate sampling," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 2440–2448.

[2] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

[3] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. 16th Int. Conf. Extending Database Technol. (EDBT)*, 2013, pp. 683–692.

[4] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[5] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," in *Proc. IEEE 28th Conf. Comput. Commun. (INFOCOM)*, Apr. 2009, pp. 504–512.

[6] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: A novel counter architecture for per-flow measurement," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 1, pp. 121–132, 2008.

[7] Y. Zhou, Y. Zhou, M. Chen, Q. Xiao, and S. Chen, "Highly compact virtual counters for per-flow traffic measurement through register sharing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[8] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Per-flow counting for big network data stream over sliding windows," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 1–10.

[9] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Highly compact virtual active counters for per-flow traffic measurement," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1–9.

[10] A. Kumar, J. Xu, and J. Wang, "Space-code Bloom filter for efficient per-flow traffic measurement," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2327–2339, Dec. 2006.

[11] F. Hao, M. Kodialam, and T. Lakshman, "ACCEL-RATE: A faster mechanism for memory efficient per-flow traffic estimation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 155–166, 2004.

[12] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Trans. Netw.*, vol. 19, no. 5, pp. 1253–1264, Oct. 2011.

[13] H. Huang *et al.*, "You can drop but you can't hide: K-persistent spread estimation in high-speed networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1889–1897.

[14] H. Huang *et al.*, "An efficient K-persistent spread estimator for traffic measurement in high-speed networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1463–1476, Aug. 2020.

[15] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[16] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proc. ACM SIGCOMM*, 2016, pp. 101–114.

[17] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 1–34, Dec. 2019.

[18] Y. Zhou, Y. Zhou, M. Chen, and S. Chen, "Persistent spread measurement for big network data based on register intersection," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, p. 15, 2017.

[19] Q. Xiao, Y. Qiao, M. Zhen, and S. Chen, "Estimating the persistent spreads in high-speed networks," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 131–142.

[20] T. Li, S. Chen, W. Luo, and M. Zhang, "Scan detection in high-speed networks based on optimal dynamic bit sharing," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 3200–3208.

[21] Q. Zhao, J. Xu, and A. Kumar, "Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 10, pp. 1840–1852, Oct. 2006.

[22] CAIDA. *The CAIDA UCSD Anonymized Internet Traces 2016*. Accessed: Jul. 28, 2019. [Online]. Available: http://www.caida.org/data/passive/passive_2016_dataset.xml

[23] N. Duffield, C. Lund, and M. Thorup, "Flow sampling under hard resource constraints," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 85–96, Jun. 2004.

[24] N. Duffield, C. Lund, and M. Thorup, "Learn more, sample less: Control of volume and variance in network measurement," *IEEE Trans. Inf. Theory*, vol. 51, no. 5, pp. 1756–1775, May 2005.

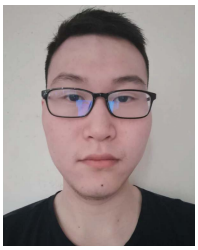
[25] S. L. Feibish, Y. Afek, A. Bremler-Barr, E. Cohen, and M. Shagam, "Mitigating DNS random subdomain DDoS attacks by distinct heavy hitters sketches," in *Proc. 5th ACM/IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, 2017, p. 8.

[26] V. Braverman, E. Grigorescu, H. Lang, D. P. Woodruff, and S. Zhou, "Nearly optimal distinct elements and heavy hitters on sliding windows," in *Proc. APPROX-RANDOM*, 2018, pp. 1–22.

[27] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: An efficient algorithm for finding heavy hitters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 1, pp. 7–16, 2008.
[28] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, 2004, pp. 101–114.
[29] T. Yang *et al.*, "HeavyKeeper: An accurate algorithm for finding top-K elephant flows," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 1845–1858, Oct. 2019.
[30] Q. Huang *et al.*, "Sketchvisor: Robust network measurement for software packet processing," in *Proc. Conf. ACM Special Interest Group Data Commun.*, New York, NY, USA, 2017, pp. 113–126.
[31] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Proc. NDSS*, 2005.
[32] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proc. ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2003, pp. 153–166.
[33] Q. Zhao, A. Kumar, and J. Xu, "Joint data streaming and sampling techniques for detection of super sources and destinations," in *Proc. 5th ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2005, p. 7.
[34] P. Flajolet and G. Nigel Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, Oct. 1985.
[35] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*. France: DMTCS, 2007, pp. 137–156.
[36] Z. Mo, Y. Qiao, S. Chen, and T. Li, "Highly compact virtual maximum likelihood sketches for counting big network data," in *Proc. 52nd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2014, pp. 1188–1195.
[37] Y. Lu and B. Prabhakar, "Robust counting via counter braids: An error-resilient network measurement architecture," in *Proc. IEEE 28th Conf. Comput. Commun. (INFOCOM)*, Apr. 2009, pp. 522–530.
[38] A. Marold, P. Lieven, and B. Scheuermann, "Distributed probabilistic network traffic measurements," in *Proc. 17th GI/ITG Conf. Commun. Distrib. Syst. (KiVS)*, vol. 17, 2011, pp. 133–144.
[39] *Cisco Sampled Netflow*. Accessed: Jan. 15, 2020. [Online]. Available: http://www.cisco.com

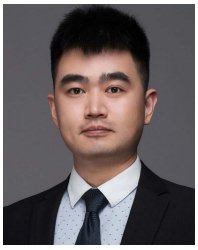
He Huang (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), China, in 2011. From 2019 to 2020, he was a Visiting Research Scholar with Florida University, Gainesville, USA. He is currently a Professor with the School of Computer Science and Technology, Soochow University, China. He has authored more than 100 articles in related international conference proceedings and journals. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of the Association for Computing Machinery (ACM). He has served as the Technical Program Committee Member for several conferences, including IEEE INFOCOM, IEEE MASS, IEEE ICC, and IEEE Globecom. He received the Best Paper Awards from Bigcom 2016, IEEE MSN 2018, and Bigcom 2018.

Yu-E Sun (Member, IEEE) received the Ph.D. degree from the Shenyang Institute of Computing Technology, Chinese Academy of Sciences. From 2019 to 2020, she was a Post-Doctoral Research Scholar with Florida University, Gainesville, USA. She is currently a Professor with the School of Rail Transportation, Soochow University, China. She has authored more than 80 articles in related international conference proceedings and journals. Her current research interests span traffic measurement, privacy preserving, algorithm design, and analysis for wireless networks. She is a member of ACM. She received the Best Paper Awards from Bigcom 2016, IEEE MSN 2018, and Bigcom 2018. She has served as the Technical Program Committee Co-Chair for ACM MSCC-MobiHoc 2016.



Chaoyi Ma (Graduate Student Member, IEEE) received the B.S. degree in information security from the University of Science and Technology of China, Hefei, China, in 2018. He is currently pursuing the Ph.D. degree in computer and information science and engineering with the University of Florida, Gainesville, FL, USA. His advisor is Prof. Shigang Chen. His research interests include network traffic measurement, big network data, and network security and privacy.

Shigang Chen (Fellow, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China in 1993 and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign in 1996 and 1999, respectively. After graduating from UIUC, he was with Cisco Systems on network security for three years and helped starting a network security company, Protego Networks. He joined the University of Florida as an Assistant Professor in 2002, where he was promoted to an Associate Professor in 2008 and a Professor in 2013. He is currently a Professor with the Department of Computer and Information Science and Engineering, University of Florida. He has published more than 190 peer-reviewed journal articles/conference papers and had 12 U.S. patents. He is an ACM Distinguished Member and an IEEE ComSoc Distinguished Lecturer. He received the IEEE Communications Society Best Tutorial Paper Award in 1999, the NSF CAREER Award in 2007, and the Cisco University Research Award in 2007 and 2012. He holds the University of Florida Research Foundation Professorship from 2017 to 2020 and the University of Florida Term Professorship from 2017 to 2020.



Yang Du (Member, IEEE) received the B.E. degree from Soochow University, China, in 2015, and the Ph.D. degree from the University of Science and Technology of China in 2020. He is currently a Post-Doctoral Fellow with the School of Computer Science and Technology, Soochow University. His research interests include network traffic measurement, mobile crowdsensing, and truth discovery.

Haibo Wang (Graduate Student Member, IEEE) received the B.E. degree in nuclear science and the master's degree in computer science from the University of Science and Technology of China in 2016 and 2019, respectively. He is currently pursuing the Ph.D. degree in computer and information science and engineering with the University of Florida. His main research interests include the Internet traffic measurement, software defined networks, and optical circuit scheduling.

Qingjun Xiao (Member, IEEE) received the B.Sc. degree from the Computer Science Department, Nanjing University of Posts and Telecommunications, in 2003, the M.Sc. degree from the Computer Science Department, Shanghai Jiaotong University, China, in 2007, and the Ph.D. degree from the Computer Science Department, The Hong Kong Polytechnic University, in 2011. He is currently an Associate Professor with Southeast University, China. His research interests include network traffic measurement, streaming data processing, and machine learning applications in network security.