Online Spread Estimation with Non-duplicate Sampling

Yu-E Sun[†‡], He Huang[*‡], Chaoyi Ma[‡], Shigang Chen[‡], Yang Du[§], Qingjun Xiao[¶]

[†]School of Rail Transportation, Soochow University, P. R. China
[*]School of Computer Science and Technology, Soochow University, P. R. China
[‡]Department of Computer and Information of Science and Engineering, University of Florida, USA
[§]School of Computer Science and Technology, University of Science and Technology of China, P. R. China
[¶]School of Cyber Science and Engineering, Southeast University, P. R. China
E-mails: {sunye12,huangh}@suda.edu.cn, ch.ma@ufl.edu, sgchen@cise.ufl.edu, jannr@mail.ustc.edu.cn, csqjxiao@seu.edu.cn
*He Huang is the corresponding author.

***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 measurement to be performed at the line rate, but it has to accept tradeoff with 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 which is common in practice. The new estimator supports online queries in real time and produces spread estimation with much better accuracy. By storing traffic data in off-chip memory, our new design faces a key technical challenge of efficient non-duplicate sampling. We propose a two-stage solution with on-chip/off-chip data structures and algorithms, which are not only efficient but also highly configurable for a variety of probabilistic performance guarantees. The experiment results based on real Internet traffic traces show that our estimator reduces the mean relative and absolute error by around one order of magnitude, and achieves both space-efficiency and accuracy-efficiency in flow classification for small flows compared to the prior art.**

***Index Terms*—Traffic measurement, sampling, spread estimation.**

I. Introduction

Per-flow spread measurement on a packet stream in high-speed networks is a fundamental problem with many practical applications [1]–[4]. Different from counting the number of packets (*i.e.*, flow-size measurement) [5]–[8], 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 [9], [10].

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 [11]–[16]. 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 [11], [12], [17]–[19]. 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. However, small-spread flows are sometimes important to certain applications, for example, detection of stealthy denial-of-quality attacks, where the attackers stay low-key to avoid detection that focuses on large flows [19].

This paper adopts an on-chip/off-chip design which is much less investigated. Not only does our choice complement the prior work, but it makes sense for two reasons: First, on-chip resources are often limited due to speed, power, and price considerations [20]. Second, 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 [21] can easily include

2440

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.

Our choice of using off-chip memory for traffic measurement is actually typical in practice (NetFlow [22], [23]) and has been studied for flow-size and heavy-hitter measurements [24]–[27]. 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 randomly select a subset of packets for recording. 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 cannot be sampled and on-chip operation must be efficient, which prevents us from using conventional methods to search for duplicates.

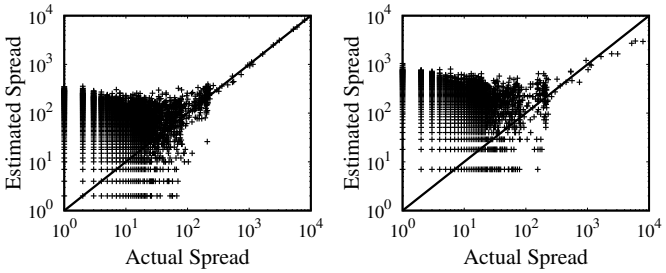
This paper proposes an efficient spread estimator based on an on-chip/off-chip design that supports online spread estimation, which may be performed even on a per-sampled-packet basis (for instance, to trigger alerts when flow spread exceeds a threshold for real-time scanner or DDoS detection). We introduce a new non-duplicate sampling method. Using only a bit array, its online operation is very simple and efficient. Our method does not sample each packet with the same probability, but it ensures that each element is sampled with the same probability when it appears for the first time, while its duplicates will not be sampled. All sampled elements are recorded in off-chip memory, where it takes a counter access to answer the query for a flow's spread. Our method combines simple operation with sophisticated control for probabilistic performance guarantees, such as bounded absolute error in spread estimation, bounded relative error, or probabilistic assurance on threshold-based classification. We perform extensive experiments based on real Internet traces downloaded from CAIDA [21]. The experimental results show that our spread estimator achieves much smaller spread estimation error, and incurs much smaller on-chip footprint than the prior art.

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.



(a) $p = 1$ (b) $p = 0.1$

Fig. 1: Spread estimation by the estimator proposed in [19] with $0.11MB$ on-chip memory.

B. Prior Art and Limitations

Most existing spread estimators [11], [12], [17]–[19] 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.

Take the spread estimator in [19] 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.

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 [28]: 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 [22], [23] and its non-Cisco equivalents enable routers to measure per-flow statistics (such as number of packets and 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 the common technique to compensate such a mismatch. The sampling probability p may be determined based on the ratio of off-chip memory throughput and packet forward 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 number of packets or 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 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, 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.

D. Desired Properties

Before we present our solution for non-duplicate sampling, we give a list of desired properties for such a 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 [4], [29], [30], FM [31], HLL [2], [32], and virtual sketches [13], [33]) 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.

E. Two-stage Solution for Non-duplicate Sampling

We 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 second-stage sampling.

The first stage is *element sampling* with a variable probability p' whose value increases over time. We will discuss how to dynamically adjust the value of p' shortly. 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.

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 sampled 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 preset probability, which we will show how to achieve 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 onto 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,

$$\begin{aligned} p'\frac{m-c}{m} &= p \\ p' &= \frac{mp}{m-c}. \end{aligned} \tag{1}$$

The sampling probability p' at the first stage increases as the number c of recorded elements increases. The maximum value of c should be limited to $m(1-p)$ 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.

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 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

$$(1 - p') + p'\frac{c}{m} = 1 - p, \tag{2}$$

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.

F. Off-chip Recording and Online Spread Estimation

We want to stress that our bit-array filter B serves a different purpose from the bitmaps in [1], [4]. 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 [31], HLL [2], [32], and their virtualized versions [13], [33], 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 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 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 divided by the sampling probability p.

The simple operations of non-duplicate sampling has an immediate benefit of online efficiency, both in spread measurement and real-time queries. Yet, 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.

IV. 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.,

$$p \geq 1 - \epsilon^{1/n}. \tag{3}$$

Let N be the total number of distinct elements in the current epoch. At the end of the epoch, all of them will have set their hashed bits in the filter, and from Section III-E 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 [11], [29], N can be approximated as $-m \ln p$. From (3), we should set m as

$$m = -\frac{N}{\ln p} \geq -\frac{N}{\ln(1-\epsilon^{1/n})}. \tag{4}$$

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:

$$m \geq -\frac{\hat{N}}{\ln(1-\epsilon^{1/n})}. \tag{5}$$

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

$$Pr\{c_f = k\} = C_{n_f}^k p^k (1-p)^{n_f-k}. \tag{6}$$

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

$$p_2 = 1 - \sum_{j=\lceil (n_f-\delta')p \rceil}^{\lfloor (n_f+\delta')p \rfloor} C_{n_f}^j p^j (1-p)^{n_f-j}. \tag{7}$$

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

$$1 - \sum_{j=\lceil (n_f-\delta')p \rceil}^{\lfloor (n_f+\delta')p \rfloor} C_{n_f}^j p^j (1-p)^{n_f-j} \leq \epsilon. \tag{8}$$

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 spread. Thus, we can obtain the optimal value of p by solving (9), which ensures that the absolute error of the flows whose spreads are smaller than n is bounded by δ' with probability $1-\epsilon$:

$$1 - \sum_{j=\lceil (n-\delta')p \rceil}^{\lfloor (n+\delta')p \rfloor} C_{n}^j p^j (1-p)^{n-j} = \epsilon. \tag{9}$$

Similar to the analysis for the absolute error bound, we can obtain the optimal value of p by solving (10), which ensures the relative error of the flows whose spreads are greater than n is bounded by δ with probability $1-\epsilon$:

$$1 - \sum_{j=\lceil (1-\delta)np \rceil}^{\lfloor (1+\delta)np \rfloor} C_{n}^j p^j (1-p)^{n-j} = \epsilon. \tag{10}$$

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

$$P_{A_1,f} = p^{n_f}. \tag{11}$$

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

$$E_{a,A_1}^f = n_f(\frac{1}{p} - 1). \tag{12}$$

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

$$P_{A_2,f} = (1-p)^{n_f}. \tag{13}$$

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

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

$$\max\{E^f_{a,A_1}, E^f_{a,A_2}\} = \max\{n_f(\frac{1}{p} - 1), n_f\}, \tag{14}$$

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

$$\max\{\frac{E^f_{a,A_1}}{n_f}, \frac{E^f_{a,A_2}}{n_f}\} = \max\{\frac{1}{p} - 1, 1\}. \tag{15}$$

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. Due to the fact that the limited SRAM only allow us to record part of the information of each flow, a precise flow classification is not feasible [4], [28], [30]. Thus, we adopt the probabilistic performance objective from [28].

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:

$$\begin{aligned} &\Pr\{\text{identify} f \text{ as an abnormal flow } |n_f \leq l\} \leq \beta, \\ &\Pr\{\text{mis-identify} f \text{ as an abnormal flow } |n_f \geq h\} \geq \alpha, \end{aligned} \tag{16}$$

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 (6), we have:

$$\begin{aligned} Pr\{c_f \leq k\} &= \sum_{j=0}^{k} C^j_{n_f} p^j (1-p)^{n_f - j}, \\ Pr\{c_f \geq k\} &= \sum_{j=k}^{n_f} C^j_{n_f} p^j (1-p)^{n_f - j}. \end{aligned} \tag{17}$$

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:

$$Pr\{c_f \geq \lceil Tp \rceil\} = \sum_{j=\lceil Tp \rceil}^{n_f} C^j_{n_f} p^j (1-p)^{n_f - j}. \tag{18}$$

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

$$\begin{cases} \sum\limits_{j=\lceil Tp \rceil}^{n_f} C^j_{n_f} p^j (1-p)^{n_f - j} \leq \beta, & \forall n_f \leq l; \\ \sum\limits_{j=\lceil Tp \rceil}^{n_f} C^j_{n_f} p^j (1-p)^{n_f - j} \geq \alpha, & \forall n_f \geq h. \end{cases} \tag{19}$$

By solving (19), 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 $h \geq \frac{l}{p}$.

V. Experimental Results

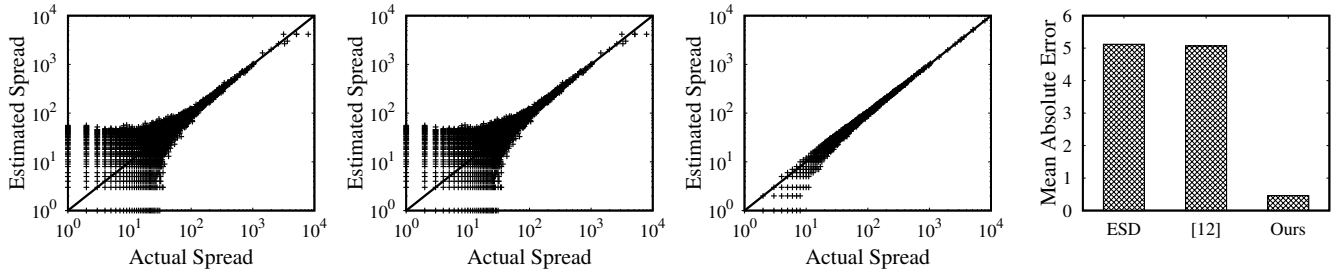
We use five minute of data downloaded from CAIDA [21] as our dataset. This dataset has 1689780 distinct per-source flows, 3150740 distinct elements, and 152163629 packets. Our goal is to estimate the spread of per-source flows in this dataset.

A. Estimation Accuracy

In this part of the experiments, we evaluate the performance of our estimator and compare it with ESD and the estimator proposed in [12] under different sizes of on-chip memory.

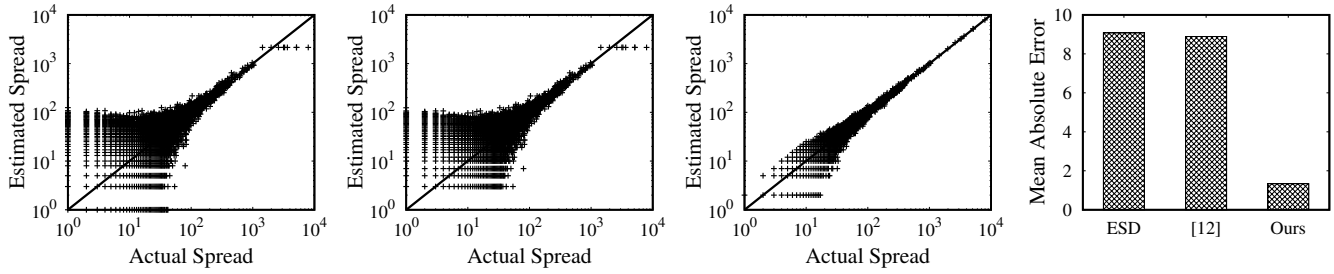
Fig. 2 - Fig. 4 and TABLE I compare our estimator, ESD, and the estimator in [12] on spread estimation accuracy. In this set of experiments, we set the parameter m of all the estimators to the optimal value of our estimator under the given p, and the size of the virtual bitmap of ESD and [12] to the minimum value that can supply a large enough estimation range for all of the flows. The experimental results show that our estimator works much better than the existing ones. 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 estimator only uses 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 estimator 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 the estimator proposed in [12] fail to obtain an accurate estimation for flows with spreads less than 50 when the



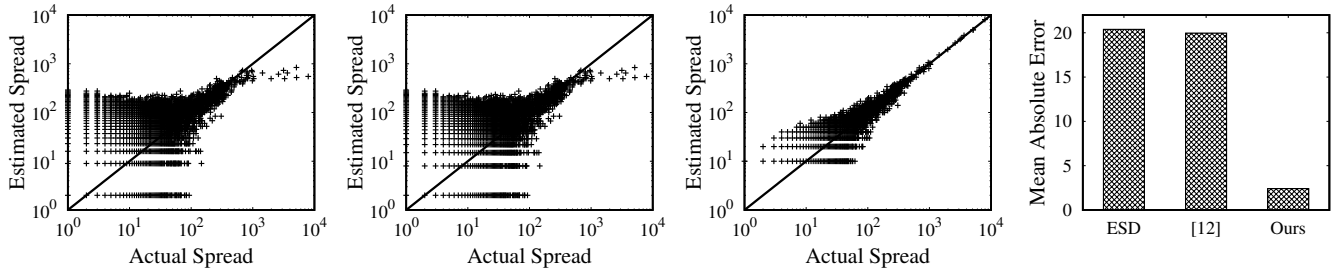
(a) ESD (b) [12] (c) Ours (d) Mean absolute error

Fig. 2: Spread estimation accuracy of ESD, the estimator proposed in [12] and our estimator when $p = 0.8$.



(a) ESD (b) [12] (c) Ours (d) Mean absolute error

Fig. 3: Spread estimation accuracy of ESD, the estimator proposed in [12] and our estimator when $p = 0.4$.



(a) ESD (b) [12] (c) Ours (d) Mean absolute error

Fig. 4: Spread estimation accuracy of ESD, the estimator proposed in [12] and our estimator when $p = 0.1$.

allocated on-chip memory is 4.48 bits per element ($p = 0.8$), for flows with spreads less than 100 when the allocated on-chip memory is 1.09 bit per element ($p = 0.4$), and for almost all of the flows when the allocated on-chip memory is 0.43 bit per element ($p = 0.1$). Our estimator is clearly the winner. It reduces the mean absolute error of all flows by around one order of magnitude compared to the prior art.

TABLE I presents the mean relative error of our estimator, ESD, and the estimator proposed in [12] for flows with different spread intervals. Compared to the existing ones, the experimental results show that our estimator reduces the mean relative error of all of the flows (or flows with spread smaller than 100) by around one order of magnitude. As the spreads of flows increases, the gap between our estimator and the existing ones becomes smaller. However, our estimator always works better than the existing ones on the flows with any spread, especially when the allocated on-chip memory is small.

We then show the performance of our estimator on bounding the absolute and relative errors. Given a set of bounds, we can obtain the optimal value of p based on (8) and (10). TABLE II 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 estimator 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 I, which indicates that our estimator not only has a higher accuracy than the existing ones in the small/medium flow spread estimation, but also in large flow spread estimation.

B. Flow mis-classification probability

Finally, we compare the flow mis-classification probability of our estimator and ESD, which are what ESD was designed for. The first set of experiments compares our estimator and ESD for the amount of memory that they need to satisfy the constraints given in (19). We set $T = (h + l)/2$. TABLE

TABLE I: Mean relative error

spread	all flows			$1 \sim 100$			$101 \sim 1000$			$1001 \sim$		
algorithm / p	ESD	[12]	ours	ESD	[12]	ours	ESD	[12]	ours	ESD	[12]	ours
0.05	19.60	19.20	1.85	19.60	19.20	1.85	0.34	0.34	0.25	0.88	0.88	0.05
0.1	17.12	16.75	1.71	17.13	16.76	1.71	0.27	0.27	0.17	0.82	0.82	0.03
0.3	9.28	9.04	1.25	9.28	9.04	1.25	0.15	0.15	0.09	0.55	0.55	0.02
0.5	6.22	6.09	0.87	6.22	6.09	0.87	0.10	0.10	0.06	0.54	0.54	0.01
0.7	4.23	4.16	0.53	4.23	4.16	0.53	0.07	0.07	0.04	0.54	0.54	0.01
0.9	2.91	2.89	0.18	2.91	2.89	0.18	0.05	0.05	0.02	0.39	0.39	0.01

TABLE II: 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

III shows the memory requirements of our estimator and ESD with respect to α, β, h, and l, which were computed according to the methods proposed in [19] and this work. However, ESD has a limited estimation accuracy for the flows with small spread. 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.

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 estimator requires when h is small, which indicates the space-efficiency of our estimator for classifying flows with small spread. 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 estimator and ESD for FPR and FNR. We set $m = 0.5MB$, and the experimental results are shown in TABLE IV. The values of FPR and FNR decrease quickly as h increases, and our estimator always works much better than ESD. For example, when $h = 100$, $l = 80$ ($l = 0.8h$), the FPR and FNR of ESD are 7.64×10^{-5} and 0.0270, respectively. There are 1689027 flows with spreads less than 80 and 555 flows with spreads larger than 100 in our database. This means 129 (15) flows are mis-identified (are not identified) by ESD, which is too many for the applications like scanner detection. However, only 13 (10) flows are mis-identified (are not identified) by our estimator in the same setting.

VI. 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.

Acknowledgement

The research of authors is partially supported by National Science Foundation (NSF) under Grant No. STC-1562485 and No. CNS-1719222, in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61873177, Grant No. 61672369, and No. Grant 61872080. This work is also supported by the grant from Florida Cybersecurity Center. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of author(s) and do not necessarily reflect the views of the funding agencies (NSF, and NSFC).

References

[1] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 3, pp. 270–313, 2003.

[2] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. of the 16th International Conference on Extending Database Technology (EDBT 2013)*, 2013, pp. 683–692.

[3] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2010)*, 2010, pp. 1–9.

[4] M. Yoon, T. Li, S. Chen, and J. kwon Peir, "Fit a Spread Estimator in Small Memory," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2009)*, 2009, pp. 504–512.

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

[6] 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. of the IEEE GLOBECOM 2016*, 2016, pp. 1–6.

[7] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Per-flow counting for big network data stream over sliding windows," in *Proc. of the IEEE/ACM IWQoS 2017*, 2017, pp. 1–10.

TABLE III: Memory requirements of our estimator and ESD (MB)

h	$\alpha = 0.9, \beta = 0.1$								$\alpha = 0.95, \beta = 0.05$							
	$l = 0.5h$		$l = 0.6h$		$l = 0.7h$		$l = 0.8h$		$l = 0.5h$		$l = 0.6h$		$l = 0.7h$		$l = 0.8h$	
	ESD	ours	ESD	ours	ESD	ours	ESD	ours	ESD	ours	ESD	ours	ESD	ours	ESD	ours
20	–	0.60	–	1.00	–	1.94	–	6.58	–	0.93	–	1.31	–	3.00	–	6.58
50	1.11	0.31	4.59	0.41	–	0.78	–	1.49	4.14	0.42	–	0.58	–	1.12	–	2.22
100	0.45	0.21	0.84	0.27	3.76	0.40	–	0.76	0.82	0.27	2.36	0.36	–	0.55	–	1.10
200	0.23	0.15	0.38	0.19	0.78	0.26	10.87	0.44	0.37	0.19	0.65	0.24	2.04	0.35	–	0.63
300	0.16	0.14	0.26	0.16	0.48	0.21	1.89	0.34	0.25	0.16	0.42	0.20	0.90	0.28	–	0.47
500	0.10	0.11	0.16	0.14	0.29	0.17	0.77	0.25	0.15	0.13	0.25	0.16	0.48	0.21	2.80	0.34
1000	0.05	0.09	0.08	0.11	0.15	0.13	0.38	0.18	0.08	0.11	0.13	0.13	0.24	0.16	0.99	0.23
2000	0.03	0.08	0.04	0.09	0.08	0.11	0.19	0.14	0.04	0.09	0.07	0.10	0.12	0.12	0.49	0.17

TABLE IV: FPR and FNR of our estimator and ESD when $m = 0.5MB$

h	FPR						FNR					
	$l = 0.5h$		$l = 0.7h$		$l = 0.8h$		$l = 0.5h$		$l = 0.7h$		$l = 0.8h$	
	ESD	ours	ESD	ours	ESD	ours	ESD	ours	ESD	ours	ESD	ours
20	2.57e-01	2.97e-04	2.21e-01	4.80e-04	1.89e-01	7.98e-04	2.00e-01	4.21e-02	2.34e-01	8.34e-02	2.67e-01	8.34e-02
50	3.53e-02	4.15e-06	2.11e-02	3.14e-05	1.60e-02	5.57e-05	6.43e-02	7.14e-03	9.21e-02	2.29e-02	1.09e-01	3.93e-02
100	3.57e-04	0.00e+00	1.14e-04	2.37e-06	7.64e-05	7.70e-06	9.01e-03	1.80e-03	1.62e-02	7.21e-03	2.70e-02	1.80e-02
200	5.92e-07	0.00e+00	2.37e-06	0.00e+00	8.88e-06	0.00e+00	6.97e-03	0.00e+00	1.05e-02	0.00e+00	1.74e-02	3.48e-03
300	5.92e-07	0.00e+00	1.78e-06	0.00e+00	2.37e-06	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	1.82e-02	0.00e+00
500	0.00e+00	0.00e+00	0.00e+00	0.00e+00	2.37e-06	0.00e+00	0.00e+00	0.00e+00	2.38e-02	0.00e+00	4.76e-02	0.00e+00

[8] ——, "Highly Compact Virtual Active Counters for Per-flow Traffic Measurement," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2018)*, 2018.

[9] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.

[10] F. Hao, M. Kodialam, and T. Lakshman, "ACCEL-RATE: a faster mechanism for memory efficient per-flow traffic estimation," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, 2004, pp. 155–166.

[11] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 5, pp. 1253–1264, 2011.

[12] H. Huang, Y. Sun, S. Chen, S. Tang, K. Han, J. Yuan, and W. Yang, "You can drop but you can't hide: k-persistent spread estimation in high-speed networks," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2018)*, 2018, pp. 1889–1897.

[13] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Highly compact virtual active counters for per-flow traffic measurement," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2018)*, 2018, pp. 1–9.

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

[15] 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. of the ACM SIGCOMM 2016*, 2016, pp. 101–114.

[16] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 3, no. 3, p. 51, 2019.

[17] Y. Zhou, Y. Zhou, M. Chen, and S. Chen, "Persistent spread measurement for big network data based on register intersection," in *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, 2017, p. 15.

[18] Q. Xiao, Y. Qiao, M. Zhen, and S. Chen, "Estimating the persistent spreads in high-speed networks," in *Proc. of the IEEE 22nd International Conference on Network Protocols (ICNP 2014)*, 2014, pp. 131–142.

[19] T. Li, S. Chen, W. Luo, and M. Zhang, "Scan detection in high-speed networks based on optimal dynamic bit sharing," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2011)*, 2011, pp. 3200–3208.

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

[21] CAIDA, "The CAIDA UCSD Anonymized Internet Traces 2016," http://www.caida.org/data/passive/passive_2016_dataset.xml, accessed July 28, 2019.

[22] N. Duffield, C. Lund, M. Thorup, and M. Thorup, "Flow sampling under hard resource constraints," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, 2004, pp. 85–96.

[23] N. Duffield, C. Lund, and M. Thorup, "Learn more, sample less: control of volume and variance in network measurement," *IEEE Transactions on Information Theory*, vol. 51, no. 5, pp. 1756–1775, 2005.

[24] 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. of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2017, p. 8.

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

[26] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic Lossy Counting: An Efficient Algorithm for Finding Heavy Hitters," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, pp. 7–16, 2008.

[27] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Application," *Proc. of ACM SIGCOMM IMC*, pp. 101–114, October 2004.

[28] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Proc. of the NDSS 2005*, 2005.

[29] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proc. of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003, pp. 153–166.

[30] Q. Zhao, A. Kumar, and J. Xu, "Joint data streaming and sampling techniques for detection of super sources and destinations," in *Proc. of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 7–7.

[31] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of computer and system sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[32] 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*, 2007, pp. 137–156.

[33] Z. Mo, Y. Qiao, S. Chen, and T. Li, "Highly compact virtual maximum likelihood sketches for counting big network data," in *Proc. of the 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2014, pp. 1188–1195.