An Efficient K-Persistent Spread Estimator for Traffic Measurement in High-Speed Networks

He Huang, *Member, IEEE, ACM*, Yu-E Sun, Chaoyi Ma, *Graduate Student Member, IEEE*, Shigang Chen, *Fellow, IEEE*, You Zhou, *Member, IEEE*, Wenjian Yang, Shaojie Tang, *Member, IEEE*, Hongli Xu, *Member, IEEE*, and Yan Qiao

***Abstract*— Traffic measurement in high-speed networks has many important functions in improving network performance, assisting resource allocation, and detecting anomalies. In this paper, we study a generalized problem called *k*-persistent spread estimation, which measures the volume of persist traffic elements in each flow that appear during at least *k* out of *t* measurement periods, where *k* and *t* are two positive integers that can be arbitrarily set in user queries, with $k \leq t$. Solutions to this problem have interesting applications in network attack detection, popular content identification, user access profiling, etc. There is very limited prior art for this problem, only addressing the special case of $k = t$ under a flawed assumption. Removing this assumption, we propose an efficient and accurate estimator for generalized *k*-persistent traffic measurement, with $k \leq t$. Our method relies on bitwise SUM, instead of bitwise AND in the prior art, to combine the information collected from different periods. This change has fundamental impact on the probabilistic analysis that derives the estimator, particular over space-saving virtual bitmaps. Based on real network traces, we demonstrate experimentally the effectiveness of our new method in estimating the *k*-persistent spreads of all network flows. Our estimator performs much better than the prior art on its case of $k = t$. We also incorporate a sampling module to the estimator for improved flexibility, and give a use study on how to detect and find DDoS attackers using the proposed estimator.**

***Index Terms*— Traffic measurement, persistent traffic, spread estimation.**

Manuscript received July 7, 2018; revised July 17, 2019; accepted March 6, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Bremler-Barr. Date of publication May 25, 2020; date of current version August 18, 2020. This work was supported in part by the National Science Foundation (NSF) under Grant STC-1562485 and Grant CNS-1719222, in part by the National Natural Science Foundation of China (NSFC) under Grant 61873177 and Grant 61672369, and in part by Florida Cybersecurity Center under a grant. The preliminary version of this article appeared in IEEE INFOCOM 2018. *(Corresponding author: Yu-E Sun.)*

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

Yu-E Sun is with the School of Rail Transportation, Soochow University, Suzhou 215131, China (e-mail: sunye12@suda.edu.cn).

Chaoyi Ma, Shigang Chen, and You Zhou 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; jayzhou@ufl.edu).

Wenjian Yang is with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China (e-mail: yangwjlx@outlook.com).

Shaojie Tang is with the Naveen Jindal School of Management, The University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: tangshaojie@gmail.com).

Hongli Xu is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230052, China (e-mail: xuhongli@ustc.edu.cn).

Yan Qiao is with the Google Inc., Mountain View, CA 94043 USA (e-mail: yqiao@google.com).

Digital Object Identifier 10.1109/TNET.2020.2982003

I. INTRODUCTION

TRAFFIC measurement over big streaming data in high-speed networks has many applications in improving network performance, assisting resource allocation, and detecting anomalies. One basic measurement function is called *spread (or cardinality) estimation* [9], [14], [18], [28], [32], which estimates the number of *distinct* elements in each network flow during a measurement period, where *flows* may be TCP flows, P2P flows, HTTP flows, or defined arbitrarily based on one or a combination of fields in packet headers, and *elements* under measurement may also be addresses/ports in packet headers or application-specific values in packet payload. In one example, we may consider all packets from the same source address as a per-source flow, and measure the number of distinct destination addresses (i.e., elements) that each source (i.e., flow label) has contacted. In another example, all packets to each server form a flow, and we may measure the number of distinct files (elements) that are accessed from that server (flow label). Spread estimation has many important applications in scan detection, worm monitoring, proxy caching, and content access profiling [29].

We stress that flow-spread estimation is different from traditional flow-size estimation that measures the number of packets in each flow [4], [22], [33] or identifying elephant flows [7], [20], [31]. Spread estimation is to count the number of *distinct* elements, requiring duplicate elements to be filtered. It is harder than size estimation, which counts simply the number of packets. For example, consider a per-source flow where a source host sends 10, 000 packets to 10 destination hosts. The flow size is simply 10, 000, but if we define the flow spread as the number of distinct destination addresses in the flow, it is just 10, where all packets to the same destination address are considered as duplicates. Such a spread measurement helps in scan detection, where a large spread indicates that the source host may be performing scanning.

This paper investigates a new problem called *k-persistent spread estimation*. Traditionally, traffic measurement is performed over each measurement period (or called epoch) whose length is pre-defined. Instead of performing spread estimation within each period in isolation, we look across a number t of consecutive periods and estimate the number of distinct, *persistent* elements that appear in a flow over at least k out of t periods, where t and k can be arbitrarily defined in user queries.

The most related work is done by Xiao *et al.* [26], which is a special case of our problem — finding the number of distinct elements that appear in a flow during all t measurement periods, i.e., $k = t$. Their work can be motivated by an application example of detecting stealthy denial-of-quality attacks [26], where malicious hosts send a sufficient number of service requests to a server to slow it down, without overwhelming the server (which would make the attack's presence obvious). Xiao *et al.* makes the following observation from real traffic traces: legitimate users connect to a server intermittently, while attacking hosts would keep sending packets to the server. Therefore, one can separate attackers from legitimate users by finding those that appear during all measurement periods. However, the attackers can easily evade such detection by not sending packets in some (or even one) periods. To counter against this evasion, we need a new, generalized approach of k-persistent spread estimation, which can find those hosts that appear in at least k out of t periods. Such an approach provides greater flexibility in identifying malicious hosts that have more intense activities than normal ones but do not necessarily show up in every single period. Measuring k-persistent spread has many other applications, such as identifying popular web files that are persistently accessed by users over at least k out of t periods, or profiling Internet access patterns by finding the number of servers that each user persistently access (during at least k out of t periods).

We want to enable k-persistent spread estimation at a high-speed network where routers forward packets at an extremely high rate, which requires packet processing to be performed by specialized network processors with limited on-chip cache memory (such as SRAM). This setting requires all online network functions such as routing, packet scheduling, quality of service, and traffic measurement to be performed efficiently in terms of processing overhead and space overhead.

Xiao *et al.* uses a space-efficient bitmap to record the elements in a flow, where each element is encoded at a certain bit location. Their approach performs bitwise AND among bitmaps from different periods in order to estimate the number of common elements in all t periods. There are two problems that prevent their approach from solving the more general problem of k-persistent spread estimation. First, bitwise AND is lossy in information. When we perform AND among t bits, the result is zero as long as one of the t bits is zero. It no longer tells how many of the t bits are one, which is critical to finding those elements that appear in at least k out of t bitmaps. Second, they make an assumption that elements appear either in all periods or in just one period. This will simplify the mathematical process of deriving an estimator. However, we will demonstrate that the assumption does not always hold in real traffic traces.

In this paper, we propose to solve the problem of k-persistent spread estimation by bitwise SUM among multiple bitmaps collected over a number of t measurement periods. Bitwise SUM has never been explored in spread estimation. This seemly simple deviation from traditional bitwise AND will change the whole mathematical process of deriving an estimator for the more complex problem of finding the number of persistent elements that appear in at least k out of t periods — a problem that was not studied before. While bitwise SUM keeps more information, it makes the analysis much harder. Yet, our analysis no longer requires the assumption that all elements appear either in all periods or in just one period. Furthermore, we show how our bitwise SUM approach can work on virtual bitmaps: Each flow is assigned a virtual bitmap, while the virtual bitmaps of different flows share their bits, which help fit many flows in a tight memory space. The virtual bitmap of a flow will not only record the elements of the flow, but also record some (noise) elements from other flows, due to bit sharing. We develop a mechanism that can filter out the noise under the new context of bitwise SUM. Moreover, in order to further reduce space overhead and processing overhead as well as to meet high link rates, we incorporate a sampling module so that only sampled elements of a flow will be recorded. Finally, we will present a use case to show how to identify stealthy DDoS attackers using our estimator together with a Bloom filter. Based on the real network traces, we demonstrate that the proposed new estimator can accurately estimate the k-persistent spreads of the flows. It also performs much better than [26] for the special case of measuring elements that appear in all periods.

The rest of the paper is organized as follows. We first formulate the k-persistent traffic measurement problem in high speed networks in Section II. Then, we propose mechanisms for k-persistent spread measurement of one flow in Section III and further design an estimator based on the virtual bitmap in Section IV. To evaluate the performance of the proposed mechanisms, we use the real network traffic traces to perform experiments in Section V. Next, we present a use study to detect and find the stealthy DDoS attackers in Section VI. Section VII describes the related work. At last, we conclude the paper in Section VIII.

II. Preliminaries

A flow is a set of packets that share the same flow label, which can be flexibly defined according to the application need. We measure elements that are carried by the packets of a flow; they can also be flexibly defined according to the application need. For example, we may define that all packets sent to the same destination host constitute a (per-destination) flow, and let elements be the source addresses carried by the packets. We define the spread of a flow as the number of distinct elements in the flow. Monitoring the number of sources in each flow helps us detect DDoS attacks when the spread of a flow spikes abnormally, i.e., it receives packets from an unusually large number of sources. As another example, we may define that all packet sent from the same source host constitute a (per-source) flow, and let elements be the destination addresses carried by the packets. Monitoring the number of destinations in each flow helps us detect scanners (or worm attackers) that have unusually high spreads, i.e., sending packets to a large number of different destinations.

Below we define the concepts of k-persistent elements and k-persistent spreads, which are the subjects of research in this paper.

Definition 1 (k-Persistent Element): Given t measurement periods of interest, we define an element of flow f as a

k-persistent element *if this element appears in the flow during at least* k *out of* t *periods, where* $1 \le k \le t$.

Definition 2 (k-Persistent Spread): Given t *measurement periods of interest, we define the k-persistent spread of flow* f *as the number of distinct k-persistent elements in the flow.*

This problem is to design data structures and algorithms to estimate the k-persistent spreads of flows that pass through a high-speed router, which performs per-packet operations in on-chip memory of a network processor.

III. Measuring k-Persistent Spread of One Flow With Element Sampling

We first consider a single flow. In each measurement period, a router records the flow in a bitmap, which is offloaded to a server at the end of the period. After a number of periods, we can make query to the server which will compute k-persistent spread of the flow based on the bitmaps it stores.

A. Online Recording of a Flow

The router creates a bitmap B of m bits to store a traffic record of flow f. Denote the ith bit of B as $B[i]$. At the beginning of each measurement period, it resets the bitmap to zero. From each arrival packet of the flow, the router extracts $\langle e, f\rangle$, where e is the element to be recorded and f is the flow label. For example, f may be the destination address and e may be the source address as in the DDoS detection application we discussed earlier.

Each element has a probability p $(0 < p \le 1)$ to be recorded by the router. Sampling becomes important when we discuss how to measure all flows through a router together, where it allows the router to set p smaller than 1 if it needs to reduce overhead in order to catch up with the line rate. With $\langle e, f\rangle$ extracted from a packet, the router computes a hash value $H(e \oplus f)$, where $H(\ldots)$ is a hash function whose default range is X. Only if $H(e \oplus f)/X \le p$, the element e will be recorded as follows: The router performs another hash $H(e) \in [0, m)$ and sets $B[H(e)]$ to one. Note that the hash range can be reduced to m by modulo m.

B. Joining Bitmaps by Bitwise SUM

After a number t of measurement periods, the server stores a set of traffic records, $\mathcal{B} = \{B_1, \ldots, B_t\}$, about flow f from the router. Given a user query with f and k, we want to compute the k-persistent spread of f based on the information in $\mathcal{B}$. Let n_j be the number of elements that appear at the router in j out of t measurement periods, for $1 \le j \le t$. The k-persistent spread n^* can be calculated as $n^* = \sum_{j=k}^{t} n_j$.

Bitwise AND has been used in the prior art [26] to join the information of multiple bitmaps. However, the operation is lossy: When AND is performed over t bits, the result is zero as long as at least one of the t bits is zero; the result does not reflect exactly how many of the t bits are one, which is important to the estimation of k-persistent spread. Hence, we join the information of multiple bitmaps by bitwise SUM, a new technique never explored in spread estimation. Let S be the resulting counter array of bitwise SUM, $B_j[i]$ the ith bit of B_j, and $S[i]$ the ith counter of S, for $1 \le j \le t$ and $1 \le i \le m$. We define $S[i] = \sum_{j=1}^{t} B_j[i]$. If a k-persistent element is recorded by the ith bit, it will set the ith bit of at least k bitmaps in $\mathcal{B}$. Hence, we must have $S[i] \ge k$.

C. Estimating k-Persistent Spread

We cannot estimate the number of k-persistent elements by counting the number of counters in S whose values are at least k. The reason is hash collision. On the one hand, when multiple elements with spreads less than k are hashed to the same ith bit, they *together* may set the ith bit of at least k bitmaps in $\mathcal{B}$, causing $S[i] \ge k$ and thus over-estimation of k-persistent spread. On the other hand, when multiple elements with spreads at least k are hashed to the same bit, they produce a single counter greater than or equal to k, resulting in under-estimation.

Let V_j, $0 \le j \le t$, be the fraction of counters in S whose values are j. Its value can be found by counting the number of j's in S and dividing that number by m. Let N be the total number of distinct elements that appear at the router during all t periods, i.e., $N = \sum_{j=1}^{t} n_j$. What we want to know are N, n_1, …, and n_t, from which the k-persistent spread n^* can be computed. What we already know are V_0, V_1, …, and V_t. If we can establish one equation for each V_j, $0 \le j \le t$, that relates the unknowns (N, n_1, …,n_t) to V_j, then we will have $t+1$ equations to solve for the values of the unknowns. Note that because $N = \sum_{j=1}^{t} n_j$, we actually have t unknowns, and that because $\sum_{j=0}^{t} V_j = 1$, we actually have t independent equations.

More specifically, as our probabilistic analysis will show, we can establish an equation that relates V_0 to N, from which N can be computed. We can establish an equation that relates V_1 to N and n_1, from which n_1 can be computed. In general, we can establish an equation that relates V_j to N, n_1, …, and n_j, from which n_j can be computed, where N, n_1, …, n_{j-1} have been computed by the previous equations, for $1 \le j \le t$. Next we derive the functional relationship between V_j and the unknowns (N and n_j), for $1 \le j \le t$.

Consider an arbitrary counter $S[i]$. The probability for $S[i]$ to be j is denoted as $\Pr\{S[i] = j\}$. There are exactly j bitmaps in $\mathcal{B}$ whose ith bits are one. Consider an arbitrary subset c_j of j bitmaps from $\mathcal{B}$. There are C_t^j ways to form such a subset. Denote the complement of the subset as c_{-j}, which consists of all bitmaps from $\mathcal{B}$ that are not in c_j. Performing bitwise SUM over c_j and c_{-j}, we denote $S_{c_j}[i] = \sum_{B_* \in c_j} B_*[i]$ and $S_{c_{-j}}[i] = \sum_{B_* \in c_{-j}} B_*[i]$. Let $\Pr\{S_{c_j}[i] = j, S_{c_{-j}}[i] = 0\}$ be the probability for $S_{c_j}[i] = j$ and $S_{c_{-j}}[i] = 0$ to happen. It is the probability that all bitmaps in c_j have their ith bit set to one while all bitmaps in c_{-j} have their ith bits stay as zero. We have

$$\Pr\{S[i] = j\} = C_t^j \Pr\{S_{c_j}[i] = j, S_{c_{-j}}[i] = 0\}. \qquad (1)$$

To derive $\Pr\{S_{c_j}[i] = j, S_{c_{-j}}[i] = 0\}$, we first derive the probability $\Pr\{S_{c_{-j}}[i] = 0\}$ that none of the elements is hashed to the ith bit of any bitmap in c_{-j}, which happens when the following two independent events happen.

Event H_1: none of the recorded $(j+1)$-persistent elements is mapped to the ith bit of any bitmap in $\mathcal{B}$. Otherwise, it will set the ith bit of at least $(j+1)$ bitmaps, including at least one in c_{-j}, because c_j only has j bitmaps.

Denote the probability of event H_1 as $\Pr\{H_1\}$. The probability of any recorded $(j+1)$-persistent element not being hashed to the ith bit is $1-\frac{1}{m}$. The number of $(j+1)$-persistent elements is $N-\sum_{l=1}^{j} n_l$, and each element has a probability of p to be recorded. Then, we have

$$\Pr\{H_1\} = (1-\frac{1}{m})^{(N-\sum_{l=1}^{j} n_l)p}. \tag{2}$$

Event H_2: Any recorded element whose spread is less than or equal to j is not hashed to the ith bit of any bitmap in c_{-j}. Note that such an element may be hashed to the ith bit, but does not appear in the measurement periods when the bitmaps in c_{-j} are produced.

Consider an arbitrary element e that is recorded by the router in l $(\leq j)$ periods. Element e has a probability of $\frac{1}{m}$ to be hashed to the ith bit, and a probability of $\frac{C_t^l-C_j^l}{C_t^l}$ to appear in at least one bitmap of c_{-j}. Thus, the probability that e sets the ith bit of at least one bitmap in c_{-j} is $\frac{1}{m}\frac{C_t^l-C_j^l}{C_t^l}$. Event H_2 means that e does not set the ith bit of any bitmap in c_{-j}. That probability is $1-\frac{1}{m}\frac{C_t^l-C_j^l}{C_t^l}$. There are $n_l p$ recorded elements appearing at the router in l periods. Hence,

$$\Pr\{H_2\} = \prod_{l=1}^{j}(1-\frac{1}{m}\frac{C_t^l-C_j^l}{C_t^l})^{n_l p}. \tag{3}$$

Combining the above analysis, we have

$$\begin{aligned}\Pr\{S_{c_{-j}}[i]=0\} &= \Pr\{H_1\} * \Pr\{H_2\} \\ &= (1-\frac{1}{m})^{(N-\sum_{l=1}^{j} n_l)p} \\ &\quad \times \prod_{l=1}^{j}(1-\frac{1}{m}\frac{C_t^l-C_j^l}{C_t^l})^{n_l p}. \end{aligned} \tag{4}$$

When $S_{c_{-j}}[i]=0$, $S_{c_j}[i]$ may be zero, 1, 2, …, or j. Hence,

$$\begin{aligned}\Pr\{S_{c_{-j}}[i]=0\} &= \sum_{l=0}^{j}\Pr\{S_{c_j}[i]=l, S_{c_{-j}}[i]=0\}, \\ &\quad \Pr\{S_{c_j}[i]=j, S_{c_{-j}}[i]=0\} \\ &= \Pr\{S_{c_{-j}}[i]=0\} \\ &\quad -\sum_{l=0}^{j-1}\Pr\{S_{c_j}[i]=l, S_{c_{-j}}[i]=0\} \end{aligned} \tag{5}$$

Below we derive $\Pr\{S_{c_j}[i]=l, S_{c_{-j}}[i]=0\}$. When $S_{c_j}[i]=l$ and $S_{c_{-j}}[i]=0$, there must exist a subset of l bitmaps (denoted as c_l) that are selected from c_j, with their ith bits being one, while the ith bits of all other bitmaps in c_j are zero. There are C_j^l ways to form such a subset. Consider an arbitrary subset c_l, and let c_{-l} be the complement of c_l, i.e., $c_{-l}=\mathcal{B}-c_l$. The probability of $S_{c_l}[i]=l$ and $S_{c_{-l}}[i]=0$ is denoted as $\Pr\{S_{c_l}[i]=l, S_{c_{-l}}[i]=0\}$. Hence,

$$\Pr\{S_{c_j}[i]=l, S_{c_{-j}}[i]=0\} = C_j^l \Pr\{S_{c_l}[i]=l, S_{c_{-l}}[i]=0\}. \tag{6}$$

By substituting (6) and (4) to (5), we have

$$\begin{aligned}&\Pr\{S_{c_j}[i]=j, S_{c_{-j}}[i]=0\} \\ &\quad = \Pr\{S_{c_{-j}}[i]=0\} - \sum_{l=0}^{j-1} C_j^l \Pr\{S_{c_l}[i]=l, S_{c_{-l}}[i]=0\} \\ &\quad = (1-\frac{1}{m})^{(N-\sum_{l=1}^{j} n_l)p}\prod_{l=1}^{j}(1-\frac{1}{m}\frac{C_t^l-C_j^l}{C_t^l})^{n_l p} \\ &\qquad -\sum_{l=0}^{j-1} C_j^l \Pr\{S_{c_l}[i]=l, S_{c_{-l}}[i]=0\}. \end{aligned} \tag{7}$$

Substituting (7) to (1), we have

$$\begin{aligned}&\Pr\{S[i]=j\} \\ &\quad = C_t^j[(1-\frac{1}{m})^{(N-\sum_{l=1}^{j-1} n_l)p}\prod_{l=1}^{j-1}(1-\frac{1}{m}\frac{C_t^l-C_j^l}{C_t^l})^{n_l p} \\ &\qquad \times (1-\frac{1}{m})^{-n_j p}(1-\frac{1}{m}\frac{C_t^j-C_j^j}{C_t^j})^{n_j p} \\ &\qquad -\sum_{l=0}^{j-1} C_j^l \Pr\{S_{c_l}[i]=l, S_{c_{-l}}[i]=0\}]. \end{aligned} \tag{8}$$

Next we show that $\Pr\{S[i]=j\}=E(V_j)$, where V_j is the fraction of counters in S that are j. It is easy to see that

$$V_j = \frac{1}{m}\sum_{i=1}^{m} I_{i,j}, \tag{9}$$

where $I_{i,j}$ is an indicator variable, whose value is 1 when $S[i]=j$ and 0 otherwise.

Clearly,

$$E(I_{i,j}) = \Pr\{S[i]=j\}.$$

Hence, $\forall\ 0\leq j\leq t$, $0\leq i<m$,

$$\begin{aligned}E(V_j) &= \frac{1}{m}\sum_{i=0}^{m-1}E(I_{i,j}) = \frac{1}{m}\sum_{i=0}^{m-1}\Pr\{S[i]=j\} \\ &= \Pr\{S[i]=j\}. \end{aligned} \tag{10}$$

Substituting (10) to (8), replacing $E(V_j)$ with the instant value V_j measured from S, and replacing n_l with its estimated value $\hat{n}_l$, $1\leq l\leq j$, we have an equation. Solving that equation for $\hat{n}_j$, we have the following recursive estimator: $\forall\ 1\leq j\leq t$,

$$\hat{n}_j = \frac{a-b-c}{p(\ln(1-\frac{C_t^j-1}{mC_t^j})-\ln(1-\frac{1}{m}))}, \tag{11}$$

where

$$\begin{aligned}a &= \ln(\frac{V_j}{C_t^j}+\sum_{l=0}^{j-1} C_j^l \Pr\{S_{c_l}[i]=l, S_{c_{-l}}[i]=0\}), \\ b &= (N-\sum_{l=1}^{j-1}\hat{n}_l)p\ln(1-\frac{1}{m}), \\ c &= \sum_{l=1}^{j-1}\hat{n}_l p\ln(1-\frac{C_t^l-C_j^l}{mC_t^l}). \end{aligned}$$

TABLE I

THE DISTRIBUTION OF FLOW SPREADS

Flow spread range	0∼10	11∼50	51∼200	201∼600	601∼1000	1001∼2000	>2000
Number of flows	14024	15101	6605	2047	379	392	451

Algorithm 1 Estimator for k-Persistent Spread

```
$\Pr\{S_{c_j}[i]=0, S_{c_{-j}}[i]=0\}=V_0$;
Compute $\hat{N}$ from (13).
for $j=1$ to $k-1$ do
    Compute $\hat{n}_j$ by (11);
    Compute $\Pr\{S_{c_j}[i]=j, S_{c_{-j}}[i]=0\}$ by (14);
end for
Set $\hat{n}^*=\hat{N}-\sum_{j=1}^{k-1}\hat{n}_j$;
Return $\hat{n}^*$.
```

We invoke the above estimator in the order of $j = 1, 2, \ldots, t$. For a specific value of j, the computation of $\hat{n}_j$ requires the values of $\hat{n}_l$, $1 \leq l < j$, which are computed earlier by (11). We need the value of N. Note that $\Pr\{S_{c_j}[i] = 0, S_{c_{-j}}[i] = 0\}$ refers to the probability that no element sets the ith bit in any bitmap. Since the probability for any recorded element not to set the ith bit is $1 - \frac{1}{m}$ and there are Np independent recorded elements in total, we have

$$\Pr\{S_{c_j}[i]=0, S_{c_{-j}}[i]=0\}=E(V_0)=(1-\frac{1}{m})^{Np}, \quad (12)$$

where V_0 is the fraction of counters in S whose values are zero. Replacing $E(V_0)$ with the instance value V_0 that can be measured from S, we compute an estimated value of N as follows

$$\hat{N}=\frac{\ln V_0}{p\ln(1-\frac{1}{m})}. \quad (13)$$

We also need the values of $\Pr\{S_{c_l}[i] = l, S_{c_{-l}}[i] = 0\}$, $0 \leq l < j$. Again by (12), we can estimate the value of $\Pr\{S_{c_j}[i] = 0, S_{c_{-j}}[i] = 0\}$ as the measured value of V_0.

Now, replacing N with its estimated value $\hat{N}$ and n_l, $1 \leq l < j$, in (7) with its estimated value $\hat{n}_l$, we have

$$\begin{aligned}\Pr\{S_{c_j}[i]&=j, S_{c_{-j}}[i]=0\}\\ &\approx (1-\frac{1}{m})^{(\hat{N}-\sum_{l=1}^{j}\hat{n}_l)p}\prod_{l=1}^{j}(1-\frac{1}{m}\frac{C_t^l-C_j^l}{C_t^l})^{\hat{n}_l p}\\ &\quad -\sum_{l=0}^{j-1}C_j^l\Pr\{S_{c_l}[i]=l, S_{c_{-l}}[i]=0\}. \quad (14)\end{aligned}$$

Therefore, we should compute (11) and (14) alternatively as j is increased from 1 to t. After $\hat{n}_j$ is computed by (11), we feed it in (14) to compute $\Pr\{S_{c_j}[i] = j, S_{c_{-j}}[i] = 0\}$, which is in turn used in the next iteration to compute $\hat{n}_{j+1}$ by (11). This iterative process is carried out by Algorithm 1 to estimate the number of k-persistent elements, denoted as $\hat{n}^*$.

D. An Example

Suppose the element sets of flow f are $\{e_1, e_4, e_6, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{14}, e_{17}\}$, $\{e_1, e_2, e_3, e_5, e_8, e_{11}, e_{14}, e_{15}, e_{16}, e_{17}\}$ and $\{e_3, e_5, e_6, e_7, e_8, e_9, e_{13}, e_{14}, e_{15}, e_{18}\}$ in measurement period 1, 2, and 3, respectively. We use a bitmap of size 6 to record the elements of f in each measurement period, set $p = 0.5$, and want to know the 3-persistent traffic volume. Assume e_1 is the first arrival element of measurement period 1, and its hash outputs $H(e_1 \oplus f)$ and $H(e_1)$ are $0.4N$ and 4, respectively. Then, we have e_1 sets $B_1[4]$ to one according to our online recording mechanism. At the end of the third period, we have three traffic records B_1, B_2 and B_3(each bit of one is set by the elements below in Fig. 1). Then, the router performs bitwise SUM among these three bitmaps, and get the counter array S as shown in Fig. 1. We can easily get that $V_0 = \frac{1}{6}$, $V_1 = \frac{1}{6}$, $V_2 = \frac{1}{3}$, and $V_3 = \frac{1}{3}$.

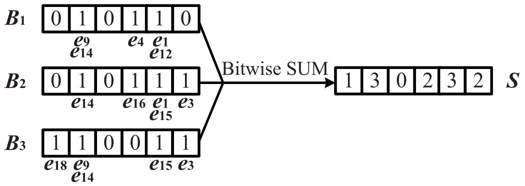


Fig. 1. An example of bitwise SUM.

To measure the 3-persistent traffic volume, we first compute the value of N according to (13), which is equal to 19.6549. By substituting $\hat{N}$, $\Pr\{S_{c_j}[i] = 0, S_{c_{-j}}[i] = 0\} = V_0 = \frac{1}{3}$ and V_1 into (11), we have $\hat{n}_1 = 8.9151$. Then, we can compute $\Pr\{S_{c_j}[i] = 1, S_{c_{-j}}[i] = 0\} = 0.0556$ according to (14) since $\hat{N} = 19.6549$ and $\hat{n}_1 = 8.9151$ have been computed. After that, we can get $\hat{n}_2 = 8.9677$ by substituting $\hat{N}$, $\hat{n}_1$, $\Pr\{S_{c_j}[i] = 1, S_{c_{-j}}[i] = 0\}$ and V_2 to (11), and get the estimated 3-persistent traffic volume $\hat{n}* = 1.7722$ finally. Since the actual volumes of N, n_1, n_2 and n_3 are 18, 8, 8 and 2, respectively, each estimated spread of our estimator is close to its actual value.

IV. MEASURING k-PERSISTENT SPREADS OF MULTIPLE FLOWS WITH ELEMENT SAMPLING

In this section, we consider the case of multiple flows with element sampling. The distribution of flow spreads in real traffic from CAIDA is shown in TABLE I. On the one hand, most flows have spreads less than 100. On the other hand, the largest spread among all flows reaches 10^6 although the number of elephant flows is very small. Since the router does not know the spreads of the flows during online recording, it has to assign the flows with bitmaps of the same size, which should be large enough to accommodate the largest possible spread. We point out that the size of on-chip SRAM is typically small in the introduction. This will limit the number of concurrent flows that the router can handle.

To address the above problem, Yoon *et al.* introduced the concept of virtual bitmaps in [29]. Their idea is to let the bitmaps of different flows to share a common pool of bits, instead of occupying separate memory space. The router allocates a logical virtual bitmap to each flow. More specifically, it selects a certain number of bits pseudo-randomly from a common physical bit array to form a virtual bitmap per flow. The virtual bitmaps of all flows share the bits in the array, where each bit may be selected (shared) by multiple flows. Through such bit sharing, the router is able to handle a large number of flows in a tight space. Overestimation of flow spreads is the main challenge for virtual bitmaps, where bits can be set by other flows due to sharing. There must be a mechanism that can effectively remove such "noise" introduced by other flows.

Yoon's work [29] is not on persistent spread estimation, but its concept of virtual bitmaps has been borrowed by [26] to measure the number of persistent elements. As we point out earlier, this work is a special case of our k-persistent estimation. It finds the number of elements that appear in t out of t measurement periods, not k out of t periods, for $k < t$, that we consider in this paper. We also point out earlier that their bitwise AND operation is lossy and cannot be used to find k-persistent spread. Moreover, they make a flawed assumption that elements either appear in all periods or appear in just one period. This assumption makes their mathematical analysis tractable, but causes large error in real traffic traces that violate this assumption, as we will demonstrate through experiments. In addition, the prior work [26], [29] records all elements of all flows. We incorporate a sampling module to allow one extra dimension of flexibility to reduce space overhead and processing overhead with fewer elements recorded. This flexibility becomes important if the traffic measurement function cannot catch up with the line rate.

Below we develop a new estimator for the general case of k-persistent spread estimation. We do not require the assumption that elements either appear in all periods or appear in just one period.

A. Recording Sampled Traffic With Virtual Bitmaps

Let B be a physical bit array of u bits that will be used to record the elements of all flows. For each flow f, let B_f be a virtual bitmap of m bits that are randomly taken from B through hashing.

$$B_f[i] \equiv B[H(f \oplus H(i))], \quad 0 \leq i < m, \tag{15}$$

where $\oplus$ is bitwise XOR. Two virtual bitmaps of different flows may take (thus share) the same bits from B. We omit the modulo operation in the above formula, assuming the hash output is always modulo'ed to the desired range. For instance, $H(i)$ will produce a hash output of the same length as f.

At the beginning of each measurement period, all bits in B are reset to zero. From each coming packet, the router extracts $\langle e, f\rangle$, where e is the element to be recorded and f is the flow label. Each element is sampled with probability p $(0 < p \leq 1)$ for recording. The router computes $H(e \oplus f) \in [0, N)$. Only if $H(e \oplus f)/N \leq p$, this element will be recorded at the $H(e)$th bit in virtual bitmap B_f. However, we cannot set that bit $B_f[H(e)]$ to one because it is virtual. We set the corresponding bit $B[H(f \oplus H(H(e)))]$ in the physical bit array B.

Packets from different flows are all recorded in B as described above. At the end of each period, the content of B is offloaded to a server.

B. Estimating k-Persistent Spread From Virtual Bitmap

After a number t of measurement periods, the server has t bit arrays, $\mathcal{B} = \{B_1, B_2, \ldots, B_t\}$. When receiving a query about k-persistent spread of a flow f, the server explicitly constructs a virtual bitmap of f from each physical bit array in $\mathcal{B}$ based on (15). These virtual bitmaps are denoted as $\mathcal{B}_f = \{B_{f,1}, B_{f,2}, \ldots, B_{f,t}\}$. This is an offline operation and thus overhead is less of a concern.

From $\mathcal{B}_f$, we can estimate the k-persistent spread of flow f by using the method proposed in Section III. However, not only are the bits in $B_{f,j}$ set by the elements of f, but they may also be set by elements of other flows who share bits with $B_{f,j}$ (from the common bit pool B_j). Hence, the k-persistent spread estimated from $\mathcal{B}_f$ is likely to be larger than the actual value. To solve this problem, we need to find a way to remove the noise in $\mathcal{B}_f$ that is introduced by other flows due to bit sharing.

Let $n_{f,j}$ be the actual number of elements that persistently stay in flow f during any j out of t measurement periods, and $n^m_{f,j}$ be the number of elements from all flows that appear during any j out of t periods, for $1 \leq j \leq t$. We have

$$n^m_{f,j} = n_{f,j} + n^u_{f,j}, \tag{16}$$

where $n^u_{f,j}$ is the noise from other flows. We can compute an estimation $\hat{n}^m_{f,j}$ for $n^m_{f,j}$ by applying the method in Section III to $\mathcal{B}_f$.

To compute the noise, we consider a grand flow F that consists of all flows. We view the physical bit array B in Section IV-A as the bitmap of F for online recording. F includes elements of all flows. Let N^u_j be the number of elements in F that appear at the router during j out of t periods. We join the physical bit arrays of t measurement periods by performing bitwise SUM, which results in a physical counter array M^*, where $M^*[i] = \sum_{j=1}^{t} B_j[i]$. Then, we compute an estimate $\hat{N}^u_j$ for N^u_j, $1 \leq j \leq t$, using the method in Section III based on the grand flow's traffic records in $\mathcal{B}$ and M^*.

From flow f's point of view, beyond its $n_{f,j}$ elements, all the other $(N^u_j - n_{f,j})$ elements in the grant flow are potential noise, for $1 \leq j \leq t$. These noise elements are randomly recorded by the u counters in the physical counter array M^*. The mean noise in each counter is thus $\frac{N^u_j - n_{f,j}}{u}$. Define the virtual counter array of f as the bitwise SUM of the virtual bitmaps of f over the t periods. Similar to how f's virtual bitmap is formed from the physical bit array, its virtual counter array is essentially formed with m counters randomly selected from the physical counter array M^*. The mean noise in the m counters of f's virtual counter array is thus $\frac{m(N^u_j - n_{f,j})}{u}$, i.e.,

$$n^u_{f,j} = \frac{m}{u}(N^u_j - n_{f,j}). \tag{17}$$

Combining (16) and (17), replacing N_j^u with its estimate $\hat{N}_j^u$ and $n_{f,j}^m$ with its estimate $\hat{n}_{f,j}^m$, we have the following estimation $\hat{n}_{f,j}$ for $n_{f,j}$:

$$\hat{n}_{f,j} = \frac{u\hat{n}_{f,j}^m - m\hat{N}_j^u}{u-m}. \tag{18}$$

Then, the estimator for the record k-persistent spread $N_{f,k}$ of flow f is

$$\hat{N}_{f,k} = \sum_{j=k}^{t} \hat{n}_{f,j} = \sum_{j=k}^{t} \frac{u\hat{n}_{f,j}^m - m\hat{N}_j^u}{u-m}. \tag{19}$$

In summary, our estimator based on virtual bitmaps can be computed in three steps. First, we join the physical bit arrays of t measurement periods by performing bitwise SUM, and get the resulted counter array M^*. Let $V_{j,*}$ is the fraction of counters in M^* that are j. We estimate N_j^u, $1 \leq j \leq t$, as follows:

$$\hat{N}_j^u = \frac{a-b-c}{p(\ln(1-\frac{C_t^j-1}{uC_t^j}) - \ln(1-\frac{1}{u}))}, \tag{20}$$

where

$$a = \ln(\frac{V_{j,*}}{C_t^j} + \sum_{l=0}^{j-1} C_j^l \Pr\{S_{c_l}[i] = l, S_{c_{-l}} = 0\}),$$

$$b = (N - \sum_{l=1}^{j-1} N_l^u) p \ln(1 - \frac{1}{u}),$$

$$c = \sum_{l=1}^{j-1} N_l^u p \ln(1 - \frac{C_t^l - C_j^l}{uC_t^l}).$$

Second, we construct t virtual bitmaps for each flow f to get the virtual counter array of f by performing bitwise SUM on these virtual bitmaps. Then, we measure the values of $\hat{n}_{f,j}^m$ for each $1 \leq j \leq t$ by using Equation (11).

Finally, we filter the noise from other flows and obtain the estimated k-persistent spread $\hat{N}_{f,k}$

$$\hat{N}_{f,k} = \sum_{j=k}^{t} \frac{u}{u-m}\hat{n}_{f,j}^m - \frac{m}{u-m}\hat{N}_j^u. \tag{21}$$

C. Probabilistic Error Bound

During each measurement period, the virtual bitmaps of different flows are formed by pseudo-randomly selecting bits from the same physical bit array. Hence, any bit in the physical bit array may be chosen by multiple virtual bitmaps. Such bit sharing introduces noise in the virtual bitmaps of different flows.

Our k-persistent estimator computes the mean noise caused by bit/counter sharing, and subtracts this mean noise from the spread estimation, which is the term $\frac{m}{u-m}\hat{N}_j^u$ in (21). However, the actual noise in the virtual counter array of flow f is not a constant but a random variable from the probabilistic element encoding process. The actual noise is an instance from a distribution, and it is not predictable. The difference between the actual noise and the removed noise mean is called the *noise deviation*. We can measure and remove the mean noise, but we cannot remove the noise deviation, which causes estimation error. Below we show that the noise deviation (i.e., the unremoved noise) is bounded probabilistically.

Theorem 1: For an arbitrary flow and an arbitrary positive integer d*, the probability for the noise deviation introduced by bit/counter to be bounded by* d *is*

$$\sum_{r=a_2-d}^{a_2+d} C_{a_1}^r (\frac{m}{u})^r (1-\frac{m}{u})^{(a_1-r)},$$

where $a_1 = \sum_{j=k}^{t}(N_j^u - n_{f,j})$*, and* $a_2 = \frac{m}{u}\sum_{j=k}^{t}(N_j^u - n_{f,j})$*.*

Proof: Each k-persistent element from flows other than f has a probability of $\frac{m}{u}$ to choose a counter in the virtual counter array of f. There are $a_1 = \sum_{j=k}^{t}(N_j^u - n_{f,j})$ k-persistent elements from other flows. Hence, the mean noise for flow f is $a_2 = \frac{m}{u}\sum_{j=k}^{t}(N_j^u - n_{f,j})$. Consider a noise deviation bound of d elements. The actual noise bound for f is the sum of the mean noise and the noise deviation bound, i.e., $d + a_2$. The probability for $(d + a_2)$ k-persistent elements from other flows to be encoded in the virtual counter array of f is $(\frac{m}{u})^{d+a_2}$, and the probability for other k-persistent elements not being encoded in the virtual counter array of f is $(1-\frac{m}{u})^{a_1-d-a_2}$. There are $C_{a_1}^{d+a_2}$ ways to choose $d + a_2$ elements from the a_1 elements of other flows, and the actual noise within the deviation bound d is distributed in the range $[a_2 - d, a_2 + d]$. Thus, we have the probability for the unremoved noise deviation to be bound by d is $\sum_{r=a_2-d}^{a_2+d} C_{a_1}^r (\frac{m}{u})^r (1-\frac{m}{u})^{(a_1-r)}$. □

Assume any flow spread is negligibly smaller than the sum of all flows' spreads. Namely, $n_{f,j} \ll N_j^u$. Then, the probabilistic bound becomes independent of the flow's spread. In other words, all flows have similar probabilistic bounds no matter whether they are large flows or small flows. This theoretical result is indeed observed by our experimental results in Fig. 4 and Fig. 7. Theorem 1 characterizes the absolute error bound. If we define the relative error bound as the absolute bound divided by the flow's spread, then it decreases as we examine larger flows, which is also observed in the experiments.

V. Experiments

A. Implementation

We implement our scheme in both hardware and software. For the software implementation, we run our scheme on a machine with Intel(R) Core(TM) i7-4720HQ @ 2.6GHz CPU and 8GB memory. For the hardware implementation, we implement the proposed estimator on an XLINX Nexys4 A7-100T development board, with 15850 logic units, 4860Kbits Block RAM, and a clock rate of 100MHz.

We observe that there is no difference between our hardware implementation and our software implementation in terms of memory and measurement accuracy — which is expected. Thus, we discuss the software experimental results of our estimator in Section V-C and Section V-D. The only difference between the two is throughput, which is as shown in Fig. 2. Hardware is much faster than software, and the throughput of software implementation is increasing with the decrease of p.

TABLE II

ACTUAL AND ESTIMATED k-PERSISTENT SPREAD, WITH EACH PERIOD ONE MINUTE, $u = 0.5$MB

Server Address		the proposed estimator								MVPE
		$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=8$
70.7.230.6	Actual Spread	2339	1324	796	492	329	242	175	108	108
	Estimated Spread	2208	1420	792	497	309	221	172	113	93
224.243.45.196	Actual Spread	11153	7248	4021	2310	1281	639	269	102	102
	Estimated Spread	11332	7176	3851	2301	1266	650	270	94	177
224.243.42.120	Actual Spread	32760	21200	13187	8668	6296	4946	4090	3492	3492
	Estimated Spread	33086	21045	12937	8942	6442	5172	4167	3565	3898
224.243.44.1	Actual Spread	45409	27009	14703	7953	4430	2601	1452	677	677
	Estimated Spread	45908	27147	14608	7979	4355	2675	1618	767	1882

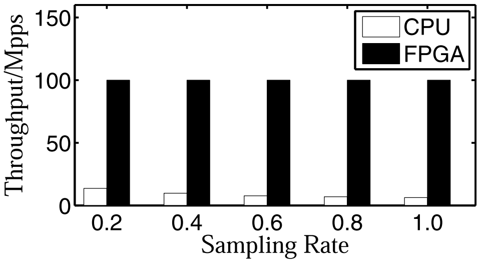


Fig. 2. Throughput of the hardware implementation and software implementation.

B. Experiment Setup

We use one hour of data downloaded from CAIDA as our dataset and estimate the spread of per-destination flows in that dataset. It has 38963 distinct flows, and 7179130 distinct elements. We observe that the flow spreads are distributed extremely unbalance (as shown in TABLE I). The spreads of more than 91% flows are less than 200. However, the largest spread of elephant flow reaches 319807.

In our experiments, we vary the length of each measurement period from 1 to 5 minutes, let $t = 8$, and vary k from 1 to 8. The number of distinct elements in all flows during a period is in the range of [1081531, 1225640]. We vary the memory space allocated to the estimator from 0.25MB to 1MB, and vary the sampling probability from 1 to 0.1. The spread estimation range is bounded by $-m \ln m$. We set $m = 2^{15} = 32768$ bits under a spread upper bound of 340, 696.

C. Estimation Accuracy and Operating Range

We evaluate the performance of the proposed estimator, and compare it with MVPE [26] under $k = 8$ and $p = 1$, which are what MVPE was designed for. TABLE II shows the numerical results of 4 randomly chosen flows, where the measurement period is one minute. We will show the results of all flows in the figures that will follow. The actual spread and the estimated spread of each flow under our estimator are shown with $k = 1 \dots 8$. The last column presents the experimental results of MVPE with $k = 8$. The table shows that the proposed estimator achieves reasonably good accuracy in all cases, with small to medium relative errors, particularly when k is small. For example, when $k = 3$, the relative error is less than 5%. With $k = 8$, our estimator outperforms MVPE. The reason is because MVPE assumes that non-persistent elements appear only in one of the eight period, which is not always true in the traffic trace.

Fig. 3 presents the experimental results of all flows by MVPE when the length of each period is one minute. Each flow is represented by a point in the figure, where the x-coordinate is the actual persistent spread and the y-coordinate is the estimated persistent spread. The closer a point is to the equality line $y = x$ (which is shown in the figure), the better the estimation is. The four plots are the estimations under different memory availability, ranging from 0.25MB to 1MB. Because most flows have small spreads, we see most points are clustered at the left bottom corner. In all four plots, there are points that significantly deviate from the equality line, suggesting large estimation errors, due to the reason we have explained in the previous paragraph. In comparison, Fig. 4 presents the experimental results by the proposed estimator under the same setting. The points are all clustered close to the equality line, suggesting better estimation accuracy than MVPE in Fig. 3. A close examination on the actual data confirms that estimation accuracy is improved under both estimators when the memory u is increased from 0.25MB to 1MB, which is expected since a larger memory reduces sharing between virtual bitmaps and thus reduces the noise they introduce to each other.

Next we study the impact of period length on the performance of the estimators. We increase each period to 5 minutes and repeat the previous experiments, and convert the x and y axes to log scale. The results of MVPE are shown in Fig. 5, and the results of the proposed estimator are shown in Fig. 6. The performance of both estimators degrades under longer periods. The reason is that, as the periods become longer, more elements will be recorded in each period, which will increase the noise in virtual bitmaps through bit sharing. When we compare the proposed estimator with MVPE, the former remains much better than the latter, whose invalid assumption of non-persistent elements appearing only in one period causes significant positive bias in estimation as many points deviate above the equality line in Fig. 5.

We now vary the value of k. Fig. 7 presents the experimental results of the proposed estimator under $k = 1, 3, 5, 7$ in the four plots, respectively. Our estimator performs well under different k values. In fact, the performance is largely insensitive to k for large flows, with modest accuracy decrease for small flows as k increases. Note that small flows are less important than large flows in most applications.

D. Estimation Accuracy Over Sampling Probability p

The reason for introducing a sampling module in our estimator is to provide a mechanism to reduce the processing

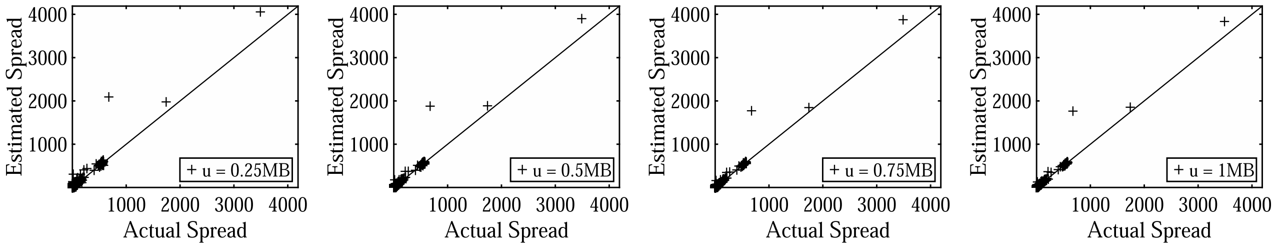


Fig. 3. Persistent spread estimation by MVPE, with each period one minute, $k = 8$.

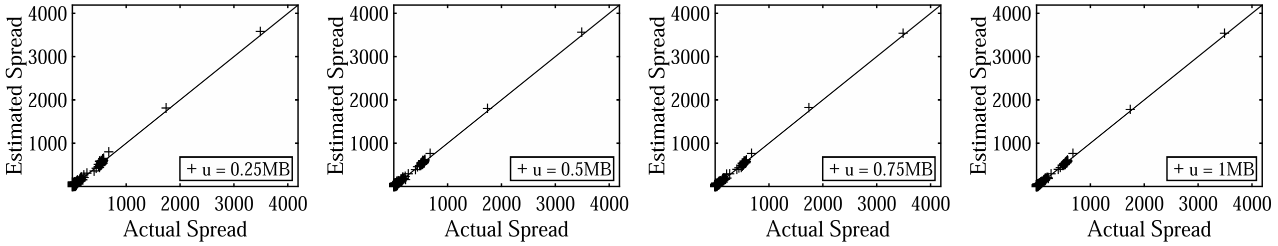


Fig. 4. k-persistent spread estimation by the proposed estimator, with each period one minute, $k = 8$.

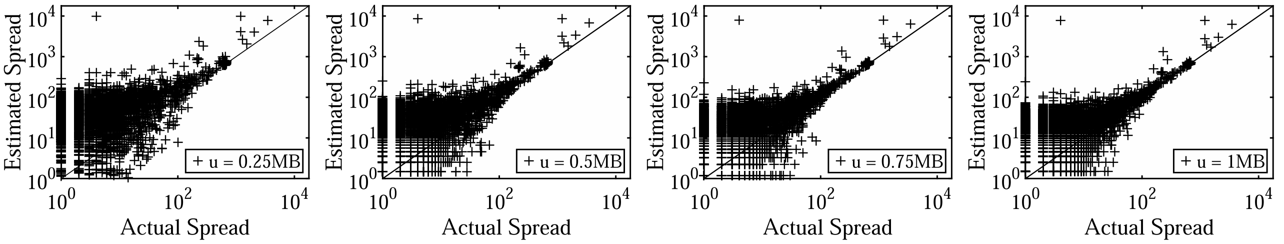


Fig. 5. Persistent spread estimation by MVPE, with each period five minutes, $k = 8$.

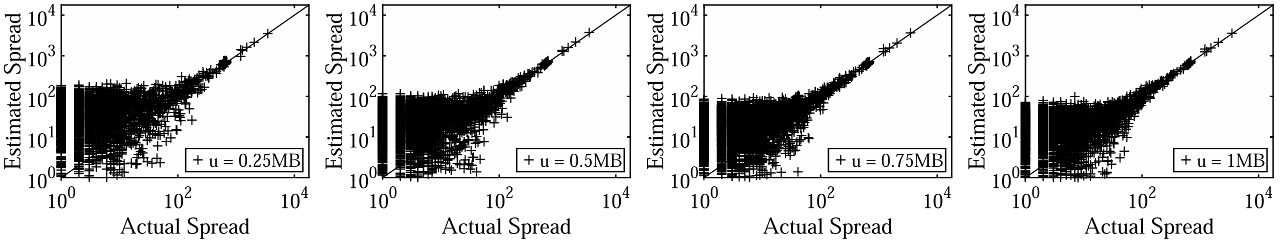


Fig. 6. k-persistent spread estimation by the proposed estimator, with each period five minutes, $k = 8$.

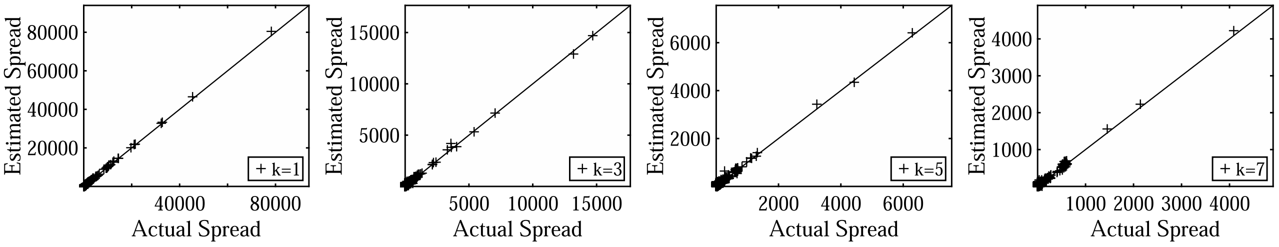


Fig. 7. k-persistent spread estimation by the proposed estimator, with each period one minute, $u = 0.25$MB, under different k values.

overhead of traffic measurement in order to keep up with the line rate. When traffic measurement becomes a bottleneck, we can reduce the sampling probability p. Below we study the impact of sampling on the accuracy of persistent spread estimation.

On the one hand, sampling error will decrease estimation accuracy. On the other hand, sampling reduces the total number of elements recorded in the available memory. From each flow's point of view, when the number of elements from other flows is reduced, the noise introduced by other flows

TABLE III

THE AVERAGE ABSOLUTE ESTIMATION ERROR OF ALL FLOWS

u	0.5MB				0.25MB				0.125MB				0.0625MB			
m p	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$
1.0	25	18	13	10	49	34	24	16	141	89	58	38	724	399	229	130
0.6	29	20	14	11	50	35	24	18	109	74	49	35	384	225	140	91
0.3	37	26	18	14	58	40	28	21	106	71	48	34	255	154	99	70
0.1	57	40	28	22	87	59	41	31	139	93	62	46	255	154	102	74

TABLE IV

THE AVERAGE RELATIVE ESTIMATION ERROR OF FLOWS WITH SPREAD LARGER THAN 10000

u	0.5MB				0.25MB				0.125MB				0.0625MB			
m p	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$	$2^{17}b$	$2^{16}b$	$2^{15}b$	$2^{14}b$
1.0	0.03	0.03	0.04	0.05	0.10	0.04	0.06	0.10	0.10	0.10	0.07	0.13	0.49	0.22	0.27	0.14
0.6	0.06	0.03	0.03	0.05	0.08	0.03	0.05	0.10	0.20	0.05	0.06	0.16	0.12	0.14	0.09	0.36
0.3	0.04	0.04	0.03	0.02	0.05	0.05	0.03	0.04	0.19	0.07	0.04	0.05	0.19	0.10	0.06	0.07
0.1	0.05	0.06	0.07	0.04	0.08	0.11	0.05	0.04	0.16	0.11	0.05	0.06	0.17	0.09	0.06	0.07

into its virtual bitmap will also be reduced, which will in turn improve estimation accuracy. The final accuracy is determined by the combined effect of the above two factors.

We define the absolute estimation error of a flow as the absolute difference between the flow's actual persistent spread and its estimated spread. We measure the average absolute estimation error per flow, which is defined as the sum of all flows' absolute estimation errors divided by the number of flows in an experiment. TABLE III shows the average absolute errors under different settings, with the memory u varying from 0.065MB to 0.5MB, $k = 4$, and sampling probability p from 1 to 0.1. From the table, as p decreases, the absolute errors first increase when u is small (such as 0.125MB), with the reduced noise balancing out much of the impact by higher sampling errors. However, when p becomes very small (such as 0.1), the sampling errors start to dominate, causing significant increase in average absolute errors. With the same sampling probability, as the memory increases, the absolute errors decrease, which is expected because more memory reduces bit sharing and thus noise.

Most applications are interested in super-spreaders, i.e., flows with large spreads. TABLE IV presents the average relative estimation errors among flows whose 1-persistent spreads are greater than 10, 000 under the same parameter settings as in TABLE III. From the table, as u decreases, the relative error increases. Moreover, the optimal value of m is related to p, *i.e.*, a larger p will record more elements in the virtual bitmap and generally need a larger virtual bitmap. However, the relative error will increase with m when u is very small due to there are too many noises in the virtual bitmap. In the case, decreasing the value of p can help decrease the noise and further get a better accuracy.

VI. USE STUDY: DETECT AND FIND THE STEALTHY DDOS ATTACKERS

A. Stealthy DDoS Attack Detection

Consider a classical DDoS attack, where a number of attacking hosts send service requests to a target server at their highest rates in order to overwhelm the server's processing capacity so as to prevent legitimate requests from being served. In such an attack, we can identify the attackers by measuring the request rates. Now consider a stealthy DDoS attack, where the attackers slow down their request rates below a threshold, e.g., one or even zero request in each measurement period. This makes it difficult to separate attackers from legitimate users that show up in a single period because they all make at least one request. However, if we look at a number t of periods, things will become different. To perform a sustained attack, attackers will not stop after making a request in a single period. They need to persistently send requests in the duration of attack that consists of many periods.

For this study, we use per-destination flows, where packets from all source addresses (attackers and legitimate users) to the same destination address (a server) constitute a flow. By measuring the k-persistent spreads of all flows, we can detect a potential stealthy DDoS attack if we observe an usually high persistent spread for a server — there are too many source addresses (potential attackers) that are sending requests to the server persistently.

Unlike the attackers, most normal users tend to use a server intermittently with less persistency [26]. By analyzing normal real-world Internet traffic traces from CAIDA, we observe that the k-persistent spreads of per-destination flows decrease rapidly as k increases, which confirms that most users in normal traffic traces do not exhibit persistent behavior in server access. From Table. II, we can see that most normal users are present in no more than 3 out of 8 periods. As k increases, the number of users accessing a server in at least k periods decreases quickly. Our data analysis shows that $k = 4$ can be an appropriate parameter that filters out most legitimate users when we apply k-persistent spread estimation for stealthy DDoS attacks. (In practice, the value of k should be set based on the normal traffic examination in the network of application.)

With proper parameter setting, we estimate the k-persistent spreads of flows based on the data structures and the estimation procedure described in Section IV. If the k-persistent spread

TABLE V
THE ACTUAL AND ESTIMATED k-PERSISTENT SPREAD WHEN $u = 0.5$MB, $m = 32768$B DURING ATTACK

Server Address		Our estimator based on virtual bitmap							
		$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
70.7.230.6	Actual Spread	4678	3663	3135	2831	1892	1028	175	108
	Estimated Spread	4599	3815	3195	2907	1903	1006	184	106
224.243.45.196	Actual Spread	22306	18401	15174	13463	8762	4364	269	102
	Estimated Spread	22354	18339	14948	13395	8696	4460	425	138
224.243.42.120	Actual Spread	65520	53960	45947	41428	27988	15996	4090	3492
	Estimated Spread	66422	54707	46146	42054	28197	16363	3970	3447
224.243.44.1	Actual Spread	90818	72418	60112	53362	34476	17584	1452	677
	Estimated Spread	91209	71920	60036	53892	34256	18767	2139	389

of a flow is abnormally large, i.e., larger than a threshold that most flows in normal traffic traces will fall under, we classify the flow's destination as a potential victim of stealthy DDoS attack and raise a flag for the network admin to further investigate. One problem is that our data structures in Section IV do not record the individual persistent elements, i.e., the exact source addresses that persistently access the server. To identify these stealthy attackers, we need to augment k-persistent spread measurement with additional information.

B. Finding Stealthy DDoS Attackers

After a potential stealthy DDoS attack is detected through k-persistent spread measurement, the router will be instructed to use a Bloom filter to help find the attackers. A Bloom filter is a space efficient data structure for encoding a set, and has a controlled false positive ratio, i.e., the probability for any non-member to be mis-classified. Each Bloom filter is a bit array that is initialized to zero at the beginning of each measurement period. When an element arrives, it randomly maps to l bits in the array by l independent hash functions, and sets those bits to one. To check whether an element e is in the set or not, we can map e to l bits in the array again (by using the same independent hash functions as it recorded). If all l bits are one, we claim that b belongs to the set; otherwise b does not belong to the set. Next, we will show how to combine Bloom filter in our original design to find the stealthy DDoS attackers.

Suppose the router detects flow f is under attack in period r. From the $(r+1)$ period, it allocates a separate Bloom filter to flow f in each period until f is not under attack. Let $A_{i,f}$ be the Bloom filter of m_f bits that encodes the elements of f in period i. When an element $\langle e, f\rangle$ arrives in period i, the router first hash e to the $H(f \oplus H(H(e)))$th bit of the physical bit array B. Then, it hashes e to l bits of $A_{i,f}$ separately by l independent hash functions $H_1(\ldots), H_2(\ldots), \ldots, H_l(\ldots)$, and sets those bits to one. At the end of period i, $A_{i,f}$ is offload to a server.

After t periods, the server has t Bloom filters, $\mathcal{A} = \{A_{r+1,f}, A_{r+2,f}, \ldots, A_{r+t,f}\}$ for flow f. The server first joins the information of $\mathcal{A}$ by performing bitwise SUM, and the resulted counter is denoted as A_*. Then, it upload A_* to the router. After receiving A_*, the router starts to find the stealthy DDoS attackers. For each arrival element e of flow f, the router hashes e into l counters of A_* by hash functions $H_1(\ldots), H_2(\ldots), \ldots, H_l(\ldots)$. If the value of all these counters in A_* are no less than k, e has a high probability to be an attacker, and the router will drop this packet; Otherwise, e should be a legitimate user and the router will forward this packet to the destination. The intuition, a stealthy attacker, which shows in at least k out of t measurement periods, always maps to same l locations in Bloom filters for f, and increase the value in these locations by one at least k times. Therefore, the corresponding counter of a stealthy attacker in A_* is always equal or larger than k. At the end of each period, the server will update A_* based on the Bloom filters of the last t periods until there is no attack.

Due to the hash conflict, a legitimate user may be mis-classified as an attacker. Next, we discuss the false positive ratio, which is the probability of a legitimate user that may be mis-classified as an attacker in our mechanism. Let $V_{A_*,k}$ be the fraction of counters in A_* whose values are no less than k. An element e of a legitimate user randomly maps to l counters, and each counter has a chance of $V_{A_*,k}$ to be no less than k. The probability for all the l corresponding counters of A_* no less than k (thus causing mis-classification) is $(V_{A_*,k})^l$.

C. Experimental Results

1) Stealthy DDoS Attack Detection: In this part of experiment, we test the performance of our method under some artificially created stealthy DDoS attack. In particular, we add DDoS attack to those flows in TABLE II, and assume existing users are legitimate users. The number of illegal users for each flow is equal to the number of legitimate users. We let each illegal user randomly drop $2 \sim 4$ measurement periods in each estimation period. The experimental results are shown in TABLE V.

The exiting studies for low-rate DDoS attack detection are based on some observed attack characteristics, such as the abnormal change of the size of each element [19], the correlation coefficient of malicious and legitimate traffic [1], and the persistent traffic volume [26]. For example, [19] and [1] assume that the low-rate DDoS is periodic, which means the traffic size of the attacked flow is periodically changing. [26] assumes that the stealthy DDoS attacker will keep connection with the attacked server during all measurement periods. However, attackers may escape the detection by changing their attack characteristic. Based on the observation of the flows in the real-world Internet traffic traces used in our experiments, the k-persistent spreads decrease rapidly with the increase of k. Most of the users only contact with their target server for at most 3 measurement periods. In the following, we will show that our estimator can detect the stealthy attack as long as the stealthy attackers will keep connection with the target sever significantly longer than legitimate users. Since most of

TABLE VI
THE RATIO OF MIS-CLASSIFIED ELEMENTS WHEN $u = 0.5$MB, $m = 32768$B DURING ATTACK

Server Address	Mis-classified Spread	All Detected Spread	measured ϵ
70.7.230.6	0	2791	0
224.243.45.196	0	13214	0
224.243.42.120	29	39063	0.0007
224.243.44.1	171	50349	0.0034

elements stay in a flow for at most 3 measurement periods, we can choose $k = 4$. Obviously, low-rate stealthy attackers can hardly organize an efficient attack or be distinguished from legitimate users if they only stay in a flow during less than 4 measurement periods. Thus, we let each illegal user randomly drop $2 \sim 4$ measurement periods in each estimation period.

Since $k = 4$ is an appropriate value for the data used in our experiment, the 4-persistent elements should be viewed as illegal users. Based on the experimental results, we found that the value of the 4-persistent spread is close to the actual number of illegal users added to each flow, which validates the accuracy and effectiveness of our estimator against stealthy DDoS attack.

However, we cannot assert a server is under attack or not by estimating the t-persistent spread. For example, the true value of 8-persistent spread of server 224.243.44.1 is only 677, which is much smaller than 90818 (the total number of illegal users and legitimate users). Therefore, the MVPE fails to detect the stealthy DDoS attack, let alone estimating the scale of the attack.

2) Finding the Stealthy DDoS Attackers: We demonstrate that we can detect a flow is under DDoS attack or not based on our k-persistent estimator. In this part of experiment, we test the performance of the proposed method based on Bloom filters to find the stealthy DDoS attackers. We again add DDoS attackers as in Section VI-C.1, and set the length of one measurement period is one minute. After 8 measurement periods, we detect that the flows in TABLE V are under DDoS attack. Then, we allocate a Bloom filter with the optimal setting, i.e. $m_f = \frac{N_{f,1} l}{\ln 2}$ for each flow f that is under attack, where $N_{f,1}$ is the average number of 1-persistent elements of flow f in previous measurement periods.

The detection results are as shown in TABLE VI, which shows the false positive ratio of our method. For the k-out-of t persistent spread measurement, we must wait until all t periods end, which is 8 minutes in our experiment. The computation time is less than a minute on a desktop. It should be much smaller when the real deployment uses a more powerful server. Hence, the time for detection is slightly more than 8 minutes. In the analysis of Section VI, we assume that all the k-persistent elements are attackers. The estimated number of mis-classified elements and the false positive ratio (denoted as ϵ) are shown in the second and the fourth columns of TABLE VI, respectively. From this table, we observe that our Bloom filter based can find the k-persistent elements efficiently with a very small false positive ratio.

VII. RELATED WORK

To the best of our knowledge, there is no prior work on k out of t persistent spread measurement. For the network traffic measurement, a large body of studies have been devoted to the passive flow size or flow spread estimation, which take advantage of the built-in components in routers or switches to monitor the passing traffic passively.

Flow size refers to the number of packets for each active flow during a certain measurement period. Chen *et al.* propose a scalable counter architecture which can achieve better memory efficiency with high estimation accuracy [4]. [17], [21], [35], [36], [37] also focus on memory efficiency per-flow traffic estimation by introducing the similar statistical memory sharing. Flow spread estimation (also known as the flow cardinality estimation) mainly aims to estimate the number of distinct elements for each flow during a predefined time period [10], [23], [27], [29]. In [29], Yoon *et al.* design a new spread estimator which can achieve good performance in a very tight memory space through building a virtual bit vector. Paper [30] provides a three-stage framework, called OpenSketch, to pipeline the traffic measurement tasks and use flow-size measurement as example. Paper [24] proposes a two-level filtering scheme to find the sources which are connected to a large number of distinct destinations. Papers [2] and [3] study the problem of estimating the distinct element volume and finding the k-heavy hitters in the sliding window model. Paper [11] presents an algorithm to find the flows with spread larger than the threshold. However, all these studies are designed to measure flow spread or identify super spreaders in one measurement period and a sliding time window. They do not keep track of persistent elements over multiple measurement periods or multiple sliding windows. Nor do they monitor persistent elements within a time window. They do not measure the number of times any element appears over time for persistency measurement.

To solve this problem, the research on flow's persistent spread estimation has been attracted more attention since it can detect the long-term network anomalies, such as stealthy DDoS attack or network scan. In [26], authors present a persistent spread estimation method by using the multi-virtual bitmaps for the tight memory scenario. The proposed method can estimate the number of distinct elements which persist in all the predefined t time periods. Zhou *et al.* [34] propose a highly compact Virtual Intersection HyperLogLog (VI-HLL) architecture for the persistent spread measurement for big network data. Dai *et al.* [5], [6] concentrate on finding the persistent items in data streams or distributed datasets. Literature [15] studies the privacy preserving persistent traffic measurement for the intelligent transportation. We have proposed a new problem called k-persistent spread estimation, which measures the number of distinct elements that persist in a flow for at least k out of t predefined number of time periods. The design is based on the observation that the active time for the stealthy attackers are often longer than most of the legitimate users. Further, we find that the most of the malicious users will occupy most of the scanning time periods instead of all the time periods. These observations make the study of k-persistent spread estimation more challenge and meaningful. Due to the very limited size of on-chip SRAM, it is highly desirable to design a light weight estimator. There have been plenty of cardinality estimators proposed in [8], [10], [12], [13], [25], however, these methods will cause accuracy degradation due to the limited memory space. Thus, we choose the data structure of virtual bitmaps which can achieve very high estimation accuracy.

VIII. Conclusion

In this paper, we propose a new k-persistent spread estimator that is able to measure the volume of elements that stay in a flow for at least k out of t predefined measurement periods. This is the first work that studies the generalized persistent estimation problem without the limiting assumption that an element either appears in all periods or appears in one period. The new estimator combines bitwise AND for joining bitmaps, an iterative algorithm in persistent spread computation, and virtual bitmaps with sampling. They together provide a flexible framework for accurately estimating k-persistent spreads in a tight memory space. Experiments based on real traffic trace demonstrate the performance of the proposed estimator, particularly the estimation accuracy under various parameter settings. The results also show that it outperforms the prior art on the special case of persistent spread estimation that the existing work is designed for. Finally we provide a use case against DDoS attacks to demonstrate how to apply the proposed estimator to solve real-time world problems.

References

[1] A. Ain, M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “Rank correlation for low-rate DDoS attack detection: An empirical evaluation,” *IJ Netw. Secur.*, vol. 18, no. 3, pp. 474–480, 2016.

[2] E. Assaf, R. B. Basat, G. Einziger, and R. Friedman, “Pay for a sliding Bloom filter and get counting, distinct elements, and entropy for free,” in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 2204–2212.

[3] 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.

[4] M. Chen, S. Chen, and Z. Cai, “Counter tree: A scalable counter architecture for per-flow traffic measurement,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1249–1262, Apr. 2017.

[5] H. Dai, M. Li, A. X. Liu, J. Zheng, and G. Chen, “Finding persistent items in distributed datasets,” *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 1–14, Feb. 2020.

[6] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong, “Finding persistent items in data streams,” *Proc. VLDB Endowment*, vol. 10, no. 4, pp. 289–300, Nov. 2016.

[7] 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.

[8] M. Durand and P. Flajolet, “Loglog counting of large cardinalities,” in *Proc. Eur. Symp. Algorithms*, 2003, pp. 605–617.

[9] 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, Aug. 2003.

[10] C. Estan, G. Varghese, and M. Fisk, “Bitmap algorithms for counting active flows on high-speed links,” *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.

[11] 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.*, Oct. 2017, pp. 1–6.

[12] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, “HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm,” in *Proc. Anal. Algorithms (AofA)*, 2007, pp. 137–156.

[13] P. Flajolet and G. N. Martin, “Probabilistic counting algorithms for data base applications,” *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, Oct. 1985.

[14] 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.

[15] H. Huang, Y.-E. Sun, S. Chen, H. Xu, and Y. Zhou, “Persistent traffic measurement through vehicle-to-infrastructure communications,” in *Proc. ICDCS*, Jun. 2017, pp. 394–403.

[16] H. Huang *et al.*, “You can drop but you can’t hide: K-persistent spread estimation in high-speed networks,” in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 1889–1897.

[17] T. Li, S. Chen, and Y. Ling, “Fast and compact per-flow traffic measurement through randomized counter sharing,” in *Proc. INFOCOM*, Apr. 2011, pp. 1799–1807.

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

[19] H. Liu and M. S. Kim, “Real-time detection of stealthy DDoS attacks using time-series decomposition,” in *Proc. IEEE Int. Conf. Commun.*, May 2010, pp. 1–6.

[20] 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.

[21] 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.

[22] S. Ramabhadran and G. Varghese, “Efficient implementation of a statistics counter architecture,” in *Proc. ACM SIGMETRICS*, Jun. 2003, vol. 31, no. 1, pp. 261–271.

[23] M. Roesch *et al.*, “Snort-lightweight intrusion detection for networks,” in *Proc. 13th USENIX Conf. Syst. Admin. (LISA)*, 1999, vol. 99, no. 1, pp. 229–238.

[24] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, “New streaming algorithms for fast detection of superspreaders,” in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2005. [Online]. Available: https://www.ndss-symposium.org/ndss2005/new-streaming-algorithms-fast-detection-superspreaders/

[25] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, “A linear-time probabilistic counting algorithm for database applications,” *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.

[26] Q. Xiao, Y. Qiao, M. Zhen, and S. Chen, “Estimating the persistent spreads in high-speed networks,” in *Proc. IEEE ICNP*, Oct. 2014, pp. 131–142.

[27] Q. Xiao, Y. Zhou, and S. Chen, “Better with fewer bits: Improving the performance of cardinality estimation of large data streams,” in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[28] M. Yoon, T. Li, S. Chen, and J.-K. Peir, “Fit a spread estimator in small memory,” in *Proc. INFOCOM*, Apr. 2009, pp. 504–512.

[29] 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.

[30] M. Yu, L. Jose, and R. Miao, “Software defined traffic measurement with OpenSketch,” in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 29–42.

[31] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, “Online identification of hierarchical heavy hitters: Algorithms, evaluation, and application,” in *Proc. ACM SIGCOMM IMC*, Oct. 2004, pp. 101–114.

[32] 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.

[33] Q. Zhao, J. Xu, and Z. Liu, “Design of a novel statistics counter architecture with optimal space and time efficiency,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 323–334, 2006.

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

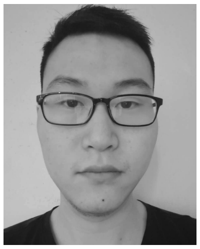
[35] Y. Zhou, Y. Zhou, M. Chen, Q. Xiao, and S. Chen, “Highly compact virtual counters for per-flow traffic measurement through register sharing,” in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[36] 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.

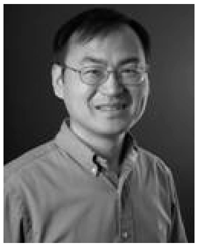
[37] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Highly compact virtual active counters for per-flow traffic measurement," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1–9.

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), in 2011. He is currently a Professor with the School of Computer Science and Technology, Soochow University, China. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of the ACM.

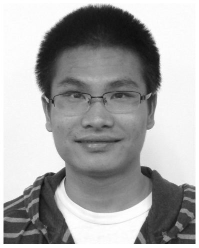
Yu-E Sun received the Ph.D. degree from the Shenyang Institute of Computing Technology, Chinese Academy of Science (CAS), in 2011. She is currently an Associate Professor with the School of Rail Transportation, Soochow University, China. Her current research interests span traffic measurement, algorithm design and analysis for wireless networks, and network security.



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. S. Chen. His research interests include network traffic measurement, big network data, and network security and privacy.



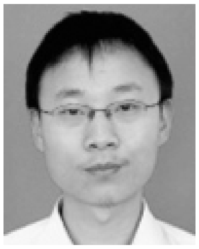
Shigang Chen (Fellow, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduating from UIUC, he 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, and was promoted to an Associate Professor in 2008 and to 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/conference papers and had 12 U.S. patents. He holds the University of Florida Research Foundation Professorship and the University of Florida Term Professorship since 2017. Prof. Chen 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.



You Zhou (Member, IEEE) received the B.S. degree in electronic information engineering from the University of Science and Technology of China, Hefei, China, in 2013. 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. S. Chen. His research interests include network security and privacy, big network data, and the Internet of Things.

Wenjian Yang received the B.S. degree from the Department of Mathematics and Computer Science, Anhui Normal University of China, Wuhu, China, in 2016. He is currently pursuing the M.S. degree with the School of Computer Science and Technology, Soochow University, Suzhou, China. His current research interest is traffic measurement, including transportation traffic measurement and network traffic measurement.

Shaojie Tang (Member, IEEE) received the Ph.D. degree in computer science from the Illinois Institute of Technology in 2012. He is currently an Assistant Professor with the Naveen Jindal School of Management, The University of Texas at Dallas. His research interests include social networks, mobile commerce, game theory, e-business, and optimization. He received the Best Paper Awards in ACM MobiHoc 2014 and the IEEE MASS 2013. He also received the ACM SIGMobile Service Award in 2014. He has served in various positions (as the Chair and a TPC member) at numerous conferences, including ACM MobiHoc, the IEEE INFOCOM, and the IEEE ICNP. He is an Editor of the *INFORMS Journal on Computing* and the *International Journal of Distributed Sensor Networks*.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, USTC. He has published over 100 articles in famous journals and conferences, including ToN, JSAC, TMC, TPDS, INFOCOM, and ICNP. He held over 30 patents. His main research interests are software-defined networks, edge computing, and the Internet of Thing. He received the Outstanding Youth Science Foundation from NSFC in 2018. He also received the best paper award or the best paper candidate in several famous conferences.

Yan Qiao received the B.S. degree in computer science and technology from Shanghai Jiao Tong University, China, in 2009, and the Ph.D. degree from the University of Florida in 2014. She joined Google Inc., as a Software Engineer. Her research interests include network measurement, algorithms, and RFID protocols.