You Can Drop but You Can't Hide: K-persistent Spread Estimation in High-speed Networks

He Huang[1], Yu-E Sun[2], Shigang Chen[3], Shaojie Tang[4], Kai Han[5], Jing Yuan[6], Wenjian Yang[1]

[1]School of Computer Science and Technology, Soochow University, China
[2]School of Rail Transportation, Soochow University, China
[3]Department of Computer and Information of Science and Engineering, University of Florida, US
[4]Naveen Jindal School of Management, University of Texas at Dallas, US
[5]School of Computer Science and Technology, University of Science and Technology of China, China
[6]Department of Computer Science, University of Texas at Dallas, US
E-mail: {huangh, sunye12}@suda.edu.cn, sgchen@cise.ufl.edu
*Yu-E Sun is the corresponding author.

***Abstract*—Traffic measurement in high-speed networks has many applications in improving network performance, assisting resource allocation, and detecting anomalies. In this paper, we study a new problem called k-persistent spread estimation, which measures persist traffic elements in each flow that appear during at least k out of t measurement periods, where k and t can be arbitrarily defined in user queries. Solutions to this problem have interesting applications in network attack detection, popular content identification, user access profiling, etc. Yet, it is under-investigated as the prior work only addresses a special case with a questionable assumption. Designing an efficient and accurate k-persistent estimator requires us to use bitwise SUM (instead of bitwise AND typical in the prior art) to join the information collected from different periods. This seemly simple change has fundamental impact on the mathematical process in deriving an estimator, particular over space-saving virtual bitmaps. Based on real network traces, we show that our new estimator can accurately estimate the k-persistent spreads of the flows. It also performs much better than the existing work on the special case of measuring elements that appear in all periods.**

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

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 [1], which counts the number of distinct elements in each network flow, 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. For instance, all packets from the same source address form a per-source flow, and we may measure the number of distinct destination addresses (i.e., elements) that each source (i.e., flow label) has contacted. For another example, all packets to each HTTP server form a flow, and we may measure the number of distinct files (elements) that are accessed from that server (i.e., flow label). Spread estimation has many important applications in scan detection, worm monitoring, proxy caching, and content access profiling [2]. We stress that spread estimation is different from counting the number of packets in each flow [3]–[5] or identifying elephant flows [6]–[8]. The former is to count the *distinct* number of elements, which requires duplicates to be filtered and is thus harder than the latter, which counts simply the number of packets.

This paper investigates a new problem called k-persistent spread estimation. Network traffic is summarized in compact traffic records over each measurement period 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 investigate the 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.* [9], which is a special case of our problem — finding the number of distinct elements that appear in a flow during all measurement periods. Their work can be motivated by an application example of detecting stealthy denial-of-quality attacks [9], where malicious hosts send a sufficient number of service requests to a server to slow the server down, without overwhelming the server to make the attack 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 the legitimate users by finding those that appear during all measurement periods. However, the attackers can easily circumvent such detection by dropping some periods, *i.e.,* give up sending packets during some (or even one) periods. In such a scenario, a new function with the ability of finding those that appear in at least k out of t periods becomes useful because it provides great flexibility of separating malicious hosts that have more intense activities than normal ones. More broadly, the function of measuring k-persistent spreads 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

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

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 measurement 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 high-speed cache memory (such as SRAM) on chip. It requires all online network functions such as routing, packet scheduling, quality of service, and traffic measurement to be performed efficiently in terms of both 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. Hence, this result does not tell how many of the t bits are ones, 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 already 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 bitwise AND changes the whole mathematical process in deriving an estimator for the more complex problem of finding the number of persist elements in each flow that appear in at least k out of t periods — a problem with interesting applications 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 all 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. Based on real network traces, we show that our new estimator based on bitwise SUM can accurately estimate the k-persistent spreads of the flows. It also performs much better than [9] for the special case of measuring elements that appear in all periods.

II. Preliminaries

A flow is a set of packets that share the same flow identifier, 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. As shown in Figure 1, 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. The spread of a flow is still defined as the number of distinct elements in the flow. Also shown in Figure 1, 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.

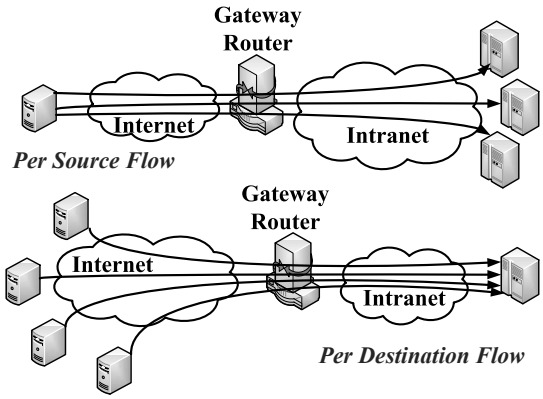

Fig. 1: An illustration of per-source flow and per-destination flow.

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 $1 \leq k \leq t$.*

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

This paper is to design the data structures and the algorithms that 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.

TABLE I demonstrates a possible application. The second line of the table shows the k-persistent spread of a flow consisting of packets sent to an HTTP server at $224.243.38.27/80$ in a real network traffic traces downloaded from CAIDA, and the third line shows the k-persistent spread of a flow to a server during World Cup 1998. The measurement period is 3

TABLE I: The relationship of k-persistent spread and the value of k.

k	1	2	3	4	5	6	7	8
k-persistent spread (CAIDA)	35693	1545	195	33	10	2	1	0
k-persistent spread (The World Cup 1998)	398	109	34	12	3	1	0	0

minutes, and there are 8 periods in total. It can be observed that in normal traffic, the k-persistent spread decreases rapidly as k increases. Persistent denial-of-quality attackers are likely to increase the k-persistent spread beyond its normal value, which makes such attacks detectable.

We observe that a legitimate user often connects to a server intermittently, while attackers would continuously send packets to the server, thus an appropriate k should be able to distinguish normal users from attackers.

III. Measuring k-Persistent Spread of One Flow

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 that will compute k-persistent spread of the flow based on the bitmaps it stores.

A. Online Recording of a Flow

Online recording is similar to [2] (though they solve different problems). The router creates a bitmap B_f of m bits to store a traffic record of flow f. Denote the ith bit of B_f as $B_f[i]$. At the beginning of each measurement period, it resets the bitmap to zeros. From each arrival packet of the flow, the router extracts $< e, f >$, 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 an application we discussed in the introduction. The router performs $H(e) \in [0, m)$ and sets $B[H(e)]$ to one, where $H(...)$ is a hash function.

B. Joining Bitmaps by Bitwise SUM

After a number t of measurement periods, the server stores a set of traffic records, $\mathcal{B} = \{B_1, ..., B_t\}$, 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 [9] 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 ones, 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 js. Its value can be found by counting the number of js 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 ones. 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 ones while all bitmaps in c_{-j} have their ith bits stay as zeros. We have

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

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.

1) Event H_1: none of the $(j+1)$-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 $(j+1)$-persistent element not being hashed to the ith bit is $1-\frac{1}{m}$. The number of $(j+1)$-persistent elements is $N-\sum_{l=1}^{j} n_l$. We have

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

2) Event H_2: Any 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 appears at 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 elements appearing at the router in l periods. Hence,

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

Combining the above analysis, we have

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

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

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

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 ones, while the ith bits of all other bitmaps in c_j are zeros. 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,

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

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

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

Substituting (7) to (1), we have

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

Next we show that $\Pr\{S[i]=j\} = E(V_j)$, where V_j is the fraction of counters in S that are js. 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: $\forall\ 1 \leq j \leq t$,

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

where

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

$$b = (N - \sum_{l=1}^{j-1} \hat{n}_l) \ln(1 - \frac{1}{m}),$$

$$c = \sum_{l=1}^{j-1} \hat{n}_l \ln(1 - \frac{C_t^l - C_j^l}{mC_t^l})$$

.

We invoke the above estimator in the order of $j = 1, 2, ..., t$. 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 element not to set the ith bit is $1 - \frac{1}{m}$ and there are N independent elements in total, we have

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

where V_0 is the fraction of counters in S whose values are zeros. 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

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

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

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

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 $\hat{n}^*$.

Algorithm 1: Estimator for k-persistent spread

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

IV. Measuring k-persistent Spreads of Numerous Flows Simultaneously

In the previous section, we consider the case of a single flow, which provides a basic estimator to support the measurement of k-persistent spreads of many flows simultaneously, as we will discuss in this section.

Most of the flow spreads in real traffic from CAIDA are less than 100. However, the largest spread among all flows reaches 10^6 although the number of elephant flow 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 measure the largest spread. We point out in the introduction that the size of on-chip SRAM is typically small. This will limit the number of concurrent flows that the router can handle at high line speed.

To address the above problem, Yoon et al. introduced the concept of virtual bitmap in [2]. Their idea is to let the bitmaps of different flows to share a common pool of bits, instead of occupying separate memory space. Logically, the router allocates a virtual bitmap to each flow; physically, these virtual bitmaps share their bits from the common bit pool. 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 [2] is not on persistent spread estimation, but its concept of virtual bitmap has been borrowed by [9] to measure the number of persistent elements in *all* measurement periods. As we point out earlier, this work is a special case of k-persistent measurement. It finds the number of elements that appear in t out of t measurement periods, not k out of t periods, $k \leq t$, that we study in this paper. We also point out in the previous section that their bitwise AND operation is lossy and cannot be used to find k-persistent spread. Moreover, there is a questionable assumption in [9] 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.

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

A. Recording Flows 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.

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

TABLE II: 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

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 zeros. From each arrival packet, the router extracts $< e, f >$, where e is the element to be recorded and f is the flow label. It hashes e to the $H(e)$th bit in the virtual bitmap $B_f[i]$. However, we cannot set that bit 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, $\mathcal{B} = \{B_1, B_2, ..., B_t\}$. 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 $\mathcal{B}_f = \{B_{f,1}, B_{f,2}, ..., B_{f,t}\}$. 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). 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 number of elements that persistently stay in flow f during j out of t measurement periods, and $n^m_{f,j}$ be the number of elements that are recorded in $\mathcal{B}_f$ and appear during j out of t periods. 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 $\mathcal{B}_f$.

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 can compute an estimation $\hat{N}^u_j$ for N^u_j, $1 \le j \le t$, using the method in Section III based on the grand flow's traffic records in $\mathcal{B}$. Among these elements, $N^u_j - n_{f,j}$ of them belong to other flows are the noise from the view point of flow f. These noise elements are randomly recorded by the u bits in the physical bit array. Recall that flow f randomly take m bits from the array to form its virtual bitmap. Each noise element has a probability of $\frac{m}{u}$ to set a bit in f's virtual bitmap. We have the following mean noise in the virtual bitmap,

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

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

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

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

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

V. Simulation

In this section, we evaluate the performance of our estimator through extensive experiments using real Internet traffic traces. Recall that we aim to design a k-persistent spread estimator which can work well on on-chip SRAM with tight space and achieve high estimation accuracy. Thus, the memory size in our setting ranges from 1.6 to 6.8 bits per element. The only existing work that can apply to such a small memory is the method proposed by Xiao *et al.* (MVPE for short) [9]. We compare our methods with MVPE in terms of estimation accuracy and operating range. The numerical results show that our estimator can accurately estimate the k-persistent spreads in a real network traffic trace. When $k = t$, our solution significantly outperforms MVPE. Moreover, the results show that our estimator effectively detects stealthy attackers that cannot be detected with MVPE.

A. Experiment Setup

Our dataset contains one hour of data downloaded from CAIDA. It has 38963 distinct flows, and 7179130 distinct elements. We observe that the flow spreads are distributed extremely unbalance (as shown in Table II). The spreads of more than 91% flows are less than 200. However, the largest spread of elephant flow reaches 319807.

In our experiments, we set 5 minutes as one measurement period, and one estimation period is composed of 8 measurement periods, *i.e.* $t = 8$. The number of distinct elements in each measurement period is 1081531, 1088492, 1089644, 1084794, 1225640, 1092274, 1111706, 1094310, separately. We fit estimators into a 0.25MB $\sim$ 1MB on-chip SRAM.

B. Estimation Accuracy and Operating Range

In this part, we first compare the estimation accuracy of our estimator and MVPE. Table. III shows the results of 6 chosen flows with spread distributed in the range $(10000, 50000)$. We demonstrate the true k-persistent spreads and estimated k-persistent spreads of our estimator in that table. It is obvious that our estimator performs much better than MVPE in terms of higher estimation accuracy. This is mainly because MVPE assumes that the legitimate users are independent among different measurement periods. However, many legitimate users will stay multiple measurement periods when we set 5-minutes as one measurement period. To overcome this shortcoming, MVPE needs to increase the length of each measurement period, which will increase the number of distinct elements in each measurement period, and reduce its operating range. Moreover, the experiment results also validate the robustness of our estimator under different k values ($1 \leq k \leq t$). As shown in Section V-C, we can detect the stealthy attacks based on an accurate estimation of k-persistent spreads.

Fig. 2 and Fig. 3 present the experimental results when $m = 65536$b and $u = 0.25, 0.5, 0.75, 1$MB, respectively. The x-coordinate is the actual spread of 8-persistent traffic and the y-coordinate is the estimated spread of 8-persistent traffic. The experimental results show that our estimator works better than the MVPE, especially when the persistent spread is large. The estimation accuracy of the MVPE when $u = 1$MB is even worse than our estimator when $u = 0.25$MB.

Our estimator has two configurable parameters: the virtual bitmap size m and the physical bit array size u. Fig. 3 states that the accuracy of our estimator increases with u when u is small, then becomes stable when $u > 0.75$MB. The accuracy of our estimator is still acceptable when $u = 0.25$MB, which is about 1.6 bits per element. In Fig. 4, we fix $u = 0.5$MB, and set $m = 16384, 32768, 65536, 131072$b, separately. On one hand, the estimation accuracy increases as m increases if we allocate a separate bitmap to each flow (As shown in Fig. **??**). On the other hand, when using our solution with virtual bitmap, a larger m will cause more noise introduced by other flows. Hence, the estimation accuracy of flows increases as m increases at the beginning, and then decreases when m is large enough.

C. 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. III, 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 simulation results are shown in Table. IV.

Based on the observation of the flows in the real Internet traffic traces used in our simulation, 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, such as the flows in Table. I and Table. III. Recall that we assume that the stealthy attackers will keep connection with the target server significantly longer than legitimate users. Thus, an appropriate k should be able to distinguish legitimate users from stealthy attackers. This value could be learned from the historical information without being attacked. For example, since most of 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 set 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 simulation, we treat 4-persistent elements as illegal users. Based on the simulation 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, simply estimating the t-persistent spread can not tell whether a server is under attack or not. For example, the true value of 8-persistent spread of server 3.46.30.10 is only 13, which is much smaller than 62638 (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.

VI. Related Work

To the best of our knowledge, there is no prior work on k out of t persistent spread measurement. As we have mentioned before, the measurement of distinct elements persist in traffic flow for at least k out of t predefined time periods can greatly help us to detect various network attacks (*e.g.*, DDoS stealthy attack, stealthy network scan), proxy caching, and content access profiling. 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. For the individual flow measurement, the studies can be classified into two categories. The first one is the flow size estimation, and another line of research is the flow spread estimation.

Flow size often 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 [5]. [10], [11] 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 [2], [12], [13]. In [2], 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.

In recent years, 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

TABLE III: The actual and estimated k-persistent spread when $u = 0.5$MB, $m = 32768$b

Server Address		Our estimator based on virtual bitmap								MVPE
		$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$t=8$
146.25.164.98	Actual Spread	16406	3729	1580	828	450	247	140	50	50
	Estimated Spread	15916	3858	1731	808	347	273	143	71	173
3.46.30.10	Actual Spread	31319	8438	1669	517	168	52	23	13	13
	Estimated Spread	30635	8644	1704	516	297	297	21	17	175
36.1.82.240	Actual Spread	36644	8197	1636	353	157	91	20	6	6
	Estimated Spread	36010	7993	1519	924	67	66	48	38	191
70.63.100.108	Actual Spread	32869	11451	4794	2445	1244	658	363	216	216
	Estimated Spread	31311	11776	5297	2551	1361	662	366	219	534
72.192.80.170	Actual Spread	35846	8557	3173	1505	830	450	247	128	128
	Estimated Spread	35199	8448	3722	1014	662	455	165	57	305
92.79.46.194	Actual Spread	41804	7181	2493	1173	607	330	178	84	84
	Estimated Spread	41590	8268	2405	1045	1036	304	211	75	342

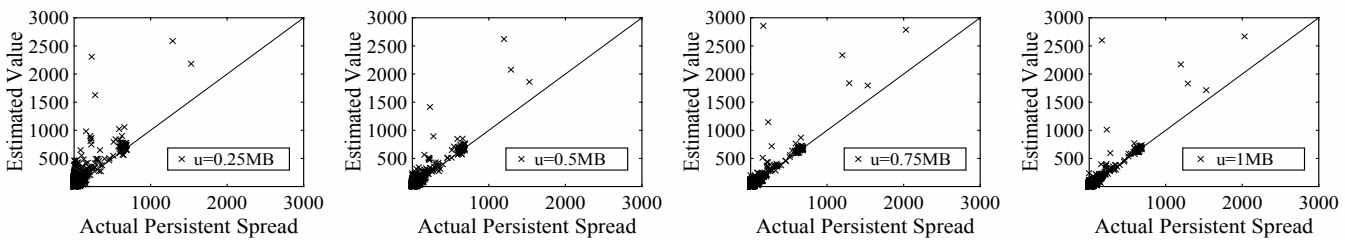

Fig. 2: t-persistent spread estimate using MVPE, with $m = 65536$b, $t = 8$.

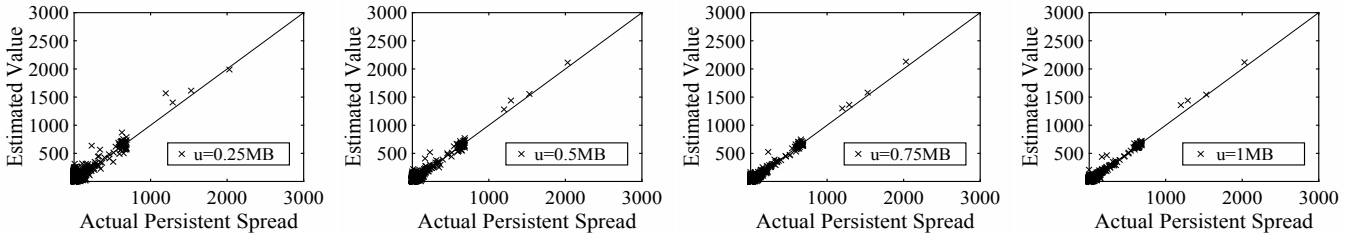

Fig. 3: k-persistent spread estimate using our estimator, with $m = 65536$b, $t = 8$, $k = 8$.

attack or network scan. In [9], 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. Dai et al. [14] concentrate on finding the persistent items in data streams. Literature [15] studies the privacy preserving persistent traffic measurement for the intelligent transportation [16]. 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 [17], [18], [19], [20], [21], 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.

VII. Conclusion

We notice that a sophisticated attacker may drop some measurement periods to avoid existing detection that is based on persistent spread estimation. In this paper, we propose a new 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. Our scheme can achieve high accuracy and fit into a size constrained memory. Even when the attackers drop some measurement periods, our k-persistent spread estimator can still detect those stealthy attacks. We conduct extensive experiments using real network traffic traces and the experiment results show that our estimator, requiring memory space with only 1.6 bits per element, can detect the stealthy DDoS attack effectively and accurately.

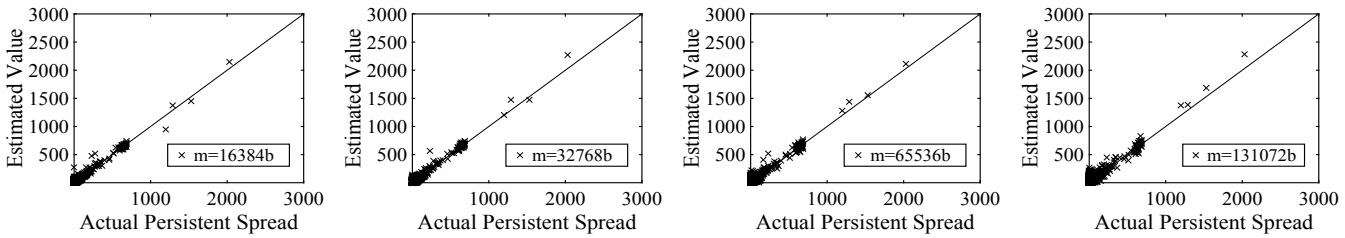

Fig. 4: k-persistent spread estimate using our estimator, with $u = 0.5$MB, $t = 8$, $k = 8$.

TABLE IV: The actual and estimated k-persistent spread when $u = 0.5$MB,$m = 32768$b

Server Address		Our estimator based on virtual bitmap							
		$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
146.25.164.98	Actual Spread	32812	20135	17986	17234	11438	5754	140	50
	Estimated Spread	32334	20586	18141	17167	11442	5670	226	40
3.46.30.10	Actual Spread	62638	39757	32988	31836	21001	10452	23	13
	Estimated Spread	61407	40694	32782	31884	21204	10763	182	103
36.1.82.240	Actual Spread	73288	44841	38280	36997	24565	12300	20	6
	Estimated Spread	73549	46008	38841	38841	25095	12588	19	19
70.63.100.108	Actual Spread	65738	44320	37663	35314	23340	11623	363	216
	Estimated Spread	64133	45090	39888	36255	24707	11086	368	368
72.192.80.170	Actual Spread	71692	44403	39019	37351	24765	12375	247	128
	Estimated Spread	71224	44255	39538	37643	25282	13065	520	0
92.79.46.194	Actual Spread	83608	48985	44297	42977	28449	14216	178	84
	Estimated Spread	82582	49773	44599	42503	29133	14702	461	111

ACKNOWLEDGEMENT

The research of authors is partially supported by National Science Foundation (NSF) CNS-1719222, STC-1562485, National Natural Science Foundation of China (NSFC) under Grant No. 61572342, No. 61672369, Natural Science Foundation of Jiangsu Province under Grant No. BK20151240, No. BK20161258. The research of Kai Han is partially supported by NSFC under Grant No. 61472460, No. 61772491, NSF of Jiangsu Province under Grant No. BK20161256. This work is also supported by the grant from Florida Cybersecurity Center.

REFERENCES

[1] M. Yoon, T. Li, S. Chen, and J. kwon Peir, "Fit a Spread Estimator in Small Memory," *Proc. of INFOCOM*, pp. 504–512, 2009.

[2] M. Yoon, T. Li, S. Chen, and J. Peir, "Fit a Compact Spread Estimator in Small High-Speed Memory," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1253–1264, October 2011.

[3] S. Ramabhadran and G. Varghese, "Efficient Implementation of a Statistics Counter Architecture," *Proc. ACM SIGMETRICS*, vol. 31, no. 1, pp. 261–271, June 2003.

[4] Q. Zhao, J. Xu, and Z. Liu, "Design of a novel statistics counter architecture with optimal space and time efficiency," *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, pp. 323–334, 2006.

[5] M. Chen, S. Chen, and Z. Cai, "Counter tree: A scalable counter architecture for per-flow traffic measurement," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 2, pp. 1249–1262, 2017.

[6] 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.

[7] 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.

[8] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," *Proc. of ACM SIGCOMM*, pp. 101–114, 2016.

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

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

[11] 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.

[12] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 925–937, 2006.

[13] M. ROESCH, "Snort-lightweight intrusion detection for networks," *LISA'99: Proc. 13th USENIX Conference on System Administration*, vol. 99, no. 1, pp. 229–238, 1999.

[14] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong, "Finding persistent items in data streams," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 289–300, 2016.

[15] H. Huang, Y.-E. Sun, S. Chen, H. Xu, and Y. Zhou, "Persistent traffic measurement through vehicle-to-infrastructure communications," in *Proc. of ICDCS 2017*, 2017.

[16] Z. Tang, A. Liu, Z. Li, Y. june Choi, H. Sekiya, and J. Li, "A Trust-Based Model for Security Cooperating in Vehicular Cloud Computing," *Mobile Information Systems*, 2016.

[17] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A Linear-time Probabilistic Counting Algorithm for Database Applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, June 1990.

[18] C. Estan, G. Varghese, and M. Fish, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *IEEE/ACM Trans. on Networking*, vol. 14, no. 5, October 2006.

[19] 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.

[20] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," *European Symposium on Algorithms*, pp. 605–617, 2003.

[21] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *AofA: Analysis of Algorithms*, 2007, pp. 137–156.