# A Generic Technique for Sketches to Adapt to Different Counting Ranges

Tong Yang[1], Jiaqi Xu[1], Xilai Liu[1], Peng Liu[1], Lun Wang[1], Jun Bi[2], Xiaoming Li[1]
[1]Department of Computer Science, Peking University, China
[2]Institute of Network Science & Cyberspace, Tsinghua University, China

***Abstract*—Sketch is a compact data structure for network measurements. To achieve fast speed, it needs to be held in the on-chip memory (SRAM), which is very small. To enable the sketch fit into the on-chip memory, the product of counter size and number of counters must be below a certain limit. If we use small counters, *e.g.*, 8 bits, some counters will overflow. If we use large counters, *e.g.*, 16 bits per counter, the total number of counters will be small, each counter will be shared by more flows, leading to poor accuracy. To address this issue, we propose a generic technique: *self-adaptive counters (SA Counter)*. When the value of the counter is small, it works as a normal counter. When the value of the counter is large, we increment it using a predefined probability, so as to represent a large value. Moreover, in SA Counter, the probability decreases when the value increases. This technique can significantly improve the accuracy of sketches. To verify the effectiveness of SA Counter, we apply SA Counter to three typical sketches, and conduct extensive experiments on one real dataset and one synthetic dataset. Experimental results show that, compared with the state-of-the-art, sketches using SA Counter improve the accuracy by up to 13.6 times.** [*]

## I. Introduction

### A. Background and Motivation

Network measurements provide indispensable information for congestion control [1], [2], DDoS attack detection [3], [4], heavy hitter identification [5]–[7], heavy change detection [8], and more [9], [10]. Measuring the size of different flows (known as per-flow size measurements, or *per-flow measurements for short*) in network traffic has attracted attentions in recent years [11]–[14]. Flow identifiers (flow IDs) are selected from the header fields of packets, such as the five-tuple: source IP address, source port, destination IP address, destination port, protocol. Flow size is defined as the number of packets in a flow. Flow volume is defined as the number of bytes in the flow.

As the line rate can be high, *e.g.*, 10Gbps, 40Gbps, it is challenging to perform per-flow measurements at line rate. To achieve high processing speed, the data structure should be small enough to be stored in the on-chip memory, such as a Block RAM in FPGA or ASIC chips [15], or the caches of CPU or GPU chips. However, the size of on-chip memory is very limited (usually less than 8.25MB [15]). This means that it is almost impossible to keep one counter for each flow to record the flow size. To achieve memory efficiency, various sketches (*e.g.*, sketch of CountMin (CM) [16], Conservative Update (CU) [17] and Count (C) [18]) allow counters to be randomly shared by multiple flows, inevitably incurring errors.

In real network traffic, it is well known that the flow size/volume distribution is highly skewed [14], [17], [19]–[24]. Specifically, most flows are very small in size, often known as mouse flows; while a few are very large, often known as elephant flows. As elephant flows are typically more important than the small ones, the size of each counter needs to be large enough to store the largest flow. Further, in many practical scenarios, one does not have any idea of the approximate flow size of elephant flows beforehand. Given the limited size of on-chip memory, if we use large counters, the number of counters will be small, and each counter will be shared by more flows, leading to poor accuracy. In this case, most counters are mapped by mouse flows, keeping a small value, and thus their significant bits are wasted. Therefore, if we use large counters, it will be a waste of memory, and the accuracy will be poor. In summary, it is challenging to achieve memory efficiency in skewed network traffic.

### B. Prior Art and Their Limitations

There are primarily two kinds of algorithms for per-flow measurements. The first kind is based on sampling [25]. However, recent works [12], [13], [26] pointed out that the sampling method might lose important information. The other kind is based on a compact data structure called sketch. There are three classic sketches: sketches of Count [18], CM [16], and CU [17]. The CM sketch is the most widely used one. These sketches share the same data structure and similar operations. Therefore, we only present the details of the CM sketch here. A CM sketch consists of $d$ arrays, each of which is associated with a hash function, denoted by $h_1(.) \dots h_d(.)$. The i$^{th}$ array is represented by $A_i$. Each array consists of $w$ counters. When inserting a packet with flow ID $e$, the CM sketch adds the packet size of $e$ to the $d$ counters: $A_1[h_1(e)] \dots A_d[h_d(e)]$, called the $d$ *mapped counters* in this paper. When querying item $e$, the CM sketch reports the minimum value among $d$ *mapped counters*: $min\{A_1[h_1(e)] \dots A_d[h_d(e)]\}$. Obviously, the accuracy of the CM sketch will decrease as the number of counters decreases. However, as each counter needs to be large enough to store the largest flows, the CM sketch cannot achieve memory efficiency, so do sketches of C and CU. To achieve memory efficiency, small counters must be used. The state-of-the-art named Counter tree [27] uses multiple layers of small counters, and the counters at the higher layers are used to store the significant bits of the elephant flow sizes. In

[*]Tong Yang and Jiaqi Xu contributed equally to this work. Jun Bi (junbi@tsinghua.edu.cn) is the corresponding author.

this way, it can improve the memory efficiency. Unfortunately, for each packet belonging to an elephant flow, Counter-Tree needs to access all layers, requiring many memory accesses for each insertion. This problem gets worse when elephant flows dominate the network traffic. It is therefore very important to achieve memory efficiency, while ensuring fast and constant processing speed to keep up with line rate.

### C. Our Solution

In this paper, we propose a generic technique aimed at making every bit count. Our technique is applicable for all sketches using counters. Recalling that the design goal is to achieve both memory efficiency and fast, constant speed. To achieve memory efficiency, we have to use small counters. To achieve constant and fast processing speed, to make the data structure as easy to use by real applications, we do not introduce multiple layers or change the basic data structure of sketches. Our challenge is that each small counter has to be able to represent the size of both mouse and elephant flows.

Our key idea is simple: *when a counter is going to overflow, for each insertion, instead of always increasing it, we increase it with a predefined probability.* When the value of the counter is small (*e.g.*, is mapped by one or several mouse flows), we consider it as a normal counter, so as to achieve high accuracy. By carefully designing the probability setting, we can use a small counter to represent a very large value. Introducing this probability could incur inaccurate recording of large values. Fortunately, we prove theoretically and experimentally that the incurred inaccuracy is negligible compared to the large value of the size of elephant flows.

Based on the above idea, we propose two versions of our technique: Static Sign Bits version and Dynamic Sign Bits version. Our technique splits each counter into two parts: (1) sign bits, and (2) counting bits. When the sign bits are all 0, we increment the counting bits normally; when the sign bits are non-zero, we increment the counting bits with a probability calculated by the value of the sign bits. As the number of bits of each counter is fixed, if we assign more bits as sign bits, the number of counting bits will decrease. In this case, the counter cannot accurately record the size of mouse flows.

For the static version, we fix the length of sign bits in advance, and then do not change it during insertions. As a result, the shortcoming of the static version is that it is hard to determine how many bits should be assigned for the sign bits. To address this issue, we propose the Dynamic Sign Bits version, which uses a self-adaptive method, and can dynamically adjust the length of sign bits according to the value that the counter needs to represent. The self-adaptive method works as follows: we regard all continuous 1 bits from the left as sign bits. The leftmost 0 bit is seen as a marker, namely, the split bit, and others are counting bits. For example, given a counter with value 11101011, it means the following. The first three bits are sign bits, representing a value of 3. The fourth bit 0 is the split bit which splits the sign and counting bits. The last four bits 1011 are the counting bits. More details about the self-adaptive methods are provided in Section III-B.

### D. Key Contributions

1) We propose a generic sketch technique in two versions, static and dynamic. Using our technique, sketches can use small counters to accurately record the sizes of both elephant and mouse flows, achieving memory efficiency as well as constant and fast speed. We have applied our technique to sketches of CM, CU and C.
2) We conduct a detailed analysis of self-adaptive counters to show their theoretical properties.
3) We carry out extensive experiments, and provide results on two real datasets and one synthetic dataset, demonstrating the superior performance of self-adaptive counters.

## II. Related Work

For per-flow measurement, there are two kinds of methods: sampling and sketches. The authors in [28] prove that sampling has poor accuracy, and propose an algorithm to improve the accuracy. To improve the accuracy, various sketches are proposed, including CM sketch [16], CU sketch [17], C sketch [18] and many other advanced techniques like A sketch [14], CSM sketch [29], Tug-of-war sketch [30] and its enhanced version [31], AMS sketch [32], and more [33]–[36]. Due to space limitation, we briefly introduce typical sketches.

### A. CU Sketch

The CU sketch [17] has the same structure as the CM sketch, but its insertion strategy is optimized using "conservative update". When inserting a packet $e$, it only adds the metric of interest of $e$ to the smallest counter(s) among $d$ *mapped counters*. When querying packet $e$, the CU sketch reports the minimum value among $d$ *mapped counters*.

### B. C Sketch

The C sketch [18] is different from the CM and CU sketch in the sense that each array is associated with two hash functions. Besides $d$ hash functions $h_1(.) \cdots h_d(.)$, there are additional $d$ hash functions $s_1(.) \cdots s_d(.)$ mapping each incoming packet to $\{+1, -1\}$. If the result of the second hash function is $+1$, then the insertion proceeds as usual. Otherwise, the metric of interest of the packet will be subtracted from the $d$ *mapped counters*. When querying, it will report the absolute value of the median one of ${}_i\{A[h_i(key)] \times s_i(key)\}$, where $i$ is in $0, 1, \cdots, d-1$.

### C. Sophisticated Sketches

Many advanced sketch techniques have been proposed recently [37]–[39]. The augmented sketch (A sketch) [14] is targeted at improving accuracy by using one additional filter to dynamically capture packets from heavy-hitters. The CSM sketch [29] provides a data encoding and decoding scheme. Two offline statistical methods are proposed in this sketch in order to extract query results.

The Counter-Tree [27] provides a scalable architecture for per-flow measurements. The key idea of the Counter-Tree is to rearrange the counter sharing scheme based on the CM sketch. The authors claim that their 2-D tree structure is able

to work with very tight space. However, in terms of accuracy, the performance of the Counter-Tree is not as good as the original CM sketch under the same memory usage. Moreover, the CM sketch and other typical sketches support the $(ID, f)$-insertion where $f \geqslant 1$. The Counter-Tree on the other hand only supports $(ID, 1)$-insertions, which makes it unsuitable in applications like flow volume measurements.

## III. The SA Counter Technique

In this section, we describe our SA Counter (Self-Adaptive Counters) approach. We propose two versions of SA Counter, a Static Sign Bits version and a Dynamic Sign Bits version. Inspired by floating-point number representation, the Static Sign Bits version uses some *sign bits* to adjust the magnitude of the counters, to enable the counters to represent larger values. However, it is hard to determine an appropriate length for the sign bits so that the mouse flows can be accurately recorded. To overcome the weakness of the static version, we further design a dynamic version.

### A. Static Sign Bits Version

**Rationale:** Given a sketch, let $n$ be the counter size, *i.e.*, the number of bits in a counter. To achieve memory efficiency, we need to use fine-grained counters, *i.e.*, small counters (*e.g.*, $n = 8$). The capacity of the counters, *i.e.*, the maximum value that counters can represent, is fixed to $2^n$-1. However, when a counter is mapped by too many flows, especially large ones, the value of the counter will easily exceed $2^n$-1.

Our technique is inspired by, but different from the encoding style of *floating-point numbers*. In a typical floating-point representation, the value can be calculated using the following three parts: 1) a *sign digit* indicating the value to be positive or negative. 2) *exponent digits* which represent an integer that controls the magnitude of this representation. 3) *significant digits* that carry meaning contributing to its measurement resolution. Similarly, in our technique, we also split a counter into two parts: a *Sign Bits* part (sign part for convenience) and a *Counting Bits* part (counting part for convenience). The counting part functions as the significant digits, while the sign part functions as the exponent digits. Specifically, for each possible value of the sign part, we pre-define a corresponding integer indicating how many times the counting part should be expanded. We call this pre-defined array the *expansion array*.

**Self Adaptive Counter (SA Counter):** Our technique is called Self-Adaptive Counters (SA Counter). Every counter of existing sketches can be replaced by a Self-Adaptive Counter (SA Counter). Next we show the data structure and operations of SA Counter.

**Data Structure:** Assume a sketch has $d$ counter arrays, $A_1, A_2, \cdots, A_d$. For the $i$-th counter array, it has $w$ counters and a hash function $h_i(.)$. Let $n$ be the number of bits in the counters. For each counter, it has a $s$-bit sign part and a ($n$-$s$)-bit counting part as shown in Figure 1. We denote the expansion array as $\gamma[0], \gamma[1], \cdots, \gamma[k-1]$, where $k = 2^s$. For example, $\gamma[i] = 2^i$. After setting up all the above parameters, the capacity of the Static Sign Bits version SA Counter is $C_{static}(n, s) = \sum_{i=0}^{2^s-1}(\gamma[i] \times 2^{n-s})$.

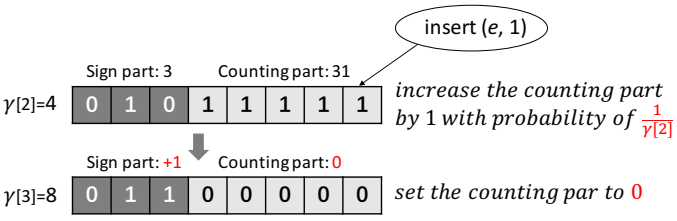



Fig. 1. An example of SA Counter with Static Sign Bits, where $n = 8$ and $s = 3$. To insert a pair $(e, 1)$, the counting part is increased by 1 with probability of $\frac{1}{4}$. If the counting part overflows, we increase the sign part by 1, and get a new expansion parameter $\gamma[3] = 8$.

**Insertion:** Take the CM sketch using Static Sign Bits version SA Counter as an example. We show the procedure to insert a flow with volume 1. The procedure to insert a flow with larger volume can be seen in **Algorithm 1** in Section IV.

To insert a flow with size 1, we first locate $d$ counters using $d$ hash functions. Next, we show the steps of how to add 1 to an SA Counter.

*Step 1*: In an SA counter, we get the sign part $s_0$, the counting part $c_0$, and the value $\gamma[s_0]$ from the expansion array.

*Step 2*: Since $\gamma[s_0]$ indicates how many times the counting part should be expanded, we first calculate $\frac{1}{\gamma[s_0]}$, and add 1 to the counting part with probability $\frac{1}{\gamma[s_0]}$.

*Step 3*: If the counting part reaches $2^{n-s}$, we increase the sign part by 1, and set the counting part to zero.

**Query:** To query a flow in a sketch using our SA Counter with Static Sign Bits, we first locate several counters using $d$ hash functions. Then, for a mapped counter $\mathcal{C}$, we calculate the value represented by $\mathcal{C}$ as follows:

1) First, we get the value of the sign part $s_0$ and the value of the counting part $c_0$. Then, we find $\gamma[s_0]$ from the expansion array and another value $stage[s_0]$ from the *stage array*. The stage array is pre-computed using the following formula:

$$\begin{cases} stage[0] = 0, \\ stage[i] = 2^{n-s} \times \sum_{j=0}^{i-1} \gamma[j], \ i > 0 \end{cases} \tag{1}$$

2) The value represented by $\mathcal{C}$ can be calculated with the following formula:

$$value(\mathcal{C}) = c_0 \times \gamma[s_0] + stage[s_0]. \tag{2}$$

The problem of the Static Sign Bits version is that, since we do not know the size of the mouse flows, we cannot determine an appropriate length for the counting and sign parts. Specifically, when the sign part is zero, the flow size is accurately recorded. When we use a large sign part, the counting part will be shortened, and thus the counter may not accurately record the mouse flows.

### B. Dynamic Sign Bits Version

**Rationale:** Given a fixed counter size, to address the issue of the space taken by a fixed-length sign part, we can use a self-adaptive method to dynamically adjust it. The length of

the sign part is initialized to 0. Except for the split digit, all other bits are used for counting. As the value represented by the counter becomes larger, we increase the length of the sign part dynamically. In this way, we can accurately record mouse flows, while being able to deal with elephant flows.

**Data Structure:** An $n$-bit Dynamic Sign Bits version SA Counter has three parts: 1) a sign part whose length $l_s$ is made up by ones, 2) a *split digit* which is the leftmost zero digit, 3) a ($n$-$l_s$-1)-bit counting part. We create an expansion array $\gamma[0], \gamma[1], \cdots, \gamma[n-2]$. After setting up the above parameters, the capacity of the Dynamic Sign Bits version SA Counter is $C_{dynamic}(n) \sum_{i=0}^{s-2}(\gamma[i] \times 2^{n-i-1})$.

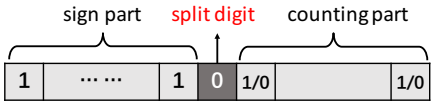



Fig. 2. Structure of Dynamic Sign Bits SA Counter.

**Insertion:** The insertion process of the dynamic version of SA Counter is similar to the static version, except for two differences: (1) In the dynamic version, the value of the sign part, *i.e.*, $s_0$, is equal to the number of ones in the sign part; For example, the value of the sign part in"111011" is 3. (2) In the dynamic version, when the counting part overflows, we turn the split digit to 1, and set the bits of the counting part to all zeroes. By doing this, the length of the sign part is increased by 1, the split digit is moved right by 1 bit, and the counting part is shortened by 1 bit.

**Query:** The query process of the dynamic version of SA Counter is similar to the static version. To calculate the value of an SA Counter with Dynamic Sign Bits, the first step is also to get $s_0$ and $c_0$ from the counter, and read the value $\gamma[s_0]$ and $stage[s_0]$ from the expansion and stage arrays, respectively. The two differences are that $s_0$ is the length of the sign part, and the stage array is computed by the following formula:

$$\begin{cases} stage[0] = 0 \\ stage[i] = \sum_{j=0}^{i-1} \left(\gamma[j] \times 2^{n-j-1}\right), \ i > 0 \end{cases} \tag{3}$$

The second step is the same as the static version. The value represented by a dynamic version SA Counter can be calculated using Formula 2.

## IV. Case Studies

To further illustrate the generality of the SA Counter technique, in this section, we show how to apply SA Counter to the sketches of CM [16], CU [17] and C [18]. Two functions, $read$ and $update$ of SA Counters are shown as follow.

### A. Apply to the CM Sketch

In CM sketch, the insertion procedure is done by calling the "$update$" function of the corresponding SA Counter. The query function is done by reporting the minimum value "$read$" by all corresponding counters.

### B. Apply to the CU Sketch

The CU sketch is similar to the CM sketch but with a different update technique called "conservative update of counters". In a CU sketch, the insertion procedure is done by calling the "$update$" function on the smallest counters in SA Counter. The query process is the same as for the CM sketch.

### C. Apply to the C Sketch

The C sketch consists of an array with $t \times k$ counters. One important feature of the C sketch is that it requires two sets of hash functions $h[1] \cdots h[t]$ and $g[1] \cdots g[t]$, where $h[i] : [0, n] \to [0, k]$ , $g[i] : [0, n] \to \{-1, 1\}$. Negative values can occur in the counters, so the first bit of each counter should be the sign bit. When inserting a packet, the sketch calls the "$update$" procedure of SA Counter to add the value to the corresponding counter. The query function is done by reporting the median value "$read$" by all corresponding counters.

Since we may add a negative number to the counters in the C sketch, in the two functions, *i.e.*, $read$ and $update$, we use the leftmost bit of a SA Counter to indicate whether the counter is positive or negative. As shown in Algorithm 1, the $read$ function takes a counter $c$ and the expansion array $\gamma$ as input, and outputs the value represented by the counter. As shown in Algorithm 1, the $update$ function takes a counter $c$, an increment value $c$, and the expansion array $\gamma$ as input.

**Algorithm 1:** $read$ and $update$ function of SA Counters

**read:** (a counter $c$, an expansion array $\gamma$)
1: $d = c$
2: **if** $c < 0$ **then**
3: $\tilde{c} + 1 \mapsto d$, thus $d$ is the two's complement of $c$
4: $s_0$: value of the sign part in $d$
5: $c_0$: value of the counting part in $d$
6: $result = stage[s_0 - 1] + (\gamma[s_0] \times c_0)$
7: **if** $c < 0$ **then**
8: $-result \mapsto result$
9: **return** $result$

**update:** (a counter $c$, a value $v$, an expansion array $\gamma$)
1: **if** read($c, \gamma$) $+v \leqslant C_{dynamic}(n)$ **then**
2: $d = c$
3: **if** $c < 0$ **then**
4: $\tilde{c} + 1 \mapsto d$, thus $d$ is the two's complement of $c$
5: $s_0$: value of the sign part in $d$
6: $c_0$: value of the counting part in $d$
7: $q = \frac{v}{\gamma[s_0]}$, $r = v\%\gamma[s_0]$
8: **if** $r \neq 0$ **then**
9: $p$: a random number in [0, 1]
10: **if** $p < \frac{r}{\gamma[s_0]}$ **then**
11: $c + 1 \mapsto c$
12: **if** $0 < q < stage[s_0] - c_0$ **then**
13: $c + q \mapsto c$
14: **if** $q \geqslant stage[s_0] - c_0$ **then**
15: $v' = v - (stage[s_0] - c_0) \times \gamma[s_0]$
16: $stage[s_0] \mapsto x$
17: update($c, v', \gamma$)

## V. Experimental Results

In this section, we first evaluate the SA Counter technique on a real-world dataset by comparing the Average Relative Error (ARE) and the Average Absolute Error (AAE) between sketches without SA Counter, with SA Counter and Counter-Tree. Then, we generate a synthetic dataset which follows the Zipf distribution. We study how the ARE and AAE values change with different parameters. These parameters are defined in Section V-B.

### A. Experimental Setup

*1) Datasets:*

$a$) IP Trace Datasets: We use the anonymous IP traces collected in 2016 from CAIDA [40]. In the experiments, a five-tuple is used as the ID of a flow, which includes, source IP address, destination IP address, source port, destination port, and protocol. Each arriving packet consists of a certain amount of bytes. The job of the sketch algorithm is to add up the number of bytes for each flow separately, and report the total size (in bytes) for every flow in the data stream at the end of a certain time period.

$b$) Synthetic Datasets: Since our goal is to find out how well SA Counter sketch performs on datasets with different characteristics, we also generate synthetic datasets following the **Zipf** [41] distribution ($p(x) = \frac{x^{-a}}{\zeta(a)}$) with different total flow sizes $F$ (1M to 10M), different numbers of packets $N_d$, and different $a$ (from 0 to 3.0 with a step of 0.3). We generate them using a performance testing tool: Web Polygraph [42].

*2) Implementation:* We have implemented the sketches of CM, CU and C in C++. We apply the SA Counter technique to these sketches, and the results are denoted as SAC CM, SAC CU, and SAC C. For comparison purpose, we also implemented Counter-Tree in C++, the result is denoted as CT in our experiments. The hash functions used in the sketches are implemented using the 32-bit or 64-bit Bob Hash [43] with different initial seeds. For the CM and CU sketches, we set the number of arrays to 3 and use 3 32-bit Bob Hashes. For the C sketch, we set the number of arrays to 3 and use 6 32-bit Bob Hashes. We set the counters to 32 or 16 bits in the classical sketches. In the sketches using SA Counter, the size of the counters is reduced to 16 or 8 bits. Furthermore, for each experiment on the datasets, we run 10 sub-experiments. The average value of the result is recorded as well as the mean square error. Both of them are plotted on the figures.

### B. Metrics

**Average Absolute Error (AAE)**: AAE is defined as $\frac{1}{|\phi|}\sum_{i\in[n]}|f_i - \hat{f}_i|$, where $f_i$ is the frequency of token $i$ that appears in the stream, $\hat{f}_i$ is the estimated frequency and $|\Phi|$ is the volume of the set.

**Average Relative Error (ARE)**: ARE is defined as $\frac{1}{|\phi|}\sum_{i\in[n]}\frac{|f_i-\hat{f}_i|}{f_i}$, where $f_i$ is the frequency of token $i$ that appears in the stream, $\hat{f}_i$ is the estimated frequency and $|\Phi|$ is the volume of the set.

**Per-Flow Memory consumption**: This quantity is defined as the overall memory size of the sketch divided by the number of different flows in a data stream.

**Per-Packet Memory consumption**: Per-Packet Memory consumption is defined as the sketch memory size divided by the number of packets in a data stream.

**Throughput**: Maximum number of insertions that can be processed per second. We use Mega-operations per second(Mops) [44]–[46] as the unit of throughput. All the experiments about speed are repeated 100 times to ensure statistical significance.

### C. Effects of SA Counter technique

*1) Flow volume measurements on CAIDA dataset:* Since the Counter-Tree does not support flow volume measurements, we only compare our technique with the original sketch in this set of experiments.

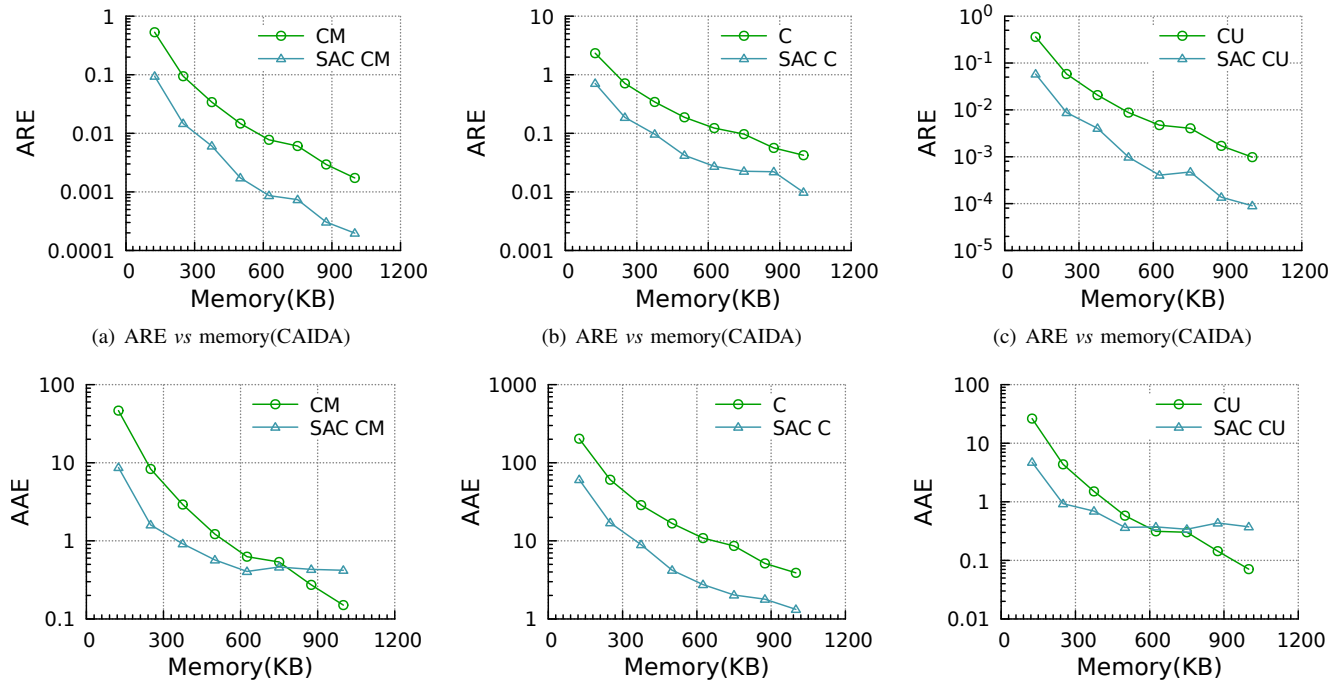
**Effect of SA Counter on CM sketch's ARE and AAE (Figures 3(a) and 3(d)):** The range of the memory size in this experiment is from 125KB to 1000KB. Here, the original CM sketch is compared with the CM sketch using SA Counter using the Dynamic Sign Bits version. We plot how the ARE and AAE (in log scale) change as a function of memory size. Our results show that when the memory is 1000KB, the original CM sketch has 8.7 times higher ARE than the CM sketch using SA Counter. When the memory is below 600KB, the CM sketch using SA Counter have both lower ARE and AAE.

**Effect of SA Counter on C sketch's ARE and AAE (Figures 3(b) and 3(e)):** Our experimental results show that the original C sketch has 4.27 times higher ARE and 3 times higher AAE than C sketch using SA Counter, when the memory is 1000KB. The SA Counter technique successfully reduces the ARE and AAE in the case of the C sketch. Comparing these results with those from the CM sketch, we can see that the CM sketch gives a better estimation of flow size than the C sketch for the same memory size.

**Effect of SA Counter on CU sketch's ARE and AAE (Figures 3(c) and 3(f)):** Our experimental results show that the original CU sketch has 10.9 times higher ARE than CU sketch using SA Counter when the memory is 1000KB. The SA Counter technique successfully reduces the ARE and AAE under small memory consumption in the case of the CU sketch. Comparing this result with the results of the CM and C sketches, we see that the CU sketch using SA Counter has the best performance of all.

**Conclusion**: In this set of experiments, we tested our SA Counter technique for flow volume measurements. We found that the CU sketch using SA Counter has the best performance of all sketches, hence we suggest its use for flow size measurements.

*2) Flow size measurements on synthetic datasets:* In this experiment, we generate datasets following the Zipf distribution ($p(x) = \frac{x^{-a}}{\zeta(a)}$) with different flow sizes and varying $a$ (ranging from 0 to 2.1). Here the task consists in measuring the flow size of each flow in the dataset [47]–[49]. We compare




(a) ARE *vs* memory(CAIDA) (b) ARE *vs* memory(CAIDA) (c) ARE *vs* memory(CAIDA)

(d) AAE *vs* memory(CAIDA) (e) AAE *vs* memory(CAIDA) (f) AAE *vs* memory(CAIDA)

Fig. 3. ARE and AAE as a function of memory consumption for different sketches (CM, C and CU) with and without SA Counter on a Real World Dataset.

the ARE and AAE of the original sketches, of sketches using SA Counter, and of Counter-Tree.

**Effect of SA Counter on CM sketch's ARE and AAE (Figures 4(a) and 4(d)):** We find that when the memory is 1000KB, the original CM sketch has 2.5 times higher ARE and AAE than the CM sketch using SA Counter, while the Counter-Tree has an ARE 5.3 times higher than the CM sketch using SA Counter. As the memory consumption decreases, the ARE and AAE of Counter-Tree gradually go down compared to the one of the CM sketch. The ARE and AAE of the CM sketch using SA Counter are always the lowest.

**Effect of SA Counter on C sketch's ARE and AAE (Figures 4(b) and 4(e)):** We observe that when the memory is 1000KB, the original C sketch has 1.5 times higher ARE and AAE than the C sketch using SA Counter, while the Counter-Tree has an ARE 13.6 times higher than the C sketch using SA Counter. The ARE and AAE of the C sketch using SA Counter are the lowest under all memory sizes. Note that compared to Counter-Tree, the C sketch using SA Counter improves accuracy by one order of magnitude.

**Effect of SA Counter on CU sketch's ARE and AAE (Figure 4(c) and 4(f)):** We find that when the memory is 1000KB, the original CU sketch has 2.7 times higher ARE and AAE than the CU sketch using SA Counter while Counter-Tree has an ARE 10.6 times higher than the CU sketch using SA Counter. As the memory consumption decreases, the ARE and AAE of Counter-Tree gradually become close to the one of the CU sketch. However, the ARE and AAE of the CM sketch using SA Counter are always lowest.

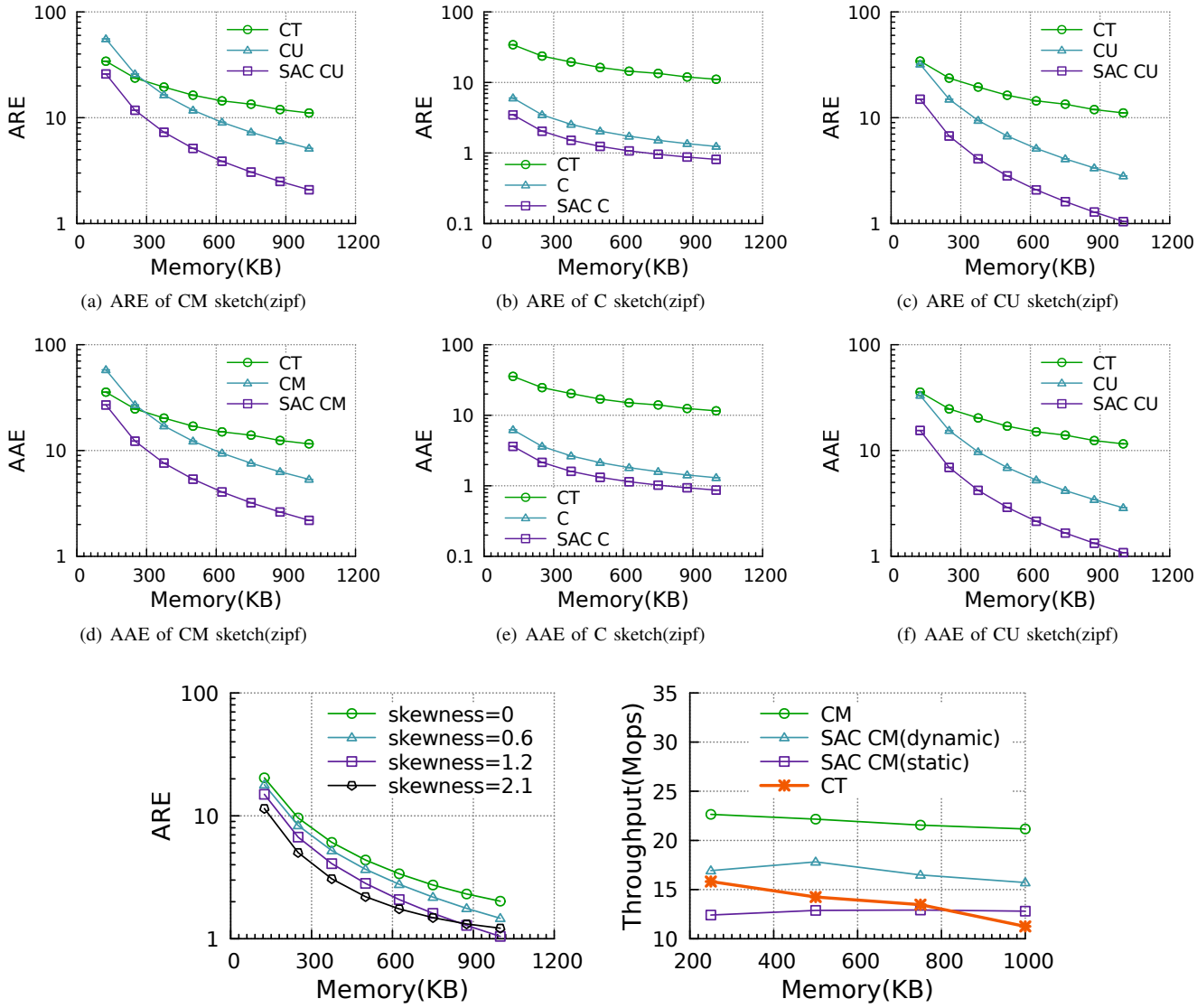
**CU sketch using SA Counter's ARE on synthetic datasets with different skewness (Figure 4(g)):** We find that the ARE decreases with the increase of skewness. The ARE for a skewness of 0 is 9.4 times higher than the ARE when skewness is 2.1. This means the CU sketch using SA Counter is also accurate with very skewed datasets.

**Throughput of CM sketch using SA Counter (Figure 4(h)):** Our experimental results show that when memory consumption is 1000KB, the throughput of the CM sketch using SA Counter is 1.4 times higher than that of Counter-Tree. The throughput of the original CM sketch is the highest.

**Conclusion:** In this set of experiments, we tested our SA Counter technique for flow size measurements. We found that the C sketch using SA Counter has the best performance of all sketches, hence we suggest its use for flow size measurements.

**Effect of per-packet memory size on C sketch's ARE and AAE with fixed $a$ and per-flow memory consumption (Figures 5(a) and 5(c)):** We find that for the C sketch, the change of per-packet memory affects ARE in a limited way compared to per-flow memory consumption. The AAE of both the C sketch and the C sketch using SA Counter drops with the increase of per-flow memory consumption. The original C sketch has 2 to 3 times larger AAE than SA Counter sketches.

**Effect of memory size on CM sketch's ARE and AAE with fixed $a$ and per-flow memory consumption (Figures 5(b) and 5(d)):** We find that the original CM sketch has larger ARE and AAE values compared to the CM sketch using SA Counter. The AAE of both the CM sketch and the CM sketch using SA Counter drop with the increase of per-flow memory




(a) ARE of CM sketch(zipf) (b) ARE of C sketch(zipf) (c) ARE of CU sketch(zipf)

(d) AAE of CM sketch(zipf) (e) AAE of C sketch(zipf) (f) AAE of CU sketch(zipf)

(g) ARE *vs* memory(zipf datasets with different skewness) (h) Throughput *vs* memory

Fig. 4. Experiments on Synthetic Datasets.

consumption. The original C sketch has more than 10 times larger AAE than SA Counter sketches.

**Conclusion:** We found that the SA Counter technique effectively adapted to different counting ranges. For the C sketch and the CM sketch, the change of per-packet memory consumption affects ARE in a limited way compared to the change of per-flow memory. However, AAE drops with the increase of per-packet memory size. Under the same per-packet memory size, the ARE increases at least 10% when per-flow memory changes from $48 \times 10^{-4}$B to $24 \times 10^{-4}$B or $24 \times 10^{-4}$B to $16 \times 10^{-4}$B. Indeed, it shows that in a data stream with many distinct packets, the SA Counter technique can improve the accuracy under small per-flow memory size.

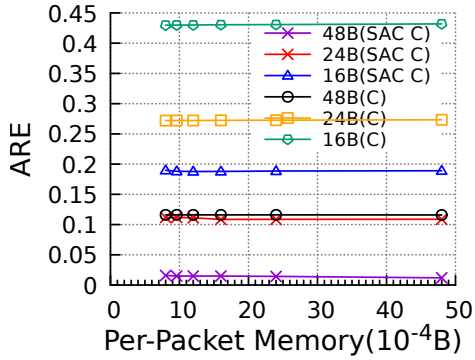
**Effect of the different versions of the CM sketch using SA Counter on ARE and AAE (Figures 6(a) and 6(b)):** In this experiment, we set the memory size of the sketches to be constant and vary the per-flow memory consumption. We find that the ARE and AAE of the Static Sign Bits version of the CM sketch with the length of sign bits $s = 3$ is 2.43 times higher than the one of the Dynamic version on average.

**Conclusion:** This set of experiments showed that the Dynamic Sign Bits version has better performance than the Static one under different per-flow memory consumption.

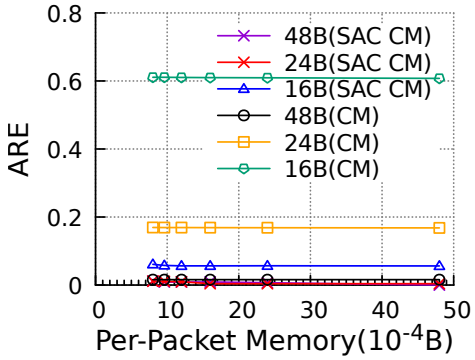
We find that the ARE and AAE of original sketches is more influenced by the decrease of per-flow memory consumption than the one of SA Counter sketches. When the per-flow memory consumption drops to 1.3B, the ratio between the ARE of the original sketch and of the SA Counter sketch is 10.83 for the CM sketch.


## ACKNOWLEDGEMENT

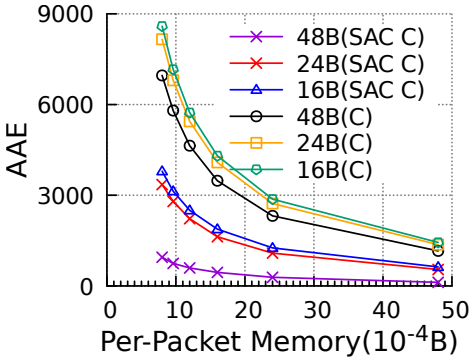
We would like to thank the anonymous reviewers for their thoughtful suggestions. This work is partially supported by Primary Research & Development Plan of China (2018YFB1004403, 2016YFB1000304), and NSFC (61672061).



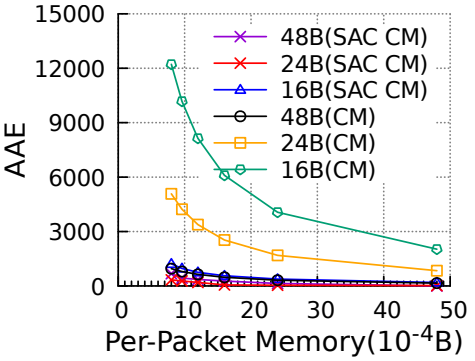

(a) ARE *vs.* per-packet memory (C).




(b) ARE *vs.* per-packet memory (CM).

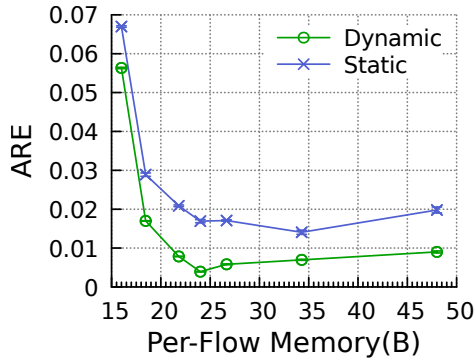



(c) AAE *vs.* per-packet memory (C).

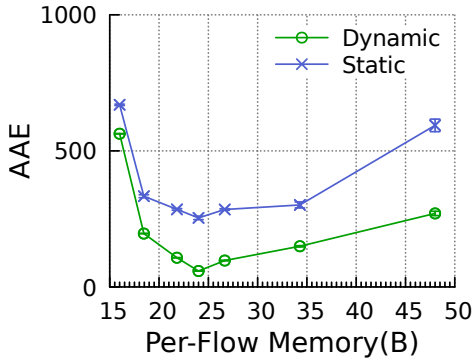



(d) AAE *vs.* per-packet memory (CM).

Fig. 5. Effect of per-packet memory on ARE and AAE using Synthetic Datasets.




(a) Two versions of the CM sketch using SAC (ARE).




(b) Two versions of the CM sketch using SAC (AAE).

Fig. 6. Comparison between the two versions of SAC on Synthetic Datasets.

## VI. Conclusion

Thanks to their memory efficiency and fast and constant speed, sketches have attracted much attention for network measurements. If the flow size distribution is even, there is little room for improvements compared to previous work. However, when flow size distribution is highly skewed, existing sketches are very inefficient in memory usage. No existing work is capable of achieving memory efficiency without hurting their constant and fast speed, which is really important in high-speed network traffic. To address this issue, we propose a generic technique, namely *self-adaptive counters (SA Counter)* in two versions, static and dynamic. Our key idea is simple: When a counter is going to overflow, we do not increase it one by one, but increase it with predefined probabilities. When the counter is small, it just works like a normal counter. SA Counter makes a small counter able to represent both small and large values. The error incurred by the probabilistic increase is theoretically and experimentally proved to be negligible compared to the size of elephant flows. We apply SA Counter to three typical sketches: sketches of CM, C, and CU. Experimental results show that, compared with the state-of-the-art, sketches using SA Counter improve the accuracy by up to 13.6 times.

## References

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

[2] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *Network Protocols (ICNP), 2015 IEEE 23rd International Conference on*. IEEE, 2015, pp. 1–10.

[3] Y. Zhou, Y. Zhou, S. Chen, and O. P. Kreidl, "Limiting self-propagating malware based on connection failure behavior," in *Proc. of Seventh International Conference on Network and Communications Security (NCS)*, 2015.

[4] H. Xu, Z. Yu, C. Qian, X.-Y. Li, Z. Liu, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in sdns," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3587–3601, 2017.

[5] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 101–114.

[6] R. Ben Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 127–140.

[7] 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. 5–5, 2008.

[8] R. Schweller, A. Gupta, E. Parsons, and Y. Chen, "Reversible sketches for efficient and accurate change detection over network data streams," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 207–212.

[9] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, G. Zhang, J. Cao, D. Zhang, K. Xie, X. Wang *et al.*, "Accurate recovery of internet traffic data: A sequential tensor completion approach," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 2, pp. 793–806, 2018.

[10] K. Xie, C. Peng, X. Wang, G. Xie, J. Wen, J. Cao, D. Zhang, and Z. Qin, "Accurate recovery of internet traffic data under variable rate measurements," *IEEE/ACM Transactions on Networking*, 2018.

[11] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 129–143.

[12] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers." in *NSDI*, 2016, pp. 311–324.

[13] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 113–126.

[14] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 1449–1463.

[15] "FPGA data sheet [on line]," Available: http://www.xilinx.com.

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

[17] C. Estan and G. Varghese, *New directions in traffic measurement and accounting*. ACM, 2002, vol. 32, no. 4.

[18] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.

[19] K. Cheng, L. Xiang, and M. Iwaihara, "Time-decaying bloom filters for data streams with skewed distributions," in *Research Issues in Data Engineering: Stream Data Mining and Applications, 2005. RIDE-SDMA 2005. 15th International Workshop on*. IEEE, 2005, pp. 63–69.

[20] I. N. Bozkurt, Y. Zhou, T. Benson, B. Anwer, D. Levin, N. Feamster, A. Akella, B. Chandrasekaran, C. Huang, B. Maggs *et al.*, "Dynamic prioritization of traffic in home networks," 2015.

[21] G. Cormode, "Sketch techniques for approximate query processing," *Foundations and Trends in Databases. NOW publishers*, 2011.

[22] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 267–278.

[23] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management." in *NSDI*, 2009, pp. 335–348.

[24] G. Cormode, B. Krishnamurthy, and W. Willinger, "A manifesto for modeling and measurement in social media," *First Monday*, vol. 15, no. 9, 2010.

[25] P. B. Gibbons and Y. Matias, "Synopsis data structures for massive data sets," *External memory algorithms*, vol. 50, pp. 39–70, 1999.

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

[27] M. Chen and S. Chen, "Counter tree: A scalable counter architecture for per-flow traffic measurement," in *Network Protocols (ICNP), 2015 IEEE 23rd International Conference on*. IEEE, 2015, pp. 111–122.

[28] C. Hu, B. Liu, H. Zhao, K. Chen, Y. Chen, Y. Cheng, and H. Wu, "Discount counting for fast flow statistics on flow size and flow volume," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 970–981, 2014.

[29] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 5, pp. 1622–1634, 2012.

[30] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 20–29.

[31] N. Hua, A. Lall, B. Li, and J. Xu, "A simpler and better design of error estimating coding," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 235–243.

[32] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends in Databases*, vol. 4, no. 1–3, pp. 1–294, 2012.

[33] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *ACM SIGCOMM 2018*, pp. 561–575.

[34] H. Dai, Y. Zhong, A. X. Liu, W. Wang, and M. Li, "Noisy bloom filters for multi-set membership testing," in *ACM SIGMETRICS*, 2016, pp. 139–151.

[35] H. Dai, L. Meng, and A. X. Liu, "Finding persistent items in distributed, datasets," in *IEEE INFOCOM*, 2018.

[36] 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.

[37] Y. Yu, D. Belazzougui, C. Qian, and Q. Zhang, "Memory-efficient and ultra-fast network lookup and forwarding using othello hashing," *IEEE/ACM Transactions on Networking*, 2018.

[38] Z. Yu, Z. Ge, A. Lall, J. Wang, J. Xu, and H. Yan, "Crossroads: A practical data sketching solution for mining intersection of streams," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 223–234.

[39] J. Liu, P. Zhang, H. Wang, and C. Hu, "Countermap: Towards generic traffic statistics collection and query in software defined network," in *Quality of Service (IWQoS), 2017 IEEE/ACM 25th International Symposium on*. IEEE, 2017, pp. 1–5.

[40] "The caida anonymized 2016 internet traces." http://www.caida.org/data/overview/.

[41] D. M. Powers, "Applications and explanations of zipf's law," in *Proceedings of the joint conferences on new methods in language processing and computational natural language learning*. Association for Computational Linguistics, 1998, pp. 151–160.

[42] A. Rousskov and D. Wessels, "High performance benchmarking with web polygraph," *Software-Practice and Experience*, vol. 1, pp. 1–10, 2003.

[43] "Hash website," http://burtleburtle.net/bob/hash/evahash.html.

[44] Z. Li, B. Chang, S. Wang, A. Liu, F. Zeng, and G. Luo, "Dynamic compressive wide-band spectrum sensing based on channel energy reconstruction in cognitive internet of things," *IEEE Transactions on Industrial Informatics*, 2018.

[45] Z. Li, F. Xiao, S. Wang, T. Pei, and J. Li, "Achievable rate maximization for cognitive hybrid satellite-terrestrial networks with af-relays," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 2, pp. 304–313, 2018.

[46] Z. Li, Y. Liu, A. Liu, S. Wang, and H. Liu, "Minimizing convergecast time and energy consumption in green internet of things," *IEEE Transactions on Emerging Topics in Computing*, 2018.

[47] F. Xiao, Z. Wang, N. Ye, R. Wang, and X.-Y. Li, "One more tag enables fine-grained rfid localization and tracking," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 1, pp. 161–174, 2018.

[48] F. Xiao, L. Chen, C. Sha, L. Sun, R. Wang, A. X. Liu, and F. Ahmed, "Noise tolerant localization for sensor networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1701–1714, 2018.

[49] H. Zhu, F. Xiao, L. Sun, R. Wang, and P. Yang, "R-ttwd: Robust device-free through-the-wall detection of moving human with wifi," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1090–1103, 2017.