This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



TABLE I
SYMBOLS USED IN THE PAPER

| Symbol | Description |
|---|---|
| $l$ | the value resided in an SEAD counter |
| $d$ | number of counter arrays in a sketch |
| $w$ | number of counters in a counter array |
| $n$ | number of bits in a counter |
| $s$ | length of *sign bits* in a counter |
| $\gamma[0], \ldots, \gamma[k-1]$ | the expansion array for SEAD Counter. |
| stage[i] | the starting value of stage i, depending on the version, can be derived from $\gamma[0], \ldots, \gamma[k-1]$. |

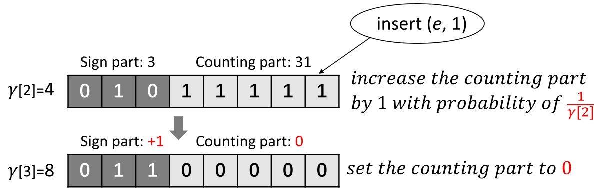



Fig. 1. Example of SEAD Counter with Static Sign Bits, where $n = 8$ and $s = 3$. To insert a pair $(e, 1)$, the counting part is increased by 1 with probability $\frac{1}{4}$. If the counting part overflows, we increase the sign part by 1, and get a new expansion parameter $\gamma[3] = 8$.

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 the value related 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 (SEAD Counter):** Our technique is called Self-Adaptive Counters (SEAD Counter). Next we show the data structure and operations of the SEAD Counter.

**Data Structure:** For the SEAD 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 of the SEAD Counter is $C_{static}(n, s) = \sum_{i=0}^{2^s-1}(\gamma[i] \times 2^{n-s})$. Here, we define the capacity of the SEAD counter as the *expected* number of increments it can take before exceeding its maximal value. Due to over-sampling problem, we should set the expansion array to make the capacity slightly higher than the number of increments we want to support. Besides, in the extreme case where over-sampling happened, we just return the capacity value, which is the maximum possible value that can be represented by our counter.

**Insertion:** We show the steps of how to add 1 to an SEAD Counter. The procedure to update SEAD Counters with larger values can be seen in **Algorithm 1**.

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

**Algorithm 1** *read* and *update* Functions of SEAD Counters

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

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

*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:** For an SEAD 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 (We assume $\gamma[j] = m^j$ in the stage expression to contrast static and dynamic version better):

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

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

$$\begin{aligned} value(\mathcal{C}) &= c_0 \times \gamma[s_0] + stage[s_0]. \\ &= c_0 m^{s_0} + \frac{2^{n-s}(m^{s_0} - 1)}{m - 1} \end{aligned} \quad (2)$$

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



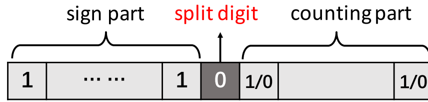



Fig. 2. Structure of Dynamic Sign Bits SEAD Counter.

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 an 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 SEAD 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 SEAD Counter is $C_{dynamic}(n) \sum_{i=0}^{n-2}(\gamma[i] \times 2^{n-i-1})$.

**Insertion:** The insertion process of the dynamic version of the SEAD 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 the SEAD Counter is similar to the static version. To calculate the value of an SEAD 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 through the following formula:

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

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

Even if there can be over-sampling, the $n$-bit SEAD Counter with Dynamic Sign Bits can increment $2^n$ times, which is the same as the normal counter.

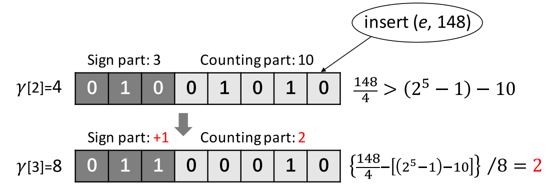



Fig. 3. An example of SEAD Counter with Static Bits Version, where $n = 8$ and $s = 3$. To increase the counter by 148, the counting part will overflow. We increase the sign part by 1, and get a new expansion parameter $\gamma[3] = 8$.

### C. Insertion to the Counters With Larger Values

We show the steps of how to add $v$ to an SEAD Counter.

*Step 1:* In an SEAD counter, we get the sign part $s_0$, the counting part $c_0$, and we check the value $\gamma[s_0]$ in the expansion array. Since $\gamma[s_0]$ indicates how many times the counting part should be expanded, we first calculate $\frac{v}{\gamma[s_0]}$, getting a quotient $q$ and a remainder $r$.

*Step 2:* We compare $q$ with $g = (2^{n-s} - c_0)$, and there are two cases:

1) $q < g$: This means the SEAD Counter can hold the value without changing the sign part. We only increase the counting part by $q$, *i.e.*, $c_0 + q \mapsto c_0$, $r \mapsto r$ and $s_0 \mapsto s_0$.
2) $q \geqslant g$: We first increase the sign part by 1, *i.e.*, $s_0+1 \mapsto s_0$, and set the counting part to zero, *i.e.*, $0 \mapsto c_0$. Then, we calculate $\frac{(q-g)\times\gamma[s_0]+r}{\gamma[s_0']}$, and get a new quotient $q'$ and a new remainder $r'$. We let $r' \mapsto r$, $q' \mapsto q$ and go back to Step 1.

*Step 3:* We further increase the counting part by 1 with probability $\frac{r}{\gamma[s_0]}$. If the counting part is increased to $2^{n-s}$, we increase the sign part by 1 and set the counting part to zero.

## IV. Case Studies

To further illustrate the generality of the SEAD Counter technique, in this section, we show how to apply the SEAD Counter to the sketches of CM [18], CU [19] and C [20]. We further extend our counters to CBF and VI-CBF. Two functions of the SEAD Counters, $read$ and $update$, are shown in Algorithm 1.

### A. Application to the CM Sketch

In the CM sketch, all normal counters are replaced by the SEAD Counters. After locating $d$ counters using $d$ hash functions, the insertion procedure is done by calling the "$update$" function of the corresponding SEAD Counter. The query function is done by reporting the minimum value of "$read$" by all corresponding SEAD counters.

### B. Application 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 the SEAD Counter. The query process is the same as for the CM sketch.

### C. Application 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

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



of hash functions $h[1]\cdots h[t]$ and $g[1]\cdots g[t]$, where $h[i]:[0,n]\rightarrow[0,k]$ , $g[i]:[0,n]\rightarrow\{-1,1\}$. 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 the SEAD Counter to add the value to the corresponding counter. The query function is done by reporting the median value of "$read$" 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 SEAD 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. The "$update$" function takes a counter $c$, an increment value $c$, and the expansion array $\gamma$ as input.

### D. Application to CBF and VI-CBF

CBF has essentially the same structure as the CM sketch, so the "$update$" and the "$read$" functions behave the same. However, they are used differently and therefore sized differently.

VI-CBF [52] works in the same model as CBF, which means that the same element may be inserted more than once. VI-CBF differs from CBF in that the increments and decrements are variable. While all the corresponding hashed counters are incremented by one in CBF when inserting an element, in the case of VI-CBF, all the corresponding hashed counters are incremented by a certain hashed value, which is previously defined as a set of possible variable increments $D$. For the same element, the corresponding hashed counters are incremented with the same corresponding hashed value. For CBF, we check in each of its counters if its hashed value is positive. For VI-CBF, we check in each of its counters if its hashed value in $D$ could be part of the sum. We adopt the scheme proposed in [52], which uses variable increments but only relies on a single variable-increment counter per entry, without the additional counter that indicates the number of hashed elements. Specifically, in each array entry, the counter is updated using variable increments selected from a set $D=\{v_1,v_2,\ldots,v_\ell\}$. We use again two sets of $k$ hash functions, $H=\{h_1,\ldots,h_k\}$ and $G=\{g_1,\ldots,g_k\}$. Upon insertion, at each corresponding array position $h_i(x)$, the counter is incremented by the element $v_{g_i(x)}$ of the set $D$. Likewise, upon deletion, the counter $h_i(x)$ is decremented by $v_{g_i(x)}\in D$.

When an element is inserted, we use the "$update$" function in Algorithm 1 and increment the counter with the corresponding value. When we want to query whether the element $y$ belongs to the set, we see whether the values of the $k$ hashed entry of $y$ can be the sum of the increment. Specifically, let $y$ be an element whose $i$-th hash function $h_i(y)$ hashes into an entry of value $c_i$. If there exists $i$ such that $\left(c_i-v_{g_i(y)}\right)\in(-\infty,-1]\cup[1,L-1]$, then $y\notin S$. If there is no such $i$, we say that $y\in S$. Here the value $c_i$ is $c$ in Algorithm 1, which is derived by the "$read$" function.

## V. Theoretical Analysis

In this section, we analyze the improvements that using SEAD Counter for a given amount of memory. We also compare the performance of the Static Sign Bits and the Dynamic Sign Bits version. The result shows that the Dynamic Sign Bits version of SEAD Counter solves the problem of errors introduced at an early stage with the Static Sign Bits version SEAD Counter.

### A. More Capacities

Suppose a counter has $n$ bits. Let's calculate the capacities of a regular and a self-adaptive counter. The capacity of a regular counter is $2^n$. In the Static Sign Bits version of the SEAD Counter, the capacity is raised to $\sum_{i=0}^{2^s-1}(\gamma[i]\times 2^{n-s})$, where $s$ is the length of the sign part. In the Dynamic Sign Bits version, the capacity is $\sum_{i=0}^{n-2}(\gamma[i]\times 2^{n-i-1})$. For simplicity, we choose $\gamma[i]=m^i$ as an example: with $k$ bits, for the Static Sign Bits version to reach a capacity of $2^n$, we have:

$$2^n=2^{k-s}\times(\frac{m^{2^s}-1}{m-1})$$

Since $k$ must be an integer, we can solve:

$$k=n+s-\lfloor log_2(\frac{m^{2^s}-1}{m-1})\rfloor \tag{4}$$

For example, if we set the length of the sign part to $s=2$, and the expansion array to $\gamma[i]=4^i$, the resulting space will be reduced to $k=n-4$. That means that using the static version of SEAD Counter technique, four bits can be saved to reach the same capacity as a regular counter.

If we set the expansion array to $\gamma[i]=2^i$ in the Dynamic Sign Bits Version, we have:

$$2^n=2^{k-1}\times(k-1)$$

For example, if we choose $n=16$, then we have $k=13$ and 3 bits can be saved.

For the Dynamic Sign Bits Version to reach a capacity of $2^n$ with $m\neq 2$, we have:

$$2^n=2^{k-1}\times\frac{(\frac{m}{2})^{k-1}-1}{\frac{m}{2}-1},m\neq 2$$

If we set the expansion array to $\gamma[i]=4^i$, we have:

$$k=\lceil\frac{n}{2}\rceil+1 \tag{5}$$

With the Dynamic Sign Bits Version, almost half of the space can be saved to reach the same capacity as for a regular counter.

### B. Upper-Bound of Relative Error

To evaluate the accuracy of the SEAD Counter, we prove the unbiasness of SEAD Counter and give the analysis of its variance and relative error.

*1) unbiasness:* In each $stage[s]$, the probability of incoming packet added to the SEAD Counter remains the same: $p=\frac{1}{\gamma[s]}$. Suppose there are $t$ packet arrivals in this stage, we define $X$ as the increments of the actual value stored in the counter, and we define $V(X)$ as the increments of the represented value of $X$. $X$ follows a Binomial distribution: $X\sim B(t,\frac{1}{\gamma[s]})$. Deriving the expectation of $V(X)$ is not difficult:

$$\begin{aligned}E(V(X))&=E(\gamma[s]\times X)\\&=\gamma[s]\times E(X)\\&=\gamma[s]\times\frac{t}{\gamma[s]}=t\end{aligned}$$

This means that increments are unbiased in each stage, and the counter estimation is thus unbiased.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



2) variance and relative error: we discuss the quality of estimation mainly in terms of the root mean squared relative error (RMSRE), or relative error in short. For each $stage[s]$, their increments and decrements processes can be considered independent. Then using the same notation in 1), for $stage[s]$,

$$\begin{aligned} var(V(X)) &= E(V(X)^2) - E^2(V(X)) \\ &= E(\gamma^2[s] \times X^2) - t^2 \\ &= \gamma^2[s]E(X^2) - t^2 \\ &= \gamma^2[s](Var(X) + E^2(X)) - t^2 \\ &= \gamma^2[s](\frac{t(\gamma[s]-1)}{\gamma^2[s]} + \frac{t^2}{\gamma^2[s]}) - t^2 \\ &= t(\gamma[s]-1) \end{aligned}$$

We define $\hat{Y}$ as the random variable representing the estimation value of a flow after $Y$ packets have arrived. The root mean square relative error (RMSRE) is

$$RMSRE[Y] = \sqrt{\mathbb{E}\left[\left(\frac{\hat{Y}-Y}{Y}\right)^2\right]}$$

We give a similar proof as in [41]. Let $Q_l(Y)$ be the probability to have a value $l$ resided in the counter given that exactly $Y$ packets have arrived at the flow. As mentioned before, the estimator is unbiased, thus

$$\sum_l Q_l(Y)V(l) = Y. \tag{6}$$

In order to calculate the relative error, we should first find the variance of the estimation value. The Static Sign Bits version can accurately record $[0, 2^{(n-s-1)}]$, while the Dynamic Sign Bits version can accurately record $[0, 2^{n-1}]$, where $n$ is the number of bits in a counter and $s$ is the length of the sign part in the static version. Therefore, when $Y \leq 2^{(n-s-1)}$ for Static Sign Bits version and $Y \leq 2^{(n-1)}$ for Dynamic Sign Bits version, the relative error is also 0. For greater values, we already know its mean, so let us find

$$\mathbb{E}\left[\hat{Y}^2\right] = \sum_l Q_l(Y)V^2(l).$$

We first compute $\mathbb{E}\left[(Y \hat{+} 1)^2 - \hat{Y}^2\right]$. If the counter's value is $l$ and it's at stage $s_l$, when a packet arrives the counter's value is incremented with probability $\frac{1}{\gamma[s_l]}$ or remains unchanged with probability $1 - \frac{1}{\gamma[s_l]}$. Therefore,

$$\begin{aligned} \mathbb{E}\left[(Y \hat{+} 1)^2 - \hat{Y}^2\right] &= \sum_l \left(V(l+1)^2 - V(l)^2\right) \cdot \frac{1}{\gamma[s_l]} Q_l(Y) \\ &= \sum_l \left(V(l+1) + V(l)\right) Q_l(Y) \\ &= \sum_l \left(2V(l) + \gamma[s_l]\right) Q_l(Y) \\ &= \sum_l 2V(l)Q_l(Y) + \gamma[s_l]Q_l(Y) \\ &= 2Y + \sum_l \gamma[s_l]Q_l(Y) \end{aligned} \tag{7}$$

Therefore,

$$\begin{aligned} \mathbb{E}\left[\hat{Y}^2\right] &= \sum_{i=0}^{Y-1} \left(\mathbb{E}\left[(\widehat{i+1})^2\right] - \mathbb{E}\left[(\hat{i})^2\right]\right) + \mathbb{E}\left[\hat{0}^2\right] \\ &= \sum_{i=0}^{Y-1} (2i + \sum_l \gamma[s_l]Q_l(i)) \\ &= Y(Y-1) + \sum_{i=0}^{Y-1} \sum_l \gamma[s_l]Q_l(i) \end{aligned} \tag{8}$$

We define the expected counter's value of $i$ as $l_i$. As the stage value remain constant in a relative long interval, we can assume $l_i$'s stage is approximately no larger than $s_{l_i} + 1$. Suppose $a_k = \gamma[s_{l_i} + 2 - k]$, $b_k = Q_l(i)$, $k = 2, \cdots, s_{l_i} + 2, a_1 = \gamma[s_{l_i} + 1], b_1 = \sum_{l=l_i}^{i} Q_l(i)$, we can know $a_k$ is nonincreasing and nonnegative, $\sum_{l=1}^{k} Q_l(i) \leq 1$. Then using Abel's inequality,

$$\begin{aligned} \mathbb{E}\left[\hat{Y}^2\right] &= Y(Y-1) + \sum_{i=0}^{Y-1} \sum_l \gamma[s_l]Q_l(i) \\ &\lessapprox Y(Y-1) + \sum_{i=0}^{Y-1} \gamma[s_{l_i} + 1] \times 1 \\ &\leq Y(Y-1) + Y(\gamma[s_{l_Y} + 1]) \end{aligned} \tag{9}$$

Hence, the variance is

$$\mathbb{V}\left[\hat{Y}\right] = \mathbb{E}\left[\hat{Y}^2\right] - \mathbb{E}\left[\hat{Y}\right]^2 \leq Y(\gamma[s_{l_Y} + 1] - 1) \tag{10}$$

and the relative error is

$$RMSRE[Y] = \sqrt{\frac{\mathbb{V}\left[\hat{Y}\right]}{Y^2}} \leq \sqrt{\frac{(\gamma[s_{l_Y} + 1] - 1)}{Y}} \tag{11}$$

As $\gamma[s_{l_Y}]$ is relevant with $Y$, to show $RMSRE[Y]$ is bounded, we choose $\gamma[i] = m^i, m = 2$ under Dynamic Sign Bits Version as an example to simplify the analysis. Suppose $s_{l_Y} = i$, from (3),

$$\begin{aligned} RMSRE[Y] &\leq \sqrt{\frac{(\gamma[s_{l_Y} + 1] - 1)}{Y}} \\ &\leq \sqrt{\frac{2^{i+1} - 1}{i \times 2^{n-1}}} \leq \sqrt{\frac{1}{i \times 2^{n-i-2}}} \end{aligned} \tag{12}$$

As *RHS* of (12) is nondecreasing, we know when $Y$ gets bigger, its $RMSRE$ value becomes larger. However, it's still bounded by $\sqrt{\frac{1}{n-2}}$ (when $i = n-2$). When $m \neq 2$, the upper bound is similar:

$$\begin{aligned} RMSRE[Y] &\leq \sqrt{\frac{(\gamma[s_{l_Y} + 1] - 1)}{Y}} \\ &\leq \sqrt{\frac{m^{i+1} - 1}{2^{n-1} \times \frac{(\frac{m}{2})^{i+1} - 1}{\frac{m}{2} - 1}}} \leq \sqrt{\frac{m}{2^{n-i}}} \end{aligned} \tag{13}$$

### *C. Advantage of Dynamic Sign Bits Version*

In real applications, the first entry of the expansion array $\gamma[0]$ is always set to 1, which means that the SEAD Counter will accurately record values when the sign part is 0. Since there is enough memory to record accurate values when they are small, we shouldn't do worse than normal counters when counting small values. If we use $\gamma[0]$ greater than 1,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



the counter values are inaccurate even for small values, which means that the result is worse than that of normal counters and this is not acceptable. The Static Sign Bits version can accurately record $[0, 2^{(n-s-1)}]$, while the Dynamic Sign Bits version can accurately record $[0, 2^{n-1}]$, where $n$ is the number of bits in a counter and $s$ is the length of the sign part in the static version.

If we want these two versions to have the same number of stages, then $2^s \approx n-1$. Typically, for a counter of 8 bits, $n = 8$ and $s = log_2(8-1) \approx 3$. Then, $stage[1] = 16$ for the Static Sign Bits version, and $stage[1] = 64$ for the Dynamic Sign Bits version. A larger $stage[1]$ enables the Dynamic Sign Bits version to have a larger exact counting range. As a result, it is more accurate when recording the size of mouse flows.

At early stages of the SEAD Counter, when it is more likely to get accurate results, there is more capacity in the Dynamic Sign Bits Version than in the Static Sign Bits version. At later stages of the SEAD Counter on the other hand, when it is more likely to get inaccurate results, there is less capacity in the Dynamic Sign Bits Version than in the Static Sign Bits version. This property in terms of accuracy is a way to explain the advantage of the Dynamic Sign Bits Version of the SEAD Counter.

### *D. Computational Model of SEAD Counter on Sketches*

For a vector $\boldsymbol{a}$ with dimension $n$, we define its current state at time $t$ as $[a_1(t), \cdots, a_i(t), \cdots, a_n(t)]$. Initially, $\boldsymbol{a}$ is set to $\mathbf{0}$ and $a_i(t)$ is 0 for all i. The $t$-th update to the individual entries of the vector is presented as pair $(i_t, c_t)$, which means,

$$\begin{aligned} a_{i_t}(t) &= a_{i_{t-1}}(t-1) + c_t \\ a_{i'_t}(t) &= a_{i'_{t-1}}(t-1) \qquad i'_t \neq i_t \end{aligned} \tag{14}$$

In some cases, $c_t$s are always strictly positive, which means the entries will only increase. we call this model as *cash register* model. In other cases, $c_t$ could be positive or negative. We call it as *turnstile* model. Under the *turnstile* model, if all $a_i(t)$s are non-negative for all time, we call this case as $non-negative$ case; if $a_i(t)$s may be negative, we call this case as $general$ case. Different models correspond to their specific scenarios and therefore figuring out the models sketch can be applied is important. For example, CM sketch works in the non-negative turnstile model while C sketch works in the general turnstile model.

When applying SEAD Counter on sketches, it will introduce its new error to the sketches. Therefore, we should determine when replacing counters with estimators, what kinds of computational model each sketch works on. For estimators, when we delete some elements, we will introduce probabilistic updates. After certain times of deletion, the error may not be proportional to the norm of the size-vector. Therefore, estimators don't satisfy the requirements of turnstile model. As for estimators in the cash register model, their error introduced by insertion is bounded by the norm of the size-vector. Therefore, we conclude estimators work in the cash register model.

For the error bounds of SEAD Counter on CM sketch, we give a similar theorem like Theorem 1 in [18]. To simplify the analysis, we only consider the Dynamic Sign Bits Version here.

*Theorem 1: Let $\|\boldsymbol{a}\|_1$ denote the sum of all entries of $\boldsymbol{a}$ and $s$ denote the stage number such that $stage[s] < a_i + \frac{\|\boldsymbol{a}\|_1}{2e} < stage[s+1]$. Given a small variable $\epsilon$ no less than $min\{\sqrt{\frac{m}{2^{n-s-2}}}, \frac{4e(max_{i=1,\cdots n}a_i)}{\|\boldsymbol{a}\|_1}\}$, the estimate $\hat{a}_i$ of SEAD Counter on CM sketch($w \times d$ counters) has the following guarantees: with probability at least $1-\delta$,*

$$|a_i - \hat{a}_i| \leq \epsilon \|\boldsymbol{a}\|_1 \tag{15}$$

*Proof:* Firstly, as the original CM sketch always overestimates $a_i$, we only need to proof the direction with larger error: $\hat{a}_i - a_i \leq \|\boldsymbol{a}\|_1$. Besides, as SEAD Counter has no error before the first $2^{n-1}$ updates, we can assume $\|\boldsymbol{a}\|_1 > 2^{n-1}$.

We introduce indicator variables $I_{i,j,k}$, which equals to 1 if $(i \neq k) \wedge (h_j(i) == h_j(k))$ and 0 otherwise. As hash functions are pairwise independent, then by setting $w = \frac{4e}{\epsilon}$, we have

$$\mathbb{E}(I_{i,j,k} = Pr[h_j(i) = h_j(k)]) \leq \frac{1}{range(h_j)} = \frac{\epsilon}{4e}$$

Define the random variable $X_{i,j}$ as $\sum_{k=1}^{n} I_{i,j,k}a_k$ and we can see $X_{i,j} > 0$ as all $a_k > 0$. Define $Z_j = count[j, h_j(i)]$ as the value which should be counted in the place $(j, h_j(i))$ of the hash table and we have $Z_j = a_i + X_{i,j}$. Clearly, min $Z_j > a_i$. Then,

$$\mathbb{E}(X_{i,j}) = \mathbb{E}(\sum_{k=1}^{n} I_{i,j,k}a_k) \leq \sum_{k=1}^{n} a_k \mathbb{E}(I_{i,j,k}) \leq \frac{\epsilon}{4e}\|\boldsymbol{a}\|_1$$

Define $\widehat{Z_j} = \widehat{count[j, h_j(i)]}$ as the estimation value after $count[j, h_j(i)]$ has been counted by the SEAD Counter, we have

$$\begin{aligned} &Pr(\hat{a_i} > a_i + \epsilon\|\boldsymbol{a}\|_1) \\ &\quad = Pr(\forall j,\ \widehat{Z_j} > a_i + \epsilon\|\boldsymbol{a}\|_1) \\ &\quad \leq Pr(\forall j,\ \widehat{Z_j} - Z_j + a_i + X_{i,j} > a_i + \epsilon\|\boldsymbol{a}\|_1) \\ &\quad \leq Pr(\forall j,\ \widehat{Z_j} - Z_j > \frac{\epsilon}{2}\|\boldsymbol{a}\|_1 \vee X_{i,j} > \frac{\epsilon}{2}\|\boldsymbol{a}\|_1) \end{aligned} \tag{16}$$

For the former probability in the RHS of (16), by the Chebshev inequality,

$$\begin{aligned} Pr(\widehat{Z_j} - Z_j > \frac{\epsilon}{2}\|\boldsymbol{a}\|_1) &\leq \frac{4\mathbb{V}(\widehat{Z_j})}{\epsilon^2\|\boldsymbol{a}\|_1^2} \leq \frac{4Z_j(\gamma[s]-1)}{\epsilon^2\|\boldsymbol{a}\|_1^2} \\ &\leq \frac{4}{\epsilon^2} \cdot \frac{(a_i + X_{i,j})^2}{\|\boldsymbol{a}\|_1^2} \cdot \frac{\gamma[s]-1}{Z_j} \\ &\lessapprox \frac{4}{\epsilon^2} \cdot (\frac{\epsilon}{4e} + \frac{\epsilon}{4e})^2 \cdot \frac{m}{2^{n-s}} \\ &= \frac{1}{2e} \cdot \frac{m}{e \cdot 2^{n-s-1}} \lessapprox \frac{1}{2e} \end{aligned} \tag{17}$$

For the latter probability in the RHS of (16), by the Markov inequality,

$$Pr(X_{i,j} > \frac{\epsilon}{2}\|\boldsymbol{a}\|_1) < Pr(X_{i,j} > e\mathbb{E}(X_{i,j})) < \frac{1}{2e} \tag{18}$$

Therefore, if we choose $\delta = ln(1/d)$

$$RHS\ of\ (16) \leq (\frac{1}{2e} + \frac{1}{2e})^d = e^{-d} = \delta \tag{19}$$

□

For the error bounds of C sketch, as the variance brought by SEAD technique (inequality (10)) is proportionally to the square of L1 norm, the L2 accuracy guarantee in [20] may be not satisfied. When applying other estimators like ICEBuckets on C sketch, as their variances are also $O(\|\boldsymbol{a}\|_1^2)$, they can only provide L1 accuracy guarantee. Therefore, when using C

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



sketch and requiring L2 accuracy guarantee (typically enough memory space), one should use the original counter instead.

From the proof of SEAD on CM sketch above, some inequalities will not be satisfied when extreme cases happens. When applying counter estimation strategies (SEAD, ICEBuckets and etc.) to the sketches, we should be careful to avoid some extreme cases theoretically. For example, a large number of updates always come to the same entry, which means $a_i = \|\boldsymbol{a}\|_1$. However, in real network datasets, these extreme cases will not happen. We also know from Theorem 1, there is a lower error bound for $\epsilon$ when counting large flows. Therefore, counter estimation strategies should only be used when the memory is limited and they have no advantage in scenarios where there are sufficient memories.

## VI. Experimental Results

We applied our SEAD Counter to the task of flow measurements and set representation, where we use our SEAD Counter in sketches, as estimators, and in Bloom filters to evaluate the performance on real applications utilizing approximate counters. In this section, we start by evaluating the SEAD Counter technique on a real-world dataset, by comparing the Average Relative Error (ARE) and the Average Absolute Error (AAE) across (1) sketches without the SEAD Counter, (2) with the SEAD Counter and (3) Counter-Tree. Then, we generate a synthetic dataset which follows the Zipf distribution. We study how the ARE and AAE values change when we vary parameters (defined in Section VI-B). We also compare with other state-of-the-art methods on two large datasets. Finally, we extend our SEAD Counter to CBF and VI-CBF, and evaluate these on two real-world datasets by comparing the false positive rate (FPR) among Bloom filters without SEAD Counter and with SEAD Counter. We have opensourced our codes on Github [53].

### A. Experimental Setup

*1) Datasets:*

*a)* IP Trace Datasets: We use the anonymous IP traces collected in 2016 from CAIDA [54]. 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. We consider about 250K packets for our experiment. To provide an evaluation of traces that are several orders of magnitude longer, we also use the anonymous IP traces collected in 2018 from CAIDA, for which we consider 30M packets and 1.46M flows for our experiment, and we denote the dataset as CAIDA-Large.

*b)* Kosarac Dataset: We use this dataset to evaluate our technique in Bloom filters. The Kosarac dataset contains (anonymized) click-stream data of a Hungarian on-line news portal and it is downloaded from [55]. It contains 41270 distinct items and around 8M clicks.

*c)* Synthetic Datasets: Since our goal is to find out how well SEAD Counter sketch performs on datasets with different characteristics, we also generate synthetic datasets following the **Zipf** [56] 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). The flow size is the number of drawn samples from the parameterized Zipf distribution. The domain from which the elements are sampled is $N_d$. We generate them using a performance testing tool, the Web Polygraph [57].

*d)* Datacenter Datasets: We use the datacenter trace from [58]. In the experiments, we also use the five-tuple as the ID of a flow, which includes source IP address, destination IP address, source port, destination port, and protocol. We consider 30M packets and 8.64M flows for our experiment.

*2) Implementation:* We have implemented the sketches of CM, CU and C in C++. We apply the SEAD Counter technique to these sketches, and the results are denoted as SEAD CM, SEAD CU, and SEAD C, respectively. For comparison purposes, we also implemented Counter-Tree in C++, denoted as CT in our experiments. The hash functions used in the sketches are implemented using the 32-bit Bob Hash [59] with different initial seeds. When the width is small, we can compute a single hash for different indices. But when the width is large, We can only compute different hashes for different counters. To avoid such issues, we compute different hashes for different counters. 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 10 and use 20 32-bit Bob Hashes. We set the counters to 32 bits in the classical sketches. In the sketches using the SEAD Counter, the size of the counters is reduced to 16 bits. Furthermore, for each experiment on the datasets, we run 10 sub-experiments. We also implement CBF and VI-CBF in C++. For VI-CBF, We adopt the setting in [52] where $D$ is $D_L = [L, 2L-1]$ and $L$ is set to 4. Therefore, we don't have to save the lookup table and we can rely on a single variable-increment counter per entry, without the additional counter that indicates the number of hashed elements. We apply our method to these Bloom filters, and their results are denoted as SEAD CBF and SEAD VI-CBF. We set the counters to 8 bits in the classical Bloom filters, and 4 bits in the Bloom filters using SEAD Counters.

### 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. We use AAE_E and AAE_M to denote the AAE of elephant flows and mouse flows respectively, and use AAE to denote the AAE of all flows.

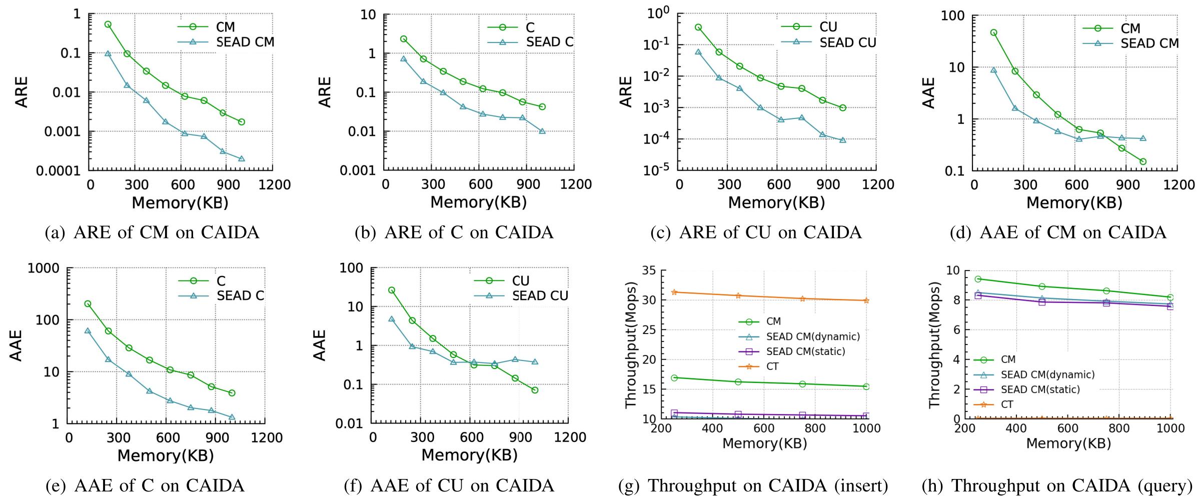
**Average Relative Error (ARE):** ARE is defined as $\frac{1}{|\phi|}\sum_{i\in[n]}\frac{|f_i - \hat{f}_i|}{f_i}$, where $f_i$ is the frequency of token $i$ that appears in the stream, $\hat{f}_i$ is the estimated frequency and $|\Phi|$ is the volume of the set. We use ARE_E and ARE_M to denote the ARE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

**False Positive Rate (FPR):** FPR is defined as the fraction of false positives, *i.e.*, elements which don't appear in the set but are reported to appear by the algorithm.

**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. We want to see how AAE and ARE change when we change per-flow memory or per-packet memory, so we explicit plot AAE/ARE as a curve of per-flow memory or per-packet memory.

**Throughput:** Maximum number of insertions that can be processed per second. We use Mega-operations per second (Mops) as the unit of throughput. We measure the processing speed by feeding one packet of a flow at a time.




(a) ARE of CM on CAIDA (b) ARE of C on CAIDA (c) ARE of CU on CAIDA (d) AAE of CM on CAIDA

(e) AAE of C on CAIDA (f) AAE of CU on CAIDA (g) Throughput on CAIDA (insert) (h) Throughput on CAIDA (query)

Fig. 4. Experiments on CAIDA.

All the experiments about speed are repeated 100 times with $t < 0.05$ for all the experiments to ensure statistical significance.

**Elephant Flows/Mouse Flows:** We define flows whose size is greater than 99% of the flows as elephant flows, and we define other flows as mouse flows.

### *C. Effects of SEAD Counter Technique on Sketches*

*1) Flow Size Measurements on CAIDA Dataset:* Since the Counter-Tree does not support flow volume measurements, we only compare our technique with the original sketch in the ARE and AAE experiments.

**Effect of SEAD Counter on CM Sketch's ARE and AAE (Figures 4(a) and 4(d)):** The range of the memory size in this experiment varies from 125KB to 1000KB. Here, the original CM sketch is compared to the CM sketch using the SEAD Counter (Dynamic Sign Bits version). We plot the ARE and AAE (in log scale) 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 the SEAD Counter. When the memory is below 600KB, the CM sketch using the SEAD Counter have both lower ARE and AAE.

**Effect of SEAD Counter on C Sketch's ARE and AAE (Figures 4(b) and 4(e)):** Our experimental results show that the original C sketch has 15.98 times higher ARE and 4.49 times higher AAE than C sketch using SEAD Counter, when the memory is 1000KB. The SEAD 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 SEAD Counter on CU Sketch's ARE and AAE (Figures 4(c) and 4(f)):** Our experimental results show that the original CU sketch has 10.9 times higher ARE than the CU sketch using the SEAD Counter when the memory is 1000KB. The SEAD Counter technique successfully reduces the ARE and AAE under small memory consumption in the case of the CU sketch. Compared to the results of the CM and C sketches, we see that the CU sketch using the SEAD Counter has the best performance of all.

**Throughput of CM Sketch Using SEAD Counter (Figure 4(g) and 4(h)):** We find that when memory consumption is 1000KB, the throughput (insert) of the Counter-Tree is the highest and is $2.5 \sim 2.9$ times higher than that of CM sketch using the SEAD Counter on CAIDA dataset: Counter-Tree only needs one hash calculation in each insertion. However, to recover the counts from the Counter-Tree, it needs more hash functions to scatter the position of each insertion and its query is much slower. When memory consumption is 1000KB, the throughput (query) of the original CM sketch is the highest.

**Conclusion:** In this set of experiments, we tested our SEAD Counter technique for flow volume measurements. We found that the CU sketch using the SEAD Counter has the best performance of all sketches, hence we suggest its use for flow volume 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 3.0). Here the task consists in measuring the flow size of each flow. We compare the ARE and AAE of the original sketches, of sketches using the SEAD Counter, and of Counter-Tree.

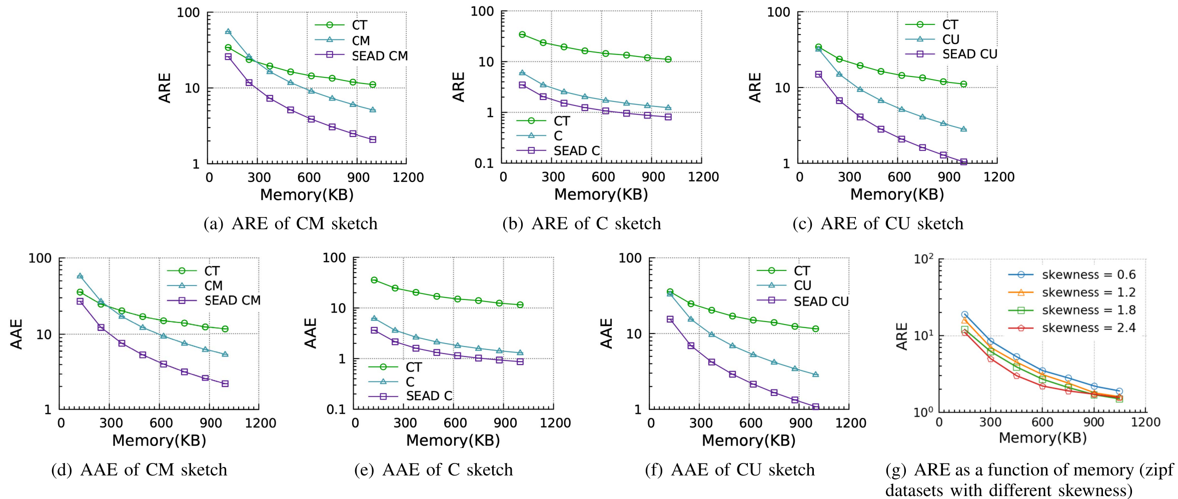
**Effect of SEAD Counter on CM Sketch's ARE and AAE (Figures 5(a) and 5(d)):** We find that when the memory is 1000KB, the original CM sketch has 2.5 times higher ARE and AAE than the CM sketch using the SEAD Counter, while the Counter-Tree has an ARE 5.3 times higher than the CM sketch using the SEAD 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 the SEAD Counter are always the lowest.

**Effect of SEAD Counter on C Sketch's ARE and AAE (Figures 5(b) and 5(e)):** We observe that when the memory is 1000KB, the original C sketch has 1.79 times higher ARE and AAE than the C sketch using SEAD Counter, while the Counter-Tree has an ARE 13.6 times higher than the C sketch using SEAD Counter. The ARE and AAE of the C sketch using the SEAD Counter are the lowest for all considered memory sizes. Note that compared to Counter-Tree, the C sketch using the SEAD Counter improves accuracy by one order of magnitude.

**Effect of SEAD Counter on CU Sketch's ARE and AAE (Figure 5(c) and 5(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 SEAD Counter while

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

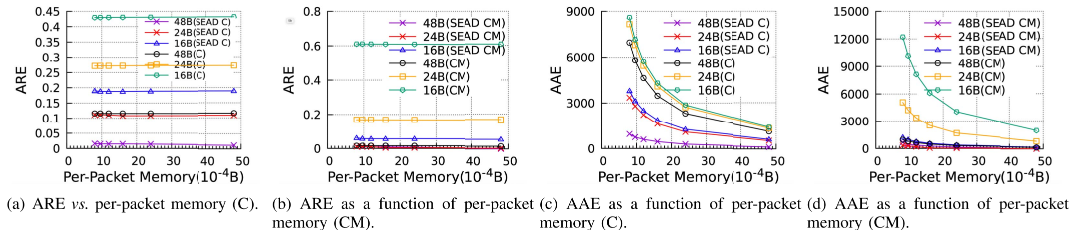





(a) ARE of CM sketch (b) ARE of C sketch (c) ARE of CU sketch

(d) AAE of CM sketch (e) AAE of C sketch (f) AAE of CU sketch (g) ARE as a function of memory (zipf datasets with different skewness)

Fig. 5. Experiments on Synthetic Datasets. (skewness is 1.2 except (g)).




(a) ARE *vs.* per-packet memory (C). (b) ARE as a function of per-packet memory (CM). (c) AAE as a function of per-packet memory (C). (d) AAE as a function of per-packet memory (CM).

Fig. 6. Effect of per-packet memory on ARE and AAE using synthetic datasets.

Counter-Tree has an ARE 10.6 times higher than the CU sketch using the SEAD 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 the SEAD Counter are always lowest.

**CU sketch Using SEAD Counter's ARE on Synthetic Datasets With Different Skewness (Figure 5(g)):** We find that the ARE decreases as skewness increases. 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 the SEAD Counter is also accurate with very skewed datasets.

**Conclusion:** In this set of experiments, we tested the SEAD Counter technique for flow size measurements. In skewed traces, there are large number of mouse flows and the Counter-Tree's variance behaves not well for mouse flows: larger than $flow\ size \times (r-1)$ ($r$ is one parameter much larger than 1) [30]. Therefore, for the Counter-Tree, their AAE and ARE is larger than other sketches. We found that the C sketch using the SEAD Counter has the best performance of all considered 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 6(a) and 6(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 SEAD Counter drops with the increase of per-flow memory consumption. The original C sketch has 2 to 3 times larger AAE than SEAD Counter sketches.

**Effect of Memory Size on CM Sketch's ARE and AAE With Fixed $a$ and Per-Flow Memory Consumption (Figures 6(b) and 6(d)):** We find that the original CM sketch has larger ARE and AAE values compared to the CM sketch using SEAD Counter. The AAE of both the CM sketch and the CM sketch using SEAD Counter drop with the increase of per-flow memory consumption. The original C sketch has more than 10 times larger AAE than SEAD Counter sketches.

**Conclusion:** We found that the SEAD 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 SEAD Counter technique can improve the accuracy under small per-flow memory size.

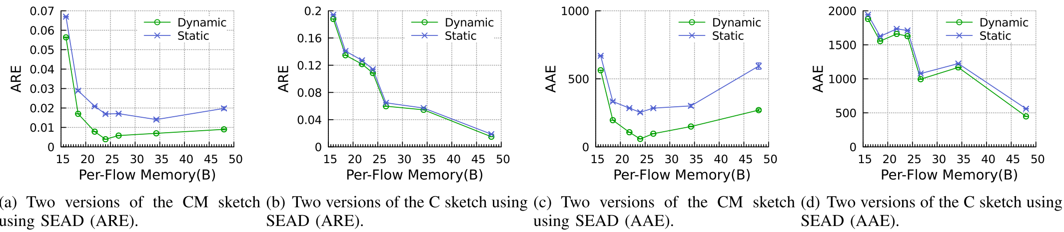
**Effect of the Different Versions of the CM Sketch Using SEAD Counter on ARE and AAE (Figures 7(a) and 7(c)):** In this experiment, we set the memory size of the sketches to be constant and vary the per-flow memory consumption.

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.

**Effect of the Different Versions of C Sketch Using SEAD Counter on ARE and AAE (Figures 7(c) and 7(d)):** We find that when the per-flow memory consumption is 4B, the ARE and AAE of the Static Sign Bits version C sketch with the length of sign bits $s = 3$ is 1.25 times higher than the one of the Dynamic version on average. The ARE and AAE of the CM sketch using SEAD Counter in both versions drops when per-flow memory increases. Under any per-flow memory consumption, the ARE of the Dynamic Sign Bits version CM sketch is smaller.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



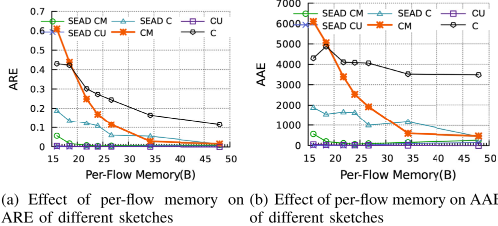



(a) Two versions of the CM sketch using SEAD (ARE). (b) Two versions of the C sketch using SEAD (ARE). (c) Two versions of the CM sketch using SEAD (AAE). (d) Two versions of the C sketch using SEAD (AAE).

Fig. 7. Comparison between the two versions of SEAD on synthetic datasets.

TABLE II
EXPERIMENTS OF SKETCHES ON CAIDA-LARGE

| | AAE | AAE_E | AAE_M | ARE | ARE_E | ARE_M | Throughput(insert) | Throughput(query) |
|---|---|---|---|---|---|---|---|---|
| CU | 14.35 | 474.13 | 9.71 | 6.64 | 0.032 | 6.70 | 7.20 Mops | 10.34 Mops |
| SEAD-CU | **8.35** | **32.12** | 8.11 | 5.52 | **0.0042** | 5.58 | 6.84 Mops | 9.88 Mops |
| SAC-CU | 13.23 | 521.82 | **8.09** | **5.51** | 0.23 | **5.57** | 6.02 Mops | 5.88 Mops |
| ICEBuckets-CU | 10.09 | 194.40 | 8.23 | 5.55 | 0.018 | 5.60 | 2.76 Mops | 3.33 Mops |
| PCU | 10.81 | 51.83 | 10.40 | 6.51 | 0.083 | 6.58 | **9.93 Mops** | **10.60 Mops** |
| CM | 22.79 | 489.74 | 18.08 | 11.10 | 0.064 | 11.21 | 8.06 Mops | 9.94 Mops |
| SEAD-CM | **14.08** | 33.60 | **13.88** | **8.52** | **0.028** | **8.60** | 6.56 Mops | 9.91 Mops |
| SAC-CM | 14.34 | 60.14 | **13.88** | **8.52** | 0.036 | **8.60** | 5.62 Mops | 5.81 Mops |
| ICEBuckets-CM | 14.24 | **23.19** | 14.15 | 8.68 | **0.028** | 8.77 | 3.47 Mops | 3.32 Mops |
| PCM | 14.64 | 53.74 | 14.26 | 11.89 | 0.106 | 12.01 | **10.48 Mops** | **10.06 Mops** |




(a) Effect of per-flow memory on ARE of different sketches (b) Effect of per-flow memory on AAE of different sketches

Fig. 8. Detailed experiment with synthetic datasets.

**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.

**Effect of Per-Flow Memory Consumption on ARE and AAE (Figures 8(a) and 8(b)):** In this experiment, we vary per-flow memory consumption between 1.3B and 4B. The parameter $a$ is set to 0 and the per-packet memory consumption is fixed to 0.0016B. We find that the ARE and AAE of original sketches is more impacted by the decrease of per-flow memory consumption than the one of SEAD Counter sketches. When the per-flow memory consumption drops to 1.3B, the ratio between the ARE of the original sketch and of the SEAD Counter sketch is 10.83 for the CM sketch, 22.93 for the CU sketch and 2.29 for the C sketch. This result is consistent with the conclusion we drew in the last section.

*3) Flow Size Measurements on CAIDA-Large and Datacenter:* We compare the ARE and AAE of the original sketches, of sketches using the SEAD Counter, SAC [38], ICEBuckets [41], and pyramid sketches [36]. We use 12.5 bits per counter for ICEBuckets and 4 bits per counter for Pyramid Sketches, which are the original settings in their papers. We use 12 bits per counter for other methods. The result is shown in Table II and III.

**Effect of the Different Versions of the CU Sketch on CAIDA-Large (Table II):** In this experiment, we set the memory size of the sketches to be 1MB. Here, the original CU sketch is compared to the CU sketch using the SEAD Counter (Dynamic Sign Bits version), SAC, ICEBuckets, and the Pyramid CU sketch. Our results show that when the memory is 1MB, the CU sketch using the SEAD Counter has a lower AAE on all flows, and has a lower AAE and ARE on elephant flows. Though AAE_M, ARE, and ARE_E are larger, they are very close to the best result achieved by SAC. Besides, the insert and query throughput is much higher than that of SAC. PCU has the highest throughput among other methods. This is because PCU uses word acceleration and Ostrich Policy to achieve a higher insertion and query speed. Among the methods that simply replace counters, our method has a higher insertion and query speed.

**Effect of the Different Versions of the CM Sketch on CAIDA-Large (Table II):** In this experiment, we set the memory size of the sketches to be 1MB. Our results show that when the memory is 1MB, the CM sketch using the SEAD Counter has both a lower AAE and ARE on all flows, and the highest throughput among the sketches that simply replacing counters. The CM sketch using ICEBuckets has a lower AAE_E. Among the sketches that simply replacing counters, our SEAD Counter has a lower AAE and ARE, and higher insertion and query speed. PCM has the highest throughput among other methods.

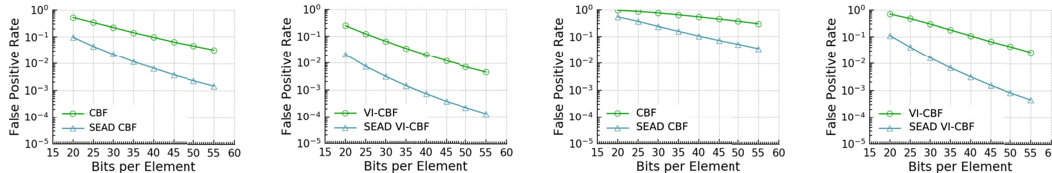
**Effect of the Different Versions of the CU Sketch on Datacenter Trace (Table III):** In this experiment, we set the memory size of the sketches to be 1MB. Our results show that when the memory is 1MB, PCU has both a lower AAE and ARE on all flows, and the highest throughput among other methods. This is because of their Ostrich policy where they ignore the second and higher layers when getting the reported values of the $d$ mapped counters. Therefore, they increment the smallest counter(s) with high probability to achieve better accuracy. Among the sketches that simply replacing counters, our SEAD Counter has a lower AAE and ARE, and higher insertion and query speed. Due to high skewness of this dataset, for ICEBuckets, most flows with a few packets (including some elephant flows in this dataset) are affected by the adjustment of its parameter $\epsilon$ brought by the largest flows in their buckets. Under such a circumstance (highly skewed data streams), our SEAD technique achieves better accuracy.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



TABLE III
EXPERIMENTS OF SKETCHES ON DATACENTER

| | AAE | AAE_E | AAE_M | ARE | ARE_E | ARE_M | Throughput(insert) | Throughput(query) |
|---|---|---|---|---|---|---|---|---|
| CU | 67.50 | 40.79 | 67.77 | 48.05 | 1.05 | 48.53 | 6.36 Mops | 12.60 Mops |
| SEAD-CU | 53.58 | 25.43 | 53.86 | 38.33 | 0.73 | 38.70 | 6.07 Mops | 9.97 Mops |
| SAC-CU | 53.60 | 27.74 | 53.86 | 38.33 | 0.73 | 38.70 | 4.47 Mops | 5.58 Mops |
| ICEBuckets-CU | 56.10 | 29.21 | 56.38 | 40.08 | 0.79 | 40.48 | 2.80 Mops | 3.48 Mops |
| PCU | **30.95** | **21.69** | **31.04** | **22.09** | **0.62** | **22.31** | **9.29 Mops** | **10.73 Mops** |
| CM | 126.89 | 130.79 | 126.85 | 87.94 | 3.59 | 88.79 | 6.16 Mops | 10.60 Mops |
| SEAD-CM | **93.82** | **94.00** | **93.82** | **65.04** | **2.66** | **65.67** | 6.05 Mops | 9.94 Mops |
| SAC-CM | **93.82** | 94.03 | **93.82** | **65.04** | **2.66** | **65.67** | 4.25 Mops | 5.86 Mops |
| ICEBuckets-CM | 98.34 | 98.45 | 98.33 | 68.17 | 2.78 | 68.82 | 3.06 Mops | 3.48 Mops |
| PCM | 174.88 | 175.74 | 174.87 | 121.17 | 4.98 | 122.34 | **10.07 Mops** | **10.09 Mops** |




(a) Comparison of false positive rate between CBF and SEAD CBF on CAIDA dataset (b) Comparison of false positive rate between VI-CBF and SEAD VI-CBF on CAIDA dataset (c) Comparison of false positive rate between CBF and SEAD CBF on Kosarac dataset (d) Comparison of false positive rate between VI-CBF and SEAD VI-CBF on Kosarac dataset

Fig. 9. Comparison between two kinds of bloom filters with and without SEAD Counter on CAIDA and Kosarac dataset.

## *D. Effects of SEAD Counter Technique on Bloom Filters*

We apply our SEAD Counter technique to both Counting Bloom Filters (CBF) and Variable-Increment Counting Bloom Filters (VI-CBF). We use both the static sign bits version and the dynamic sign bits version. The results are denoted as SEAD CBF and SEAD VI-CBF.

*1) Membership Query on CAIDA Dataset:* In this experiment, we perform a membership query for each element of the dataset. We considered the same 250K packets in the previous experiments on CAIDA. We compare the false positive rate of the original bloom filters and of the bloom filters using the SEAD Counter.

**Effect of the Different Versions of CBF Using SEAD Counter on FPR (Figure 9(a)):** The range of the memory size (in bits per element) in this experiment goes from 20 to 55 bits. The original CBF is compared with CBF using the SEAD Counter (Dynamic Sign Bits version). We plot how the FPR changes as a function of memory size. Our results show that the proposed SEAD Counter mechanism can improve upon CBF. CBF with the SEAD Counter yields the better performance, especially for more bits per element. For instance, for 30 bits per element, the original CBF has a false positive rate 10.05 times higher than the CBF with SEAD Counters. Likewise, for 50 bits per element, the SEAD Counters improve the results by 19.12 times.

**Effect of the Different Versions of VI-CBF Using SEAD Counter on FPR (Figure 9(b)):** We find that, for 55 bits per element, the original VI-CBF has a false positive rate 36.5 times higher than the VI-CBF using SEAD Counters. As the memory consumption decreases, the false positive rate of the original VI-CBF becomes close to the VI-CBF using SEAD Counters. However, the improvement from the SEAD Counter is still significant. We can achieve an order of magnitude improvement even when the number of bits per element is as small as 20 bits per element.

**Conclusion:** This set of experiments showed that the Bloom filters using SEAD Counters has better performance than the original Bloom filters in the task of membership query on the CAIDA dataset.

*2) Membership Query on Kosarac Dataset:* In this experiment, we perform a membership query for each element in the Kosarac dataset.

**Effect of the Different Versions of CBF Using SEAD Counter on FPR (Figure 9(c)):** Our results show that the proposed SEAD Counter mechanism can improve upon CBF. For 30 bits per element, the original CBF has a false positive rate 3.17 times higher than the CBF with SEAD Counters. Likewise, for 50 bits per element, the SEAD Counters improve the result by 7.36 times.

**Effect of the Different Versions of VI-CBF Using SEAD Counter on FPR (Figure 9(d)):** For 55 bits per element, the original VI-CBF has a false positive rate nearly two orders of magnitude higher than the VI-CBF using SEAD Counters. As the memory consumption decreases, the false positive rate of the original VI-CBF becomes close to the VI-CBF using SEAD Counters. However, we can still improve the false positive rate using our technique by 6.24 times.

**Conclusion:** This set of experiments showed that the Bloom filters using SEAD Counters has better performance than the original Bloom filters in the task of membership query on the Kosarac dataset.

## *E. Estimators*

We use one counter for each flow and estimate the packet number using the counter. We use AAE and ARE as the metrics to evaluate the performance of the counters. We define flows whose size is greater than 99% of the flows as elephant flows, and we define other flows as mouse flows. We compare our SEAD Counters with SAC [38], ICEBuckets [41], and BRICK [42] on CAIDA-Large and the datacenter trace. The result is shown in Table IV and V. It is shown that our SEAD Counter has a greater throughput for both insertion and query. SEAD Counter works better on mouse flows, and thus achieves a smaller ARE than the other two counters. The AAE of SEAD Counter is larger due to larger AAE of elephant flows, which accounts more for AAE. For BRICK, we use the same parameter settings as the evaluations in [42] with 64 counters per bucket, four levels, and a failure probability of

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



TABLE IV
EXPERIMENTS OF ESTIMATORS ON CAIDA-LARGE

| | AAE | AAE_E | AAE_M | ARE | ARE_E | ARE_M | Throughput(insert) | Throughput(query) |
|---|---|---|---|---|---|---|---|---|
| SEAD | 0.217511 | 21.750474 | 0.000006 | 0.000029 | 0.002237 | 0.000006 | 64.04 Mops | 16.15 Mops |
| SAC | 0.557416 | 55.728089 | 0.000136 | 0.000283 | 0.027915 | 0.000007 | 27.61 Mops | 16.25 Mops |
| ICEBuckets | 0.114035 | 9.841142 | 0.015817 | 0.002977 | 0.001786 | 0.002989 | 9.57 Mops | 13.48 Mops |

TABLE V
EXPERIMENTS OF ESTIMATORS ON DATACENTER

| | AAE | AAE_E | AAE_M | ARE | ARE_E | ARE_M | Throughput(insert) | Throughput(query) |
|---|---|---|---|---|---|---|---|---|
| SEAD | 0.001457 | 0.145599 | 0.000001 | 0.000001 | 0.000005 | 0.000001 | 15.20 Mops | 14.48 Mops |
| SAC | 0.002761 | 0.275979 | 0.000001 | 0.000001 | 0.000036 | 0.000001 | 12.17 Mops | 14.27 Mops |
| ICEBuckets | 0.001124 | 0.104862 | 0.000076 | 0.000012 | 0.000023 | 0.000011 | 6.83 Mops | 10.23 Mops |

$P_f = 10^{-10}$. The probability is relatively low because BRICK maintains exact active counters. We adopt the same failure probability as in [42]. The total storage space required for the CAIDA-Large is 1.75 MB, and the total required for the datacenter trace is 7.68 MB. Though exact statistics counters could record the value accurately, they are not suitable for data flow measurement where the memory is constrained.

### F. Parameter Settings

The choice of $n$ depends on the circumstances where we use the counters. If we assign $n$ bits for a normal counter, it is also safe to use $n$ bits for SEAD Counter. If the memory is constrained, then we could assign fewer bits to SEAD Counter. As shown in our experimental results, we use half the size as the normal counter and still achieve better performance. Therefore, we empirically suggest that when $n$ bits are required in a normal counter, we could assign $\frac{n}{2}$ bits in SEAD Counter.

When the counting part is large, SEAD Counter can have a better accuracy when the value is small since it can count more small values accurately. In the other hand, we can count larger values if the sign part is large enough. As for $s$, we would like to recommend to use $\frac{1}{4}$ of the bits as the sign bits to achieve the balance between the counting range and the counting part. We would highly recommend users to choose dynamic sign bits version since it allows to count more small values accurately than the static sign bits version. What's more, we don't have to tune $s$ in this case.

As for the choice of the expansion array, we recommend using $\gamma[i] = m^i$ though users may choose other expansion arrays. The reasons are twofold. For one thing, it is both easier to implement and faster to process if we choose the expansion array to be a geometric sequence. For the other, it is consistent with the characteristics of common data streams where most flows are small flows. We would like to accurately count the flows when they are small ($\gamma[0] = 1$ makes SEAD Counter work as a normal counter), and allow for some error when we need to count large flows. We empirically discuss the choice of the expansion array. We choose $m$ to range from 1 to 6 and use SEAD Counter as the estimator on CAIDA-Large. It is shown in Table VI that our SEAD counter achieves the best performance when $m = 4$. Note that we could achieve better performance by tuning $m$ if possible, and our general expression of the expansion array allows for improvement by more careful design.

TABLE VI
AAE AND ARE ON CAIDA-LARGE FOR DIFFERENT $m$S

| | AAE | ARE |
|---|---|---|
| m=1 | 4.726804 | 0.000302 |
| m=2 | 1.387580 | 0.000289 |
| m=3 | 0.272841 | 0.000030 |
| m=4 | 0.223005 | 0.000028 |
| m=5 | 0.230716 | 0.000031 |
| m=6 | 0.266696 | 0.000033 |

### G. Discussion

The experiments on both sketches and bloom filters show that the SEAD Counter technique is very generic and can be adapted to all kinds of sketches and bloom filters using counters. The technique achieves good performance in the case of sketches, as our mechanism can improve the space efficiency and count of both mouse and elephant flows with good accuracy. Our algorithm performs consistently well in almost all settings: when we change the per-packet memory, the per-flow memory, the flow sizes, and the skewness of the flows. Our technique is also very useful when applied to VI-CBF, as a typical VI-CBF's counter is 8 bits, and our technique can save half the space per counter. Therefore SEAD VI-CBF achieves both the efficiency and accuracy compared to VI-CBF. As long as the considered data structure uses counters where the memory usage is limited, our method can improve the space efficiency while achieving good accuracy. The SEAD Counter also allows greater counting ranges and provides more flexibility when counting, and we don't need extra space or time to read the counter. Therefore, this technique achieves a good balance between space efficiency and accuracy with little overhead. The expansion array we choose here is very simple, and there may be better choices depending on the specific setting of the task. We encourage readers to explore the use of our mechanism in different applications and try other possibilities of the expansion array.

Compared with estimators, variable-length solutions like Pyramid and BRICK are also superior in some ways. BRICK can maintain exact active counters with a low failure probability. The strength is that the values are exact but the weakness is that it needs much more space. Pyramid Sketch can dynamically assign appropriate number of bits for different items with different frequencies. It has higher throughput due to the word acceleration technique. Also, the CU sketch with Pyramid framework has a higher accuracy due to the Ostrich Policy, which can be seen from Table III. Empirically, SEAD Counter has a higher accuracy on CAIDA-Large. CM sketch with SEAD Counter and CU sketch with Pyramid framework has a higher accuracy on Datacenter.

## VII. CONCLUSION

Thanks to their memory efficient and fast and sufficient speed, sketches have attracted much attention for network measurements. If the flow size distribution is uniform, there is little room for improvements compared to previous work.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



However, when the flow size distribution is highly skewed, existing sketches are very inefficient in memory usage. No previous work could achieve memory efficiency without hurting the sufficient and fast speed, which is really important in high-speed network traffic. To address this, we proposed a generic technique, the *self-adaptive counters (SEAD Counter)*, in two versions, static and dynamic. Our main idea is the following: When a counter is going to overflow, we do not increase it one by one, but increase it by a predefined probability. When the counter is small, it just works like a normal counter. The SEAD Counter makes small counters capable of representing 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 applied the SEAD Counter to three typical sketches: sketches of CM, C, and CU. We extended our technique to two typical Bloom filters: CBF and VI-CBF. We also used our technique as the estimator. Our experimental results showed that, compared to the state-of-the-art, sketches using the SEAD Counter improve the accuracy by up to 13.6 times. The CBF and VI-CBF using the SEAD Counters can improve the false positive rate by up to one or two orders of magnitude respectively.

## Acknowledgment

The authors would like to thank the anonymous reviewers for their thoughtful suggestions.

## References

[1] T. Yang *et al.*, "A generic technique for sketches to adapt to different counting ranges," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 2017–2025.

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

[3] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 1–10.

[4] Y. Zhou, Y. Zhou, S. Chen, and O. P. Kreidl, "Limiting self-propagating malware based on connection failure behavior," in *Proc. 7th Int. Conf. Netw. Commun. Secur. (NCS)*, Dec. 2015, pp. 1–16.

[5] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in SDNs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3587–3601, Dec. 2017.

[6] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, 2004, pp. 101–114.

[7] R. B. Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 127–140.

[8] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: An efficient algorithm for finding heavy hitters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 1, p. 5, Jan. 2008.

[9] R. Schweller, A. Gupta, E. Parsons, and Y. Chen, "Reversible sketches for efficient and accurate change detection over network data streams," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2004, pp. 207–212.

[10] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 129–143.

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

[12] Q. Huang *et al.*, "SketchVisor: Robust network measurement for software packet processing," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 113–126.

[13] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proc. Int. Conf. Manage. Data*, Jun. 2016, pp. 1449–1463.

[14] D. Kim, Y. Zhu, C. Kim, J. Lee, and S. Seshan, "Generic external memory for switch data planes," in *Proc. 17th ACM Workshop Hot Topics Netw.*, Nov. 2018, pp. 1–7.

[15] *Intel FPGA*. Accessed: Sep. 16, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stx3/stx3_siii51001.pdf

[16] *Intel Tofino*. Accessed: Sep. 16, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html

[17] *Xlinx FPGA*. Accessed: Sep. 16, 2020. [Online]. Available: https://www.xilinx.com/products/technology/memory.html#internalMemory

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

[19] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 323–336, 2002.

[20] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. Int. Colloq. Automata, Lang., Program.* Berlin, Germany: Springer, 2002, pp. 693–703.

[21] K. Cheng, L. Xiang, M. Iwaihara, H. Xu, and M. M. Mohania, "Time-decaying Bloom filters for data streams with skewed distributions," in *Proc. 15th Int. Workshop Res. Issues Data Eng., Stream Data Mining Appl. (RIDE-SDMA)*, 2005, pp. 63–69.

[22] I. N. Bozkurt, Y. Zhou, and T. Benson, "Dynamic prioritization of traffic in home networks," in *Proc. CoNEXT Student Workshop*, 2015, pp. 1–3.

[23] G. Cormode, "Sketch techniques for approximate query processing," in *Foundations and Trends in Databases*. Boston, MA, USA: NOW, 2011.

[24] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 267–278, 2009.

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

[26] G. Cormode, B. Krishnamurthy, and W. Willinger, "A manifesto for modeling and measurement in social media," *1st Monday*, vol. 15, no. 9, pp. 1–18, Sep. 2010.

[27] P. B. Gibbons and Y. Matias, "Synopsis data structures for massive data sets," *External Memory Algorithms*, vol. 50, pp. 39–70, Jan. 1999.

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

[29] *CMS in Redis Module*. Accessed: Nov. 3, 2020. [Online]. Available: https://oss.redislabs.com/redisbloom/CountMinSketch_Commands/

[30] C. Min and S. Chen, "Counter tree: A scalable counter architecture for per-flow traffic measurement," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1249–1262, Apr. 2017.

[31] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[32] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of Bloom filters for distributed systems," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 131–155, 1st Quart., 2012.

[33] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2004.

[34] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey," *Comput. Netw.*, vol. 57, no. 18, pp. 4047–4064, Dec. 2013.

[35] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting Bloom filters," in *Proc. Eur. Symp. Algorithms*. Berlin, Germany: Springer, 2006, pp. 684–695.

[36] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li, "Pyramid sketch: A sketch framework for frequency estimation of data streams," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1442–1453, Aug. 2017.

[37] R. Morris, "Counting large numbers of events in small registers," *Commun. ACM*, vol. 21, no. 10, pp. 840–842, Oct. 1978.

[38] R. Stanojevic, "Small active counters," in *Proc. 26th IEEE Int. Conf. Comput. Commun. (INFOCOM)*, May 2007, pp. 2153–2161.

[39] C. Hu *et al.*, "Discount counting for fast flow statistics on flow size and flow volume," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 970–981, Jun. 2014.

[40] E. Tsidon, I. Hanniel, and I. Keslassy, "Estimators also need shared values to grow together," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1889–1897.

[41] G. Einziger, B. Fellman, R. Friedman, and Y. Kassner, "ICE buckets: Improved counter estimation for network measurement," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1165–1178, Jun. 2018.

[42] N. Hua, J. J. Xu, B. Lin, and H. C. Zhao, "BRICK: A novel exact active statistics counter architecture," *IEEE/ACM Trans. Netw.*, vol. 19, no. 3, pp. 670–682, Jun. 2011.

[43] C. Hu *et al.*, "Discount counting for fast flow statistics on flow size and flow volume," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 970–981, Jun. 2014.

[44] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1622–1634, Oct. 2012.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



[45] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. 28th Annu. ACM Symp. Theory Comput. (STOC)*, 1996, pp. 20–29.
[46] N. Hua, A. Lall, B. Li, and J. Xu, "A simpler and better design of error estimating coding," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 235–243.
[47] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Found. Trends Databases*, vol. 4, nos. 1–3, pp. 1–294, 2012.
[48] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.
[49] Y. Yu, D. Belazzougui, C. Qian, and Q. Zhang, "Memory-efficient and ultra-fast network lookup and forwarding using Othello hashing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1151–1164, Jun. 2018.
[50] 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 *Proc. Internet Meas. Conf.*, Nov. 2014, pp. 223–234.
[51] J. Liu, P. Zhang, H. Wang, and C. Hu, "CounterMap: Towards generic traffic statistics collection and query in software defined network," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 1–5.
[52] O. Rottenstreich, Y. Kanizo, and I. Keslassy, "The variable-increment counting Bloom filter," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1092–1105, Aug. 2014.
[53] *The Source Codes of SEAD Counter at GitHub*. Accessed: Dec. 3, 2020. [Online]. Available: https://github.com/SEADCounter/SEADCounter
[54] *The CAIDA Anonymized 2016 Internet Traces*. Accessed: May 16, 2018. [Online]. Available: http://www.caida.org/data/overview/
[55] *Real-Life Transactional Dataset*. Accessed: Apr. 6, 2019. [Online]. Available: http://fimi.ua.ac.be/data/
[56] D. M. Powers, "Applications and explanations of Zipf's law," in *Proc. Joint Conf. New Methods Lang. Process. Comput. Natural Lang. Learn.* Stroudsburg, PA, USA: Association for Computational Linguistics, 1998, pp. 151–160.
[57] A. Rousskov and D. Wessels, "High performance benchmarking with web polygraph," *Softw.-Pract. Exper.*, vol. 34, no. 2, pp. 187–211, 2003.
[58] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, 2010, pp. 267–280.
[59] *Hash Website*. Accessed: May 19, 2018. [Online]. Available: http://burtleburtle.net/bob/hash/evahash.html

**Xilai Liu** graduated from Peking University, advised by T. Yang. He is currently pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences (UCAS). His research interests include data sketches, in-network computing, and data stream processing systems.

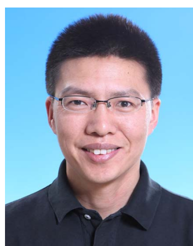
**Yan Xu** received the bachelor's degree from Peking University, advised by T. Yang. He is currently pursuing the M.Sc. degree with the University of Maryland. His research interest includes artificial intelligence.

**Peng Liu** is currently pursuing the master's degree with Peking University, advised by T. Yang. He has published several papers in network area. He is interested in networks and data stream processing.

**Tong Yang** (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University in 2013. He visited the Institute of Computing Technology, Chinese Academy of Sciences (CAS). He is currently an Associate Professor with the Department of Computer Science, Peking University. He has published papers in SIGCOMM, SIGKDD, SIGMOD, SIGCOMM CCR, VLDB, ATC, ToN, ICDE, and INFOCOM. His research interests include network measurements, sketches, IP lookups, Bloom filters, sketches, and KV stores.

**Jiaqi Xu** (Graduate Student Member, IEEE) graduated from Peking University, with double degree in physics and computer science. He is currently with the Queen Mary University of London. His research interests include wireless communication and network measurements.

**Lun Wang** received the bachelor's degree from Peking University, advised by T. Yang. He is currently pursuing the Ph.D. degree with the EECS Department, UC Berkeley, advised by Prof. D. Song. He is also a young passionate researcher with a broad interest in privacy and security, including differential privacy, applied cryptography, programming language theory, robust machine learning, and blockchain systems.



**Gaogang Xie** received the Ph.D. degree in computer science from Hunan University in 2002. He is currently a Professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences (UCAS). His research interests include internet architecture, packet processing and forwarding, and internet measurement.

**Xiaoming Li** (Senior Member, IEEE) is currently a Professor in computer science and technology and the Director of the Institute of Network Computing and Information Systems (NCIS), Peking University, China. His current research interests include search engine and web mining. He led the effort of developing a Chinese search engine (Tianwang) since 1999, and is the Founder of the Chinese web archive (Web InfoMall).

**Steve Uhlig** received the Ph.D. degree in applied sciences from the Catholic University of Louvain, Belgium, in 2004. From 2004 to 2006, he was a Post-Doctoral Fellow with the Belgian National Fund for Scientific Research (F.N.R.S.). His thesis won the annual IBM Belgium/F.N.R.S. Computer Science Prize 2005. From 2004 to 2006, he was a Visiting Scientist at Intel Research Cambridge, U.K., and at the Applied Mathematics Department, The University of Adelaide, Australia. From 2006 to 2008, he was with Delft University of Technology, The Netherlands. Prior to joining Queen Mary, he was a Senior Research Scientist with Technische Universität Berlin/Deutsche Telekom Laboratories, Berlin, Germany. In January 2012, he was a Professor of networks and Head of the Networks Research Group, Queen Mary University of London. From 2012 to 2016, he was a Guest Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.