Noise Measurement and Removal for Data Streaming Algorithms with Network Applications

Chaoyi Ma, Haibo Wang, Olufemi Odegbile, Shigang Chen
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA
Email: {ch.ma, wanghaibo, oodegbile, sgchen}@ufl.edu

***Abstract*—Data streaming has multiple applications on the Internet including traffic measurement and intrusion detection. The bedrock underlying these applications is a set of data streaming algorithms that extract useful information from network packet stream, estimate the needed statistics such as the frequencies of TCP flows, and feed them to application software. Among such algorithms, counting sketches are most prevalent, which are very compact but do so at the cost of errors in their estimations. The dominant error-control method that has been widely accepted for more than a decade is to take the min error from multiple independent estimations. This method produces a positively-biased error and the error can grow large under stringent performance and resource conditions, but no existing work makes an intensive study of this error. This paper investigates the property of the error, which is also known as noise, and claims that it can be measured and removed so as to make the estimations unbiased. We introduce two new ideas, *d-smallest noise* and *artificial data items* for measuring the noise. Based on these two ideas, we propose four noise measurement methods. The mathematical analysis and experimental results based on real network traces show that by removing the measured noise, the error of estimations will be reduced to a much lower level than what the state of the art can do.**

I. Introduction

Data streaming algorithms process continuous streams of data items in real time, often under resource constraints due to performance, cost and other reasons. They have wide applications on the Internet, including traffic measurement [1], [2], [3], intrusion detection [4], heavy hitter detection [5], [6], [7], iceberg identification [8], web services [9] and social networking [10].

A key function of the streaming algorithms is to estimate the *frequency of occurrences* that each data item appears in a given stream. Numerous networking applications can take advantage of this function after modelling their input as a data stream. For instance, consider a packet stream that arrives at a router. We model each packet as a data item, which can be arbitrarily defined according to application need. As an example, it may be the TCP flow identifier carried in packet headers, consisting of source address, destination address, protocol ID, source port and destination port. A stream of x, y, y, z, x, x represents six packets, three from TCP flow x, two from flow y and one from flow z. The frequencies (i.e., packet counts) of flows x, y and z are 3, 2 and 1, respectively. Measuring the packet count of each flow helps network operators understand traffic distributions and respond in real time for traffic balancing, traffic shaping and quality of service. The problem of frequency measurement becomes challenging if there are billions of packets that arrive at a rate of many millions per second, in which case the streaming algorithm may have to be implemented on a network processor using on-chip cache memory where packet forwarding and other networking functions are also implemented, which in turn aggravates the contention of scarce on-chip resources.

For additional Internet application examples, consider a search engine and model the sequence of search requests as a stream where each data item is a search key. Knowing the frequency of each key helps us determine which search results should be cached for faster response and which keys should be suggested to users as they type. Consider a social network and model the sequence of posts visited by users as a stream where each visited post is a data item. Knowing the frequency of each post helps us identify the trend of user interest in real time. Consider an e-commerce web site and model the sequence of products browsed as a stream where each browsed product is a data item. Knowing the frequency of each product helps us recommend popular products to online consumers.

In order to achieve extraordinarily high throughput, the algorithmic operations on streaming data should be kept very simple, and high-speed memory such as SRAM is sometimes preferred for speed, despite its small size. Such design constraints often lead to conflict between the requirement of simple processing and the desire for functionality and accuracy, as well as conflict between limited memory and a very large amount of data to be handled. One solution for the above conflicts is to condense data into a much smaller summary using compact data structures called counting sketches [11], [12], [13], [7], [14]. The summary is able to provide approximate frequency estimates. A great advantage of the counting sketches is that they can work with a small memory that has a much fewer number of counters than the number of different data items, yet still able to provide an estimate for item frequencies. However, as a tradeoff, their estimates carry errors.

The dominating error control mechanism in use is the min-error method [11], [12], which together with its many variants has been adopted widely in the data streaming literature [9], [6], [2], [7], [15], [5], [1]. Its basic idea is to map each data item x to multiple counters that are shared by others. While each counter carries the frequency count of x plus noise from other items, the smallest value of these counters carries the minimum

ISBN 978-3-903176-39-3

error. Due to its wide adoption, the excellence (or weakness) of the min-error method has profound positive (or negative) impact on a large body of data streaming work. Likewise, any superior alternative design that replaces it will go beyond the streaming algorithms themselves to improve the performance of numerous applications that build on top of them.

This paper casts doubt on the min-error method because it produces positively-biased noise (i.e., residual error) that grows large under stringent resource and performance constraints. There exists prior work [13], [7], [14] that abandons the min-error method for non-biased estimations, but while being non-biased, their actual errors are oftentimes much larger than the min-error method, as our experimental results will demonstrate.

Our observation is that estimation accuracy will be improved if we can further remove the positively-biased noise from the min-error method. However, it is challenging to measure such noise accurately and no prior art has studied how to. This paper investigates the property of the noise and finds that it can be effectively measured. We propose four noise measurement methods that work online or offline, some using artificial data items. We have implemented the noise measurement methods in both software and hardware on CPU/GPU/FPGA platforms. We use real-world network packet streams for a comprehensive experimental study to evaluate the performance of the proposed methods and compare them with existing work. The experiment results show that by removing the measured noise, we produce much more accurate frequency estimates than the streaming algorithms based on the min-error method and other streaming algorithms as well.

II. Problem Statement

A data stream is a sequence of data items that continuously arrive in real time, where any item may appear in the steam for an arbitrary number of times. A data streaming algorithm makes a single parse over a data stream to measure information from its items. A key measurement is called *item frequency*, which is the number of occurrences that each item appears in a stream during a measurement epoch. Divided by the total count of all items, it becomes a frequency statistic. However, for convenience, we will refer to the frequency of data item x simply for the count of x in the stream, denoted as f_x.

Efficient streaming algorithms have been proposed to estimate item frequencies using compact sketches that trade less accuracy for better processing/memory efficiency [1], [15], [5], [7], [6], [9], [2]. The most dominating error control mechanism adopted by these algorithms is the min-error method [11], [12]. However, this long-time method leaves positively-biased noise which can be very large under tight resource conditions. The problem is how to measure such noise so that it can be removed, which is not previously studied in the literature. The new noise measurement methods have the potential of improving a large number of streaming algorithms by upgrading them from the min-error method to new ones with noise removal.

III. Motivation

We first review the existing error control methods and then present two new ideas for noise measurement.

A. Min-Error Method

Many existing streaming algorithms [1], [15], [5], [7], [6], [9], [2] are designed with data structures similar to or extended from Count-Min [11], denoted as CM, which uses d counter arrays, C_i, $0 \leq i < d$, each of length l. We denote the jth counter of the ith array as $C_i[j]$, $0 \leq j < l$. Initially, all counters are set to zero. The two operations of CM are (1) recording each item occurrence from a data stream and (2) querying the frequency (i.e., count) of any given data item.

To record an occurrence of item x, we apply d independent hash functions, H_i, $0 \leq i < d$, which map x to d counters, $C_i[H_i(x)]$, one in each array, and increase them by one.

To query for the frequency of item x, we still hash x to its d counters, $C_i[H_i(x)]$, $0 \leq i < d$, and take their smallest value as an estimate for f_x, which is denoted as $\hat{f}_x$.

$$\hat{f}_x = \min\{C_i[H_i(x)] \mid 0 \leq i < d\} \tag{1}$$

We can record far more items than the total number of counters in C. Multiple items may be hashed to the same counter; the counter value is the total frequency of these items. Therefore, $C_i[H_i(x)]$ is the sum of f_x (*information*) and the frequencies of other items also hashed to this counter (*noise*) — the latter causes error in the estimate. The min-error method (1) takes the smallest error among the d counters. Note that all counters of x are at least f_x because they all record the f_x occurrences of x, plus noise. Hence, $\hat{f}_x - f_x \geq 0$. The estimate $\hat{f}_x$ from the min-error method is positively biased.

As we record items, we can think of each item depositing error for others. The average counter value in C roughly characterizes the average error level. The effectiveness of the min-error method can be further improved by lowering the average error level. When recording an occurrence of item x, Min-Counter Update [12], denoted as CU, only increases the smallest among the d counters of x. For example, suppose $d = 4$, and $C_0[H_0(x)] = 5$, $C_1[H_1(x)] = 6$, $C_2[H_2(x)] = 7$, $C_3[H_3(x)] = 5$. When receiving the next occurrence of x, CU only increases $C_0[H_0(x)]$ and $C_3[H_3(x)]$ by one. Our experiments will show that error of CU is much less than that of CM, but CU incurs more processing overhead, which means smaller throughput, particularly on hardware or GPU platform.

B. Min Error Can Be Large

We observe through experiments that the min error, $\hat{f}_x - f_x$, can still be large in situations where the number of counters is much fewer than the number of data items, resulting in serious hash collisions. For example, consider a packet stream received by a high-speed router at multi-terabits per second. To achieve such an extremely high packet rate, modern routers perform routing-table lookups and other essential functions on the data plane at the network interfaces, bypassing the

control panel of main memory and CPU almost entirely. On-chip cache memory is used for packet processing. It has a limited size and has to be shared among packet routing, traffic engineering and measurement functions. Moreover, there can be multiple independent measurement functions for different types of statistics. Consider one measurement function, where we abstract each packet to a flow identifier (which is the data item identifier in our model). For each flow x, we want to measure its frequency (i.e., flow size). Suppose that 10^7 bits of on-chip memory is allocated for this function, which translates into about 3.1×10^5 counters, with each counter 32 bits. If there are 10 millions of concurrent flows and each of them is hashed to $d = 4$ counters, then each counter will have to record 128 flows on average. With such a high noise level, the min error for any flow is likely to be very high.

C. Non-biased Error Control Methods

There are other error control methods. But they are mostly geared toward addressing a different problem: The min error method produces a *biased* estimate $\hat{f}_x$, which is true for CM [11] and CU [12] and also true for the numerous streaming algorithms based on them [9], [6], [2], [7], [15], [5], [1].

CountSketch (CS) [13] adopts an error canceling method. It hashes each item x to d counters. For each of the d counters, a bit from the corresponding hash value, 0 or 1, will determine whether the counter is increased or decreased by one when recording an occurrence of x. Consider other flows (noise) that are also hashed to this counter. Some of them will increase the counter while others will decrease it; they will thus be statistically cancelled out. The mean error in each counter is 0; the counter value is thus a statistically non-biased estimate of f_x. We take the medium value of the d counters as the estimate $\hat{f}_x$. Although this estimate is non-biased, it still carries significant error under tight memory due to large variance. NitroSketch [7] extends CS with geometric sampling. It improves significantly on throughput but its error is considerably larger.

Count Mean Min (CMM) [14] adopts an error reduction method. Its recording operation is the same as CM. To answer a query on item x, from each of the d counters, CMM subtracts away the average value of other counters in the same array. Then it takes the medium value of the d counters after subtraction. It can be proved that this estimate is non-biased, but its error is again significant, consistently much higher than CM or CU in our experiments.

D. First Idea: Mean of d-Smallest Noise

To improve measurement accuracy, we expect that the error can be fully discerned and subtracted from the estimate. The min-error method picks the smallest noise among d counters, which is referred to as *d-smallest noise*. The noise now is reduced to the d-smallest noise, which is an improvement but the min-error method is not a thorough error control method as the frequency estimate is the actual frequency plus the d-smallest noise. Worse, the d-smallest noise can still be very large, as we have explained in Section III-B. This paper tries to close the gap between the estimate and the actual item frequency caused by the d-smallest noise. Our key insight is that we can measure the statistical mean of d-smallest noise and subtract the mean from the frequency estimates of all data items. By doing so, not only do we reduce noise but also produce non-biased estimates. The key is how to measure d-smallest noise without incurring significant additional overhead. Let's imagine a data item of frequency zero, which is a data item that does not belong to the data stream. A simple method for generating data items that will not appear in the data stream is given here. Suppose a real data item x is represented by q bits. We extend it to $q + 1$ bits by adding a bit with the value of 0 in the front of the original bits. Then when we want to generate data items that will not appear in the data stream, we randomly generate data items with q bits and add a bit with the value of 1 in front of them. Those data items will never appear in the real data stream. For each data item of frequency zero, we hash it to d counters. These d counters are pseudo-randomly chosen from the arrays. Its frequency estimation, i.e., the min of the d counters, is purely noise and therefore represents a random sample from the d-smallest noise distribution. Taking a sufficient number of such samples, we average them as a measurement of the mean, which is then subtracted from all frequency estimates.

E. Second Idea: Artificial Data Items

The above idea, however, cannot be applied to CU, which only increments the smallest counter(s) when recording each data item. We find that the d-smallest noise under CU varies for different data items x, depending on their frequencies f_x. This can be demonstrated through experimental results in Table I (under the parameter settings and using the data streams that will be explained in Section VII-B). We can see that small-frequency data items have higher d-smallest noise in their estimates, whereas high-frequency data items have lower d-smallest noise. To measure such frequency-dependent noise, we introduce artificial data items whose frequencies are pre-set in different ranges. We can measure the average d-smallest noise among the artificial items in each range. When querying for the frequency of a (real) item, we reduce its estimate from CU by the average d-smallest noise of a range, which is selected such that the result after subtraction will fall in this range.

f_x	$\lvert\hat{f}_x - f_x\rvert$	f_x	$\lvert\hat{f}_x - f_x\rvert$
1-1	151.8	2-2	150.6
3-4	149.7	5-8	147.7
9-16	143.4	17-32	137.1
33-64	122.1	65-128	95.85
129-256	55.92	257-512	28.99

TABLE I: Measured error $|\hat{f}_x - f_x|$ under CU with respect to item frequency f_x. The error decreases as f_x increases. This is not true for CM, where error stays similar.

IV. D-SMALLEST MEAN NOISE MEASUREMENT

Based on the idea of d-smallest noise, we design a method that measures the mean of d-smallest noise. By removing the measured noise, we produce non-biased estimates. We use CM as the baseline sketch to present the method. We first consider offline measurements and then online measurements.

A. Design

We randomly generate m fake data items $X[t]$, $0 \le t < m$, that are not in the data stream. Their actual frequencies are zeros. For each of them, when we take the smallest value of its d counters, that value is entirely noise, which is thus a sample from the d-smallest noise distribution, denoted as follows:

$$n_{X[t]} = \min\{C_i[H_i(X[t])] \mid 0 \le i < d\}. \tag{2}$$

We use the average value of all the d-smallest noise samples as an estimate for its mean, denoted as N: $N = \sum_{t=0}^{m-1} n_{X[t]}/m$.

The above method is called d-smallest mean noise measurement method, which is also abbreviated as MN. Given a user query on data item x, the noise can be removed as follow:

$$\hat{f}_x = \min\{C_i[H_i(x)] \mid 0 \le i < d\} - N \tag{3}$$

For this approach to work, it must be true that the noise mean is independent of the actual frequencies of the data items, which we will prove in Section VI for MN.

B. Fast Online Noise Measurement

A serious disadvantage of basic MN is the low online throughput. Considering an arbitrary item x, if we want to measure its noise online, we have to query m fake data items. Therefore, we have to query dm counters. By comparison, the querying operation of CM only needs to query d counters. To conquer this disadvantage, we adopt the idea of online noise updating and present the d-smallest mean noise online measurement method (MN-O) as follows.

We randomly generate a number m of fake data items $X[t], 0 \le t < m$, that are supposed not in the data stream. Their actual frequencies are always zero through the process of online recording. We create another counter array $T[t], 0 \le t < m$, where the tth counter is used to store the noise for fake item $X[t]$. Besides, we keep another counter S which stores the sum of all t counters in T. Every time after we recording α real data items, we pick a fake data item $X[t]$ in order, query its noise and update $T[t], S$.

This kind of updating will reduce the online recording throughput by approximately $\gamma = \frac{1}{1+\alpha}$. When $\alpha \ge 9$, we have $\gamma \le 10\%$, which is acceptable. The memory requirement will also be increased. In additional to dl counters in C, we need $2m+1$ extra counters which are used to store $X[t]$, $T[t]$, and S. Therefore, the memory requirement will increase by $\frac{2m+1}{dl}$. This value is usually very small.

By online noise updating, we can do fast online noise measurement. We measure d-smallest noise as S/m, which may cause error. Fortunately, with suitable settings of m, α, this error can be less than 1, which is negligible according to Corollary 2. With the new design, we only need to query 1 counter to calculate noise of an item x online.

V. D-SMALLEST MEAN NOISE MEASUREMENT WITH ARTIFICIAL DATA ITEMS

A. Design

Recall that MN measures the d-smallest mean noise, which is identically distributed for all data items for CM [11] and similar algorithms. However, we discover through experiments that this is not true for other algorithms such as CU [16], [12], where the d-smallest noise for data item x is dependent on f_x; see discussion in Section III-E. Therefore, we should measure such a noise mean independently for each frequency range — measurement for each individual frequency value is however too costly and practically unnecessary. With this idea, we propose a new method called the d-smallest mean noise measurement with artificial data items (MN-AI).

Suppose we have m artificial data items of frequency f and we record them evenly when we record the true data items in the data stream. At the end of a measurement epoch, we use the average d-smallest noise of those m data items as the estimate of d-smallest mean noise for data items with frequency of f. In order to reduce the number of artificial data items, we can pre-define a set of frequency ranges and pre-generate a certain number of artificial data items within the preset frequency ranges. The method of selecting frequency is also an important point. In practice, we can set k frequencies $0 \le f_{A_0} < f_{A_i} < ...f_{A_{k-1}}$. These k frequencies directly divide the frequencies into k ranges:

$$R_i = \begin{cases} [0, (f_{A_0} + f_{A_1})/2), & i = 0 \\ [\frac{f_{A_{i-1}} + f_{A_i}}{2}, \frac{f_{A_i} + f_{A_{i+1}}}{2}), & 0 < i < k-1 \\ [\frac{f_{A_{k-2}} + f_{A_{k-1}}}{2}, f_{A_{k-1}} + \frac{f_{A_{k-1}} - f_{A_{k-2}}}{2}), & i = k-1 \end{cases} \tag{4}$$

The setting of $f_{A_i}, 0 \le i < k$ should be related to the size of the data stream, e.g. sum of frequencies of all items in the data stream. For a large data stream, we need to set larger frequencies. Here we propose one way to set suitable frequencies which are positively related to the size of the data stream. We can preset k percentage $0 \le p_0 \le p_1 \le ...p_{k-1}$ and let $f_{A_i} = F \cdot p_i$, where F is size of the data stream. It is easy to create and record artificial items under this definition. For an artificial item z, we record one occurrence of it every time after recording a number $b_i = \frac{1}{p_i}$ of real data items from the data stream. After a measurement epoch (recording all F items), the frequency of the artificial data item z we recorded is obviously $f_{A_i} = F \cdot p_i$. For each range, we generate m different artificial data items $A_i[j], 0 \le i < k, 0 \le j < m$. We calculate the average d-smallest noise of those m artificial data items as the estimate of d-smallest mean noise within R_i. We denote the d-smallest mean noises as $n_{A_i}, 0 \le i < k$.

We describe how to remove the noise of a data item below. Let $\hat{f}_x$ be the frequency of x obtained before noise removal. We first find a range that $\hat{f}_x$ belongs to in (4) and determine if $\hat{f}_x - n_{A_i}$ is still in the range. If $\hat{f}_x - n_{A_i}$ falls to a lower range j, we then subtract the noise of R_j from $\hat{f}_x$, that is $\hat{f}_x - n_{A_j}$. If it is in R_j, then the estimated frequency of f is $\hat{f}_x - n_{A_j}$. Otherwise, we repeat the process above until we successfully find the range. Besides, if $\hat{f}_x$ is larger than the largest range R_{k-1}, we simply assume its noise as 0.

MN-AI increases computational overheads because of online recording of artificial data items but since the percentage we use is usually very small, it has little influence on throughput as the experiments will show.

B. Fast Online Noise Measurement

Similar to MN, MN-AI cannot support fast online noise measurement because we have to query dkm counters when querying noise online. The throughput is rather low. To solve this problem, we take a similar approach to what we describe in Section IV-B. We keep k counter arrays of length m, $T'_i, 0 \leq k < 0$. The jth counter in ith array store the measured noise for artificial item $A_i[j]$. Besides, we keep a counter array S' of length k, where $S'[i] = \sum_{j=0}^{m-1} T'_i[j]$. Every time after we record α real items, we pick one artificial item from each range, query their noise and update the related counter in T' and S'. This method is called d-smallest mean noise online measurement with artificial data items (MN-O-AI).

MN-O-AI reduces the online recording throughput by approximately $\gamma = \frac{k}{k+\alpha} = \frac{1}{1+\frac{\alpha}{k}}$. In practice, if we keep $\frac{\alpha}{k} \geq 9$, then $\gamma \leq 10\%$, which is definitely acceptable. Besides, from Theorem 3, a smaller α can promise a better estimate of noise. Therefore, we may set a smaller α to get better estimation while losing some throughput. This is a tradeoff between accuracy and throughput. There is another tradeoff here. In order to update the noise, we need $k(m+1)$ additional counters which increase the memory requirement by $\frac{k(m+1)}{dl}$ comparing with original CU. Larger m can give us more accurate estimate of noise but it will increase the memory requirement in the meanwhile. The expectation of errors in noise estimates for every range are also less than 1 with suitable setting, which is negligible according to Corollary 3.

By online noise measurement, MN can support fast online query. This main point is using $S'[i]/m$ as the estimate of n_{A_i}. Besides, f_{A_i} and R_i should be recalculated according to the number of items recorded before the query.

VI. Analytical Results

A. Analytical Results for MN and MN-O

For MN to work, we have assumed that the mean of d-smallest noise is the same for all data items x, regardless of their frequency values f_x. That is the reason why we only use MN to measure this mean once and then subtract the estimates of all data items by this mean noise. Below, we first present Theorem 1 to support our assumption and then give the error bounds for the frequency estimates after the mean noise measured by MN and MN-O is removed.

Theorem 1: Consider an arbitrary item x in the data set X with frequency estimate f_x. Let $n_{i,x}, 0 \leq i < d$ be the noises in the d counters of x. Assuming $f_x \ll F$, where F is the sum of the frequencies of all data items, the expectation and variance of $n_{i,x}$ are give as follows:

$$\begin{aligned} &E(n_{i,x}) \approx F/l \\ &Var(n_{i,x}) \approx (1/l - 1/l^2) \sum\nolimits_{x' \in X} f_{x'}^2. \end{aligned} \tag{5}$$

Proof 6.1: In CM, we have $C_i[H_i(x)] = f_x + \sum_{H_i(x)=H_i(x'),x\neq x'} f_{x'}$, where $C_i[H_i(x)]$ is the counter for x in ith array. Then $n_{i,x} = \sum_{H_i(x)=H_i(x'),x\neq x'} f_{x'}$ is the noise in this counter. According to Section III-A, we have

$$\hat{f}_x = f_x + \min\{n_{i,x} | 0 \leq i \leq d-1\} \tag{6}$$

Let $I_{i,x,x'}$ be an indicator variable, which is 1 when $x' \neq x, H_i(x) = H_i(x')$ and 0 otherwise. Using the indicator variable, we have

$$\begin{aligned} n_{i,x} &= \sum\nolimits_{x' \neq x} I_{i,x,x'} f_{x'} \\ E(n_{i,x}) &= \sum\nolimits_{x' \neq x} E(I_{i,x,x'}) f_{x'} = 1/l(F - f_x), \\ E(n_{i,x}^2) &= E((\sum_{x'_1 \neq x} I_{i,x,x'_1} f_{x'_1})(\sum_{x'_2 \neq x} I_{i,x,x'_2} f_{x'_2})) \end{aligned} \tag{7}$$

$$\begin{aligned} &= E(\sum\nolimits_{x'_1 \neq x, x'_2 \neq x} I_{i,x,x'_1} I_{i,x,x'_2} f_{x'_1} f_{x'_2}) \\ &= \sum\nolimits_{x'_1 \neq x, x'_2 \neq x, x'_1 \neq x'_2} E(I_{i,x,x'_1} I_{i,x,x'_2}) f_{x'_1} f_{x'_2} + \\ &\quad \sum\nolimits_{x'_1 \neq x, x'_1 = x'_2} E(I_{i,x,x'_1} I_{i,x,x'_2}) f_{x'_1} f_{x'_2} \\ &= \frac{1}{l^2} \sum_{x'_1 \neq x, x'_2 \neq x, x'_1 \neq x'_2} f_{x'_1} f_{x'_2} + \frac{1}{l} \sum_{x'_1 \neq x} f_{x'_1}^2 \\ &= \frac{1}{l^2}(F^2 - 2F \cdot f_x + f_x^2) - (\frac{1}{l} - \frac{1}{l^2}) \sum\nolimits_{x'_1 \neq x} f_{x'_1}^2 \end{aligned} \tag{8}$$

Since $f_x \ll F$, we have

$$E(n_{i,x}) \approx F/l \tag{9}$$

Applying (7) and (8) to $Var(n_{i,x}) = E(n_{i,x})^2 - E(n_{i,x}^2)$,

$$Var(n_{i,x}) = E(n_{i,x})^2 - E(n_{i,x}^2) \approx (\frac{1}{l} - \frac{1}{l^2}) \sum_{x'_1 \in X} f_{x'_1}^2. \tag{10}$$

We can see that the expectation and variance of $n_{i,f}$ are independent from f_x. Since $n_x = min\{n_{i,x} | 0 \leq i \leq d-1\}$, we know $E(n_x)$ is also independent from f_x.

Corollary 1: Consider any item x in the data set X with frequency f_x. The expectation of the d-smallest noise (denoted as n_x) in CM is independent from f_x, assuming $f_x \ll F$, where F is the sum of the frequencies of all data items.

Proof 6.2: From the definition of n_x, we know

$$n_x = \min\{n_{i,x} | 0 \leq i \leq d-1\}. \tag{11}$$

From Theorem 1, when $f_x \ll F$, the expectation and variance of $n_{i,x}$ are independent from f_x. Besides, from the recording method of MN, we know that $n_{i,x}, 0 \leq i < d$ are all independent from each other. Therefore, $E(n_x)$ should also be independent from f_x.

Theorem 2: Consider an arbitrary item x with frequency f_x. We denote the value of expectation of d-smallest noise (n_x) as $E(n_x)$, which is measured through MN. By removing it, we calculate frequency estimate of item x (denoted as $\hat{f}_x$), and the bias of $\hat{f}_x$ can be bounded as

$$Prob(|\hat{f}_x - f_x| \geq \frac{eF}{l} - E(n_x)) \leq \frac{1}{e^d} \tag{12}$$

where F is the sum of frequencies of all data items, d, l are parameters in CM and e is Euler's number.

Proof 6.3: The left part of (12) can be rewritten as

$$\begin{aligned} &Prob(|\hat{f}_x - f_x| \geq eF/l - E(n_x)) \\ =&Prob(n_x - E(n_x) \leq -(eF/l - E(n_x))) + \\ &Prob(n_x - E(n_x) \geq eF/l - E(n_x)) \\ =&Prob(n_x \leq -(eF/l - 2E(n_x))) + Prob(n_x \geq eF/l). \end{aligned} \tag{13}$$

From (11) and (7), we know

$$E(n_x) \leq E(n_{i,x}) = (F - f_x)/l \leq F/l \tag{14}$$

Therefore,

$$\begin{aligned} &eF/l - 2E(n_x) > eF/l - 2F/l > 0. \\ &Prob(n_x \leq -(eF/l - 2E(n_x))) \leq Prob(n_x < 0) = 0. \end{aligned} \tag{15}$$

The last equation holds since $n_x \geq 0$. Substituting (15) to (13),

$$\begin{aligned} &Prob(|\hat{f}_x - f_x| \geq eF/l - E(n_x)) \\ =&Prob(n_x \leq -(eF/l - 2E(n_x))) + Prob(n_x \geq eF/l) \\ =&Prob(n_x \geq eF/l) \end{aligned} \tag{16}$$

Since $n_{i,x}$ are all independent, from (11), we know

$$\begin{aligned} &Prob(|\hat{f}_x - f_x| \geq \frac{eF}{l} - E(n_x)) = Prob(n_x \geq \frac{eF}{l}) \\ =&Prob(\forall n_{i,x} \geq \frac{eF}{l} | 0 \leq i < d) = (Prob(n_{i,x} \geq \frac{eF}{l}))^d \end{aligned} \tag{17}$$

According to Markov's inequality,

$$Prob(n_{i,x} \geq eF/l) \leq E(n_{i,x})l/eF \tag{18}$$

Applying (7) to (18),

$$Prob(n_{i,x} \geq \frac{eF}{l}) \leq \frac{E(n_{i,x})l}{eF} = \frac{(F - f_x)}{eF} \leq \frac{1}{e} \tag{19}$$

Substituting (19) to (17)

$$\begin{aligned} &Prob(|\hat{f}_x - f_x| \geq eF/l - E(n_x)) \\ =&(Prob(n_{i,x} \geq eF/l))^d \leq 1/e^d \end{aligned} \tag{20}$$

With the same data structure, the error bound for CM [11] is

$$Prob(|\hat{f}_x - f_x| \geq eF/l) \leq 1/e^d \tag{21}$$

Comparing (21) with (12), we know that by removing the noise measured by MN, the error bound of estimation is at least same, if not tighter, than that of CM.

Theorem 3: Let n be the d-smallest noise we estimate using MN, and n' be the d-smallest noise that we measured using MN-O. If we update the noise every time after recording α real items, the expectation of the bias is bounded by

$$E(|n - n'|) \leq \alpha(1 + m)/2l \tag{22}$$

, where l is length of each counter array.

Proof 6.4: In MN-O, the noise for the fake items are not always update-to-date. The noise for each item will be updated every time after we record $m\alpha$ real items. Considering a fake item $X[t]$, suppose its noise are measured β_t items before, e.g., it misses β_t items. Since all the items are recorded randomly in one counter array, we have

$$E(n_{i,X[t]} - n'_{i,X[t]}) = \beta_t/l, 0 \leq i < d \tag{23}$$

where $n_{i,X[t]}$ is noise for $X[t]$ in ith array now and $n'_{i,X[t]}$ is the value before recording β_t items. Therefore,

$$\begin{aligned} &E(n_{X[t]} - n'_{X[t]}) \\ =&E(\min\{n_{i,X[t]}, 0 \leq i < d\} - \min\{n'_{i,X[t]}, 0 \leq i < d\}) \\ \leq&E(n_{i,X[t]} - n'_{i,X[t]}) = \beta_t/l \end{aligned} \tag{24}$$

Substituting (24) to

$$\begin{aligned} &E(|n - n'|) = E(n - n') = E(\frac{1}{m}\sum_{t=0}^{m-1} n_{X[t]} - \frac{1}{m}\sum_{t=0}^{m-1} n'_{X[t]}) \\ =&\frac{1}{m}\sum_{t=0}^{m-1} E(n_{X[t]} - n'_{X[t]}) \end{aligned} \tag{25}$$

, we have

$$E(|n - n'|) \leq \frac{1}{ml}\sum\nolimits_{t=0}^{m-1} \beta_t \tag{26}$$

We update the noise for m fake items every time after we record α real items in order. Therefore, in the worst case, the noises are measured before recording $\alpha, 2\alpha ... n\alpha$ real items.

$$\sum\nolimits_{t=0}^{m-1} \beta_t \leq \sum\nolimits_{t=1}^{m} t\alpha = \frac{m(m+1)\alpha}{2l} \tag{27}$$

Applying (27) to (26),

$$E(|n - n'|) \leq \frac{1}{ml}\frac{m(m+1)\alpha}{2l} = \frac{\alpha(1+m)}{2l} \tag{28}$$

Corollary 2: Let n be the d-smallest noise we estimate using in MN, and n' be the d-smallest noise that we measured using MN-O. If we update the noise every time after recording α real items, $E(|n - n'|) \leq 1$ when

$$\alpha(1 + m) \leq 2l \tag{29}$$

Proof 6.5: According to Theorem 3, when we set $\alpha(1+m) \leq 2l$, we have $E(|n - n'|) \leq 1$.

B. Analytical Results for MN-AI and MN-O-AI

Although as we discuss in Section III-E, of d-smallest noise of x is dependent on f_x, we can still have similar analytical result for MN-AI/MN-O-AI under some strong assumptions.

Theorem 4: Consider an arbitrary item x with a certain frequency f. Assume we know the actual value of expectation of d-smallest noise for items with frequency f (n_f), i.e. $E(n_f)$, which can be measured by MN-AI. We remove it to get frequency estimate of item x (denoted as $\hat{f}_x$) in CU. The bias of $\hat{f}_x$ can be bounded as

$$Prob(|\hat{f}_x - f| \geq eF/l - E(n_f)) \leq 1/e^d, \tag{30}$$

where F is the sum of frequencies of all data items, d, l are parameters in CU and e is Euler's number.

Proof 6.6: Similar to Proof 6.3, we can prove it.

Corollary 3: Let n_i be the d-smallest noise for R_i we estimate using MN-AI during the online measurement, and n'_i

be the d-smallest noise for R_i that we measure using MN-O-AI. If we update the noise every time after recording α real items, $E(|n_i - n_i'|) \leq 1$ when

$$\alpha(1+m) \leq 2l \tag{31}$$

Proof 6.7: In MN-O-AI, the noise is no more than MN-O. Therefore, with a similar proof as Proof 6.5, we have

$$E(n_i - n_i') \leq \alpha(1+m)/2l \tag{32}$$

when we set $\alpha(1+m) \leq 2l$, we have $E(|n_i - n_i'|) \leq 1$.

VII. Evaluation

We evaluate the performance, in terms of accuracy and throughput, of the proposed work on CPU, GPU and FPGA. We measure noise using MN, MN-O, MN-AI and MN-O-AI, and then remove the measured noise from the frequency estimates by CM/CU to produce the final results. For convenience, in our evaluation, we use the abbreviation of a noise measurement method (e.g., MN) to also denote the algorithm of using the method to measure noise and then subtract the noise from the estimates produced by CM (or CU). We compare them with the prior art, including the min-error method (using CM [11] and CU [12] as benchmarks), the error canceling method (using CS [13] as benchmark), and the error reduction method (using CMM [14] as benchmark). There are numerous other algorithms that are based on them [17], [18], [19], [20], [21], [13], [1], [15] but do not provide better performance in our context of comparison focused on fundamentals.

A. Implementation

We implement CM, MN , MN-O, CU, MN-AI, MN-O-AI, CMM and CS on CPU, GPU and FPGA. (1) CPU Implementation: The algorithms are implemented on a computer with Intel Core i7-8700 3.2GHz CPU and 16GB memory; (2) GPU Implementation: We use the CUDA toolkit [22] to implement the proposed algorithms so that they can run on GPU. Experiments are done on an NVDIA GeForce GTX 1070 GPU with 8GB GDDR5 memory and 1920 CUDA cores at a clock rate of 1506-1683 MHz; (3) FPGA Implementation: We implement the algorithms on a XILINX Nexys A7-100T development board, with 15850 logic slices, 4860Kbits Block RAM, and a clock rate of 100MHz.

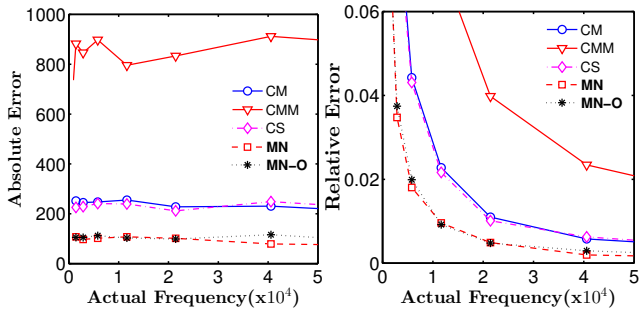
B. Experimental Setting

1) Data Stream: The data streams we use in this paper are network traffic traces from CAIDA [23], each of which contains around 20M packets. In all the experiments, we treat the packets with same source and destination IP addresses as same data item. In each traffic trace, there are around 450K different data items. The largest frequency is over 130K and the smallest frequency is 1.

2) Parameter Setting: The total memory we use for accuracy comparison is 1024Kb. The number of counter arrays in data structures of all algorithms is 4 and each counter is 20 bits long. We also change memory size to 256Kb, 512Kb, 1024Kb and 2048Kb to see how these algorithms perform under different memory limitations. For MN-O, we set $\alpha = 9$ and $m = \frac{l}{\alpha}$, which makes Corollary 2 stand. For MN-AI/MN-O-AI, we set $k = 10$ and $p_i, 0 \leq i \leq k-1$ as $\frac{1}{2^{15}}, \frac{1}{2^{16}}, \frac{1}{2^{17}}, \frac{1}{2^{18}}, ..., \frac{1}{2^{24}}$. In addition, for MN-O-AI, we set $\alpha = 90$ and $m = \frac{l}{\alpha}$ which also makes Corollary 3 stand for each range.

3) Performance Metrics: We compare all algorithms over estimation accuracy and throughput. Estimation accuracy includes relative errors and absolute errors. Throughput includes online recording throughput and online querying throughput where the first one is average number of data items recorded per second and the second one is average number of queries that can be answered per second online. The unit of throughput is Mdps, which means million data items per second.

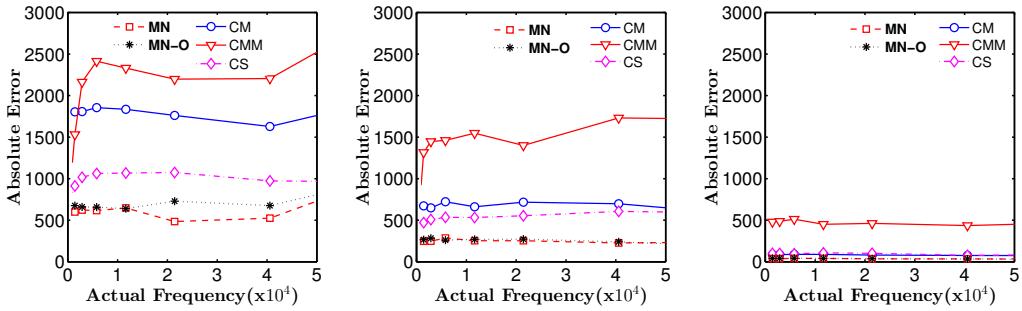
C. Accuracy Comparison



(a) Absolute error (b) Relative error

Fig. 1: Estimation accuracy w.r.t actual frequency when memory used is 1024Kb. In plot (a), for the bins of (32768,65536]/(8192,16384]/(1024,2048], the average absolute errors of MN is 34.2%/42.1%/42.8% of CM, 8.6%/13.5%/12.2% of CMM or 31.7%/44.8%/47.8% of CS.

1) Comparison among CM, CMM, MN and MN-O : Fig. 1 (a) shows that the average absolute errors of MN and MN-O are much lower than CM and CMM. For the bin of (32768,65536]/(8192,16384]/(1024,2048], the average absolute error of MN is 34.2%/42.1%/42.8% of CM, 8.6%/13.5%/12.2% of CMM or 31.7%/44.8%/47.8% of CS; the average absolute error of MN-O is 50.1%/40.1%/41.7% of CM, 12.7%/12.8%/11.9% of CMM or 46.4%/42.7%/46.5% of CS. The average relative errors of all algorithms reduce rapidly when actual frequency increases. MN and MN-O have similar absolute error and relative error, which is consistent with what we state in Corollary 2. Fig. 2 compares their estimation accuracy under different memory allocations. As expected, the result shows that the estimation errors of these four algorithms decrease when memory grows because smaller memory increases the number of data items that share a typical counter. MN/MN-O are always much more accurate than the other three algorithms. Taking the bin of (8192,16384] as an example, the average absolute error of MN is 35.3%/38.1%/45.4% of CM, 27.7%/16.3%/8.9% of CMM or 60.6%/47.4%/37.7% of CS,



(a) 256Kb (b) 512Kb (c) 2048Kb

Fig. 2: Average absolute error w.r.t memory used. For the bins of (32768,65536]/(8192,16384]/(1024,2048], the average absolute errors of MN is 35.3%/38.1%/45.4% of CM, 27.7%/16.3%/8.9% of CMM or 60.6%/47.4%/37.7% of CS, when memory used is 256/512/2048Kb.

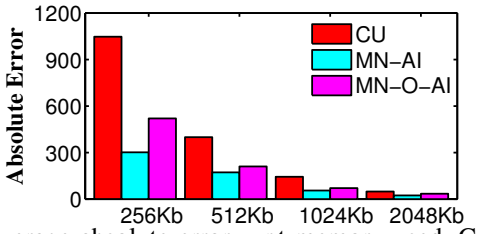


Fig. 3: Average absolute error w.r.t memory used. Compared to CU, MN-AI reduces average absolute error by 52%-73%.

Platform	CM/MN	MN-O	CU	MN-AI	MN-O-AI	CMM	CS
FPGA	100	90.0	25.0	24.8	22.5	100.0	100.0
GPU	277.1	249.3	NA	NA	NA	277.1	230.2
CPU	11.85	10.66	12.25	11.95	10.75	11.85	9.83

TABLE II: Recording throughput comparison.

when memory used is 256/512/2048Kb; the average absolute error of MN-O is 34.8%/40.6%/44.7% of CM, 27.4%/17.4%/8.8% of CMM or 59.8%/50.4%/37.2% of CS, when memory used is 256/512/2048Kb.

2) Comparison among CU, MN-AI and MN-O-AI: Fig. 3 compares CU, CS, MN-AI and MN-O-AI on average absolute error under different memory. Increasing memory can reduce average absolute errors for all algorithms. Compared to CU, MN-AI reduces average absolute error by 52%-73%. MN-O-AI is slightly worse than MN-AI which reduces average absolute error by 30%-50% for different memory allocations compared to CU. Moreover, the improvement will be much more significant under small memory allocations. This means MN-AI/MN-O-AI can largely improve the accuracy performance of CU under different memory limitations.

D. Throughput Comparison on Different Implementations

1) Online Recording Throughput Comparison: We first compare the online recording throughputs of all algorithms on GPU and FPGA implementations. From Table II, we know that CM and MN have the highest throughputs which means they are suitable for hardware implementation. In the meanwhile, MN-O is slightly slower than MN due to online noise measurement operations but is still suitable for hardware implementation. In contrast, CU/MN-AI/MN-O-AI are not suitable for hardware implementation. The reason is that when we record a data item through CU/MN-AI/MN-O-AI, we need to find the current minimum value in all d counters. Thus, it can not be fully pipelined for FPGA implementation. As a result, CU/MN-AI/MN-O-AI averagely need 4 clock cycles to process one data item while the other solutions only need 1 clock cycle per data item. The problem is even more severe for GPU implementation. Since in GPU implementation, we process thousands of packets simultaneously, we cannot get the true minimum value for each data item. The value may be influenced by the recording of other data items due to memory access collisions. CUDA does not have such complex atomic functions to avoid this kind of collisions which makes it impossible for us to realize CU/MN-AI/MN-O-AI algorithms.

Platform	CM	MN-O	CU	MN-O-AI	CMM	CS
FPGA	25.0	25.0	25.0	25.0	25.0	25.0
GPU	270.3	265.2	NA	NA	263.5	230.2
CPU	11.62	11.58	11.62	11.03	11.30	10.02

TABLE III: Online querying throughput comparison.

We also compare the online recording throughputs of all algorithms on CPU implementations. Since MN adds no extra processing overhead while recording data items, it will not affect the recording throughput of original algorithm which is consistent with our experimental results. In the experimental results shown in Table II, CM/MN have a throughput around 11.85 Mpps. Besides, the extra processing overhead of MN-AI is very small because the sampling percentages $p_i, 0 \leq i \leq k-1$ are all very small. From Table II we can see that MN-AI has a throughput of 11.95 Mdps, which is close to CU's 12.25 Mdps. CS has the lowest throughput of 9.83Mdps.

2) Online Querying Throughput Comparison: We compare the online querying throughput of all algorithms on different implementations. According to Table III, all algorithms excluding MN and MN-AI have a throughput around 25Mdps on FPGA Impelmentation. For GPU implementation, CM, MN-O and CMM have a similar throughput about 270Mdps which is higher than CS's 230Mdps. As we discussed before, the online querying throughputs of MN/MN-AI are very low and they are not designed for supporting online queries. Therefore we do not include them in the comparison. As Table III shows, for online querying on CPU implementations, CM/MN-O/CU are the fastest ones. The throughputs of them are all around 11.6Mdps. The online querying throughputs of MN-O-AI and CMM are 11.03Mdps and 11.30Mdps, respectively. CS has a slightly slower throughput of 10.02Mdps.

Generally speaking, MN-O is less accurate than MN and MN-O-AI is less accurate than MN-AI. However, MN-O/MN-O-AI can support fast online querying while MN/MN-AI cannot. Besides, MN and MN-O are suitable for hardware

implementation while the other two are not. All four algorithms have their pros and cons.

VIII. Related Work

Much prior art [1], [15], [5], [7], [6], [9], [2] uses CM [11] and CU [12] as the building blocks for their algorithms and applications. Below we briefly describe a number of them. OpenSketch [1] and cSketches [15] replace the counters in CM with other data structures such as bitmaps [24] for flow spread estimation in network monitoring. CMH [5] originates with a similar idea and an additional min-heap for super-spreader identification, which has important applications in detecting worms [25] and DDos attacks [4]. Elastic Sketch [6] uses CM or CU for the so-called light part in its system for heavy hitter identification and heavy changer detection. Univmon [2] provides a general framework of measuring data stream statistics, with CS, CM or CU as its core. Using CM for key-value storage in cache, Netcache [9] provides high aggregate throughput and low latency under highly-skewed and rapidly-changing workloads. NitroSketch [7] adds a sampling module to CM, trading higher throughput for lower accuracy.

There are other designs for item frequency estimation that are different from CM (or CU). Random Counter Sharing [18], Counter Tree [19], and Virtual Active Counters (VAC) [21] use different structures to ensure that only one counter update is needed for all (or most) data items, but their estimate accuracies are significantly worse than CM and CU. CMM [14] and CS [13] use different method control methods which we have introduced in Section III-C. Unfortunately, the average errors of estimates from them have been shown to be even higher than that of Count-Min according to [6].

IX. Conclusion

This paper addresses the long-standing problem of noise measurement that always ignored in min-error method such as Count-Min and its variants and proposes new techniques to solve it including d-*smallest noise* and *artificial data items*. Based on the techniques, we have designed noise measurement methods, called the d-smallest mean noise measurement, the d-smallest mean noise online measurement, the d-smallest mean noise measurement with artificial items, and the d-smallest mean noise online measurement with artificial items, respectively. We use them measure the noise in the estimates produced by min-error methods, and we can subtract it from the original estimates for better accuracy. We have implemented the proposed work on CPU/GPU/FPGA. Experiments based on real-world network packet streams show that MN/MN-O and MN-AI/MN-O-AI outperform Count-Min and Min-Counter Update, respectively, in terms of estimation accuracy.

Acknowledgments

This work is funded by NSF grant CNS-1719222, CNS-1718708 and STC-1562485. We thank the Xilinx University Program for hardware/software donation.

References

[1] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[2] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman†, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," *Proc. of ACM Sigcomm*, 2016.

[3] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized Error Removal for Online Spread Estimation in Data Streaming," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, pp. 1040–1052, 2021.

[4] K. Park and H. Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," *Proc. of ACM SIGCOMM'2001*, August 2001.

[5] G. Cormode and S. Muthukrishnan, "Space Efficient Mining of Multigraph Streams," *Proc. of ACM PODS*, June 2005.

[6] 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," *Proc. of ACM SIGCOMM*, August 2018.

[7] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and General Sketch-based Monitoring in Software Switches," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 334–350.

[8] H. Zhao, A. Lall, M. Ogihara, and J. Xu, "Global Iceberg Detection Over Distributed Data Streams," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 2010, pp. 557–568.

[9] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing Key-value Stores with Fast In-network Caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 121–136.

[10] A. Vakali, M. Giatsoglou, and S. Antaris, "Social Networking Trends and Dynamics Detection Via A Cloud-based Framework Design," in *Proceedings of the 21st International Conference on World Wide Web*, 2012, pp. 1213–1220.

[11] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: the Count-Min Sketch and Its Applications," *Proc. of LATIN*, 2004.

[12] A. Goyal, H. Daumé III, and G. Cormode, "Sketch Algorithms for Estimating Point Queries in NLP," in *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, 2012, pp. 1093–1103.

[13] M. Charikar, K. Chen, and M. Farach-Colton, "Finding Frequent Items in Data Streams," *Proc. of International Colloquium on Automata, Languages, and Programming (ICALP)*, July 2002.

[14] F. Deng and D. Rafiei, "New Estimation Algorithms for Streaming Data: Count-Min Can Do More," 2007.

[15] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized Sketch Families for Network Traffic Measurement," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019.

[16] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *Proc. of ACM SIGCOMM*, August 2002.

[17] S. Cohen and Y. Matias, "Spectral Bloom Filters," *Proc. of ACM SIGMOD*, June 2003.

[18] T. Li, S. Chen, and Y. Ling, "Fast and Compact Per-Flow Traffic Measurement through Randomized Counter Sharing," *IEEE INFOCOM*, 2011.

[19] M. Chen and S. Chen, "Counter Tree: A Scalable Counter Architecture for Per-Flow Traffic Measurement," *Proc. of IEEE ICNP*, November 2015.

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

[21] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Highly Compact Virtual Active Counters for Per-flow Traffic Measurement," 04 2018, pp. 1–9.

[22] Nvidia, "Nvidia Cuda C Programming Guide, Version 10.0," Online. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[23] UCSD, "CAIDA UCSD Anonymized 2015 Internet Traces on Jan. 17," http://www.caida.org/data/passive/passive_2015_dataset.xml, 2015.

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

[25] S. Chen and Y. Tang, "Slowing Down Internet Worms," *Proc. of IEEE ICDCS'04*, March 2004.