Supporting Real-Time T-Queries on Network Traffic with A Cloud-based Offloading Model

Yuanda Wang[§], Haibo Wang[§], Chaoyi Ma, Shigang Chen, Ye Xia
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida
yuandawang@ufl.edu, wanghaibo@ufl.edu, ch.ma@ufl.edu, sgchen@cise.ufl.edu, yx1@cise.ufl.edu

***Abstract*—Traffic measurement provides fundamental statistics for network management functions. To implement the measurement modules on the data plane for real-time query response, modern sketches are designed to work with limited on-die memory allocation from network processors and collect traffic statistics in epochs of a preset length. To handle real-time queries at arbitrary times over traffic in a preceding period T (called T-queries), the prior art sets the epoch length to $\frac{T}{n}$ and keeps the measurement results in a window of $n-1$ past epochs to support approximate T-queries. Such an approach however drastically increases the memory cost or decreases the accuracy in the query results if the memory allocation is fixed. In this paper, motivated by the concept of offloading in today's edge-cloud computing, we propose a collaborative edge-center traffic measurement model, where the traffic measurement modules at all network devices form the edge, which offloads the traffic measurement results to a measurement center possibly hosted in a datacenter. The center synthesizes the measurements from the past epochs and sends the aggregate results back to the measurement modules to support T-queries. We conduct experiments using real traffic traces to evaluate the performance of the proposed edge-center measurement model. The experimental results demonstrate that the proposed designs significantly outperform the prior art.**

***Index Terms*—Traffic measurement, offloading model, T-query, real-time**

I. Introduction

Traffic measurement modules such as Netflow [1] and sketch-based solutions [2]–[6] on routers and other network devices provide critical information for traffic engineering, anomaly detection and other network functions. There are numerous applications that can benefit from detailed real-time traffic statistics — for a few examples, identifying abnormal increase in traffic volume to catch denial-of-service attacks [7]–[11], detecting worm propagation or scanning attacks [12]–[14], and profiling potential botnet activities [15], [16].

To support real-time responses, it is highly desired that the traffic measurement modules are implemented on the data plane, examining the packet stream directly on the network processors. However, this means that they have to compete for the on-die processing and memory (such as SRAM) resources of network processors or other special-purpose chips that operate at line rates for the key networking functions of packet forwarding, queuing and scheduling, quality of service, firewalling, and packet inspection/classifiction for performance and security purposes. In this case, the design of a traffic measurement module must assume limited processing/memory allocation and can only record packet information for a limited time before its allocated memory space is saturated. At that time, the existing information has to be purged in order to free space for recording new packets that continuously arrive. Much of the prior art [6], [17], [17]–[21] focuses on highly compact data structures called sketches that measure a packet stream for a preset period of time called epoch. The measurements for each epoch are stored offline, and queries can be made on flow statistics in each epoch.

This paper is interested in support real-time queries on the data plane at an arbitrary time t for an arbitrary flow f about its traffic statistics in a preceding period of T before time t. This is called a T-query. One may argue that we may simply use an existing sketch to measure traffic in an epoch of $(t-T,t]$. However, because t can be arbitrary, there is no way to determine the beginning time of each epoch before queries happen. One naive solution is to support approximate T-queries with preset epochs, each of length T. When a query arrives in the middle of an epoch, we return the statistics of the flow that have been recorded so far in the current epoch. The query result will however vary greatly, covering traffic in a preceding period between 0 (when the query arrives at the beginning of the epoch) and T (when the query arrives at the end of the epoch).

Most prior art [22]–[24] addresses the above problem by using smaller epochs of length $\frac{T}{n}$. They require a measurement module to keep all measurements from a window of the most recent $n-1$ completed epochs, in addition to the current epoch. The answer to a query at time t on flow f combines the statistics of the flow from the previous $n-1$ epoch and the current epoch. The query result covers the traffic in a preceding period of $[t_0-\frac{n-1}{n}T,t)$, where t_0 is the beginning time of the current period and t is the query time. The difference between this period's length and the length of $(t-T,t]$ under query is $(t_0-\frac{n-1}{n}T)-(t-T)=\frac{T}{n}-(t-t_0)\leq\frac{T}{n}$, which can be made arbitrarily small when we increase n.

The problem of the above solution is that the measurement module now has to keep the traffic statistics of the past $n-1$ epochs, which is an undesired memory cost on the data plane, which increases with n. One may argue that the memory cost for each epoch will be smaller due to a smaller epoch length. However, the impact of a smaller epoch on memory is actually not significant. Consider using a counter to measure

[§]Equal contribution

the number of packets in each flow. Suppose that we use a 32-bit counter for an epoch of length T. Let $n = 32$. It will take a 27-bit counter for an epoch of length $\frac{T}{32}$ if we make the counter range proportional to the epoch length. The total memory for the flow will be $27 \times 32 = 864$ bits! Moreover, it is questionable whether we should reduce the counter size from 32 bits to 27 bits because network traffic is often bursty and it could happen that most packets of a flow over a period of T happens in one epoch of $\frac{T}{n}$.

The goal of this paper is to achieve the same approximation in the query result as the window approach does, while avoiding its pitfall of memory cost on the data plane. Our idea is motivated by the concept of offloading in today's edge-cloud computing. We introduce a collaborative edge-center traffic measurement model that consists of a *measurement edge* and a *measurement center*. The edge consists of all devices in a network that implement the measurement modules, and the center may be hosted in a data center that provides virtually unlimited resources for long-term storage/analysis of traffic measurements and assistance in coordinating the measurement activities across all devices in the network. Each measurement module at the edge offloads its traffic measurements of the prior $n-1$ epochs to the center, which in turn feeds aggregate measurements back to assist the measurement module in answering real-time queries. Timeliness and memory efficiency require coordination and data exchange between the edge and the center, representing a three-way tradeoff between memory cost, communication cost and query accuracy. Our main contributions are listed as follows.

- This is the first work that employs an offloading model to relieve the increased memory cost in answering T-queries in real time.
- We present two general designs that exploit offloading in measuring flow size and flow spread, respectively.
- We conduct experiments using real traffic traces to evaluate the performance of the proposed solutions. The experimental results demonstrate that our solution significantly outperforms the prior art, either reducing the memory cost or equivalently improving the accuracy in query results under the same memory allocation.

The rest of the paper is organized as follows. Section II describes the system and flow models and then states the problem studied in this paper. Our solutions and the performance analysis are given in Section III. The experimental evaluation is presented in Section IV. Section V reviews related work. Finally, we conclude the paper in Section VI.

II. Preliminaries

A. System Model

We introduce a collaborative edge-center traffic measurement model that consists of a logical measurement edge and a measurement center. The logical measurement edge is composed of all routers, switches, IDSes (introduction detection systems), firewalls, and other network devices that are equipped with a traffic measurement module, which examines the arrival packet stream and records the traffic statistics. The measurement center may be hosted in a data center that has computing/storage resources for long-term data storage and short-term coordination with the numerous measuring devices at the edge.

Time is divided into epochs, whose length is preset and fixed. The measurement modules offload their measurement data after each epoch to the center. The center records such data, synthesize them over multiple epochs or over multiple devices, and send aggregate data back to the measurement modules of the devices at the edge, so that they do not have to spend their limited on-die memory in keeping detail information in order to support real-time queries from local network security/performance functions such as identifying large-sized flows (heavy hitters) for traffic shaping or identifying large-spread flows (super spreaders) for anomaly detection. As the edge does not keep detailed traffic measurements over multiple epochs, each measurement module will answer real-time queries based on both the local measurement data and the aggregate data provided from the center.

B. Flow Model

The network traffic at a measurement module is a packet stream which consists of a continuous sequence of packets, where f is a flow label and e is an element identifier. The flow label f is typically a combination of selected packet-header fields, which may include source address, destination address, protocol identifier, and/or other fields in the packet headers, depending on the type of measurement functions and application requirements. The element e can also be one or multiple header fields or certain content in packet payload. All packets with the same flow label form a flow. With different flow labels, flows under measurement may be per-source flows, per-destination flows, TCP flows, WWW flows, etc. Based on the packet stream, there are various types of traffic statistics that are crucial for network applications. This paper focuses on two of them, i,e., flow size and flow spread, but our collaborative edge-center measurement model solutions may be extended for other types of measurement as well.

Flow size is defined as the number of elements in each flow, where elements may be packets, bytes, or occurrences of a certain header-field value. One may measure the number of SYM packets from each source address, the number of ACK packets sent to each address, the number of bytes that each host downloads, or the number of bytes in each source-destination flow. Such information is very useful to anomaly detection, capacity planning, flow rerouting. For instance, measuring the number of SYN/ACK packets provides a means for detecting SYN attacks [25].

Flow spread is defined as the number of distinct elements in each flow. Consider per-destination flows, where all packets sent to a common destination constitute a flow. If the number of distinct source addresses in a flow suddenly surges, it may signal a DDoS attack. An institutional gateway may determine the popularity of external web content for caching priority by tracking the number of distinct source addresses in each flow

(which consists of all outbound HTTP requests for the same web content).

C. Problem Statement

The T-query can be made by a network function or a user to a traffic measurement module at any time t for a flow f's real-time statistic, e.g., flow size and flow spread, in the period of $(t-T,t]$, which is the period of length T immediately preceding time t, where T is a preset parameter. Exact implementation of T-query is very difficult because it requires the measurement module to track the exact traffic in $(t-T,t]$. As time goes from t to $t+\Delta t$, not only do we have to record new information from $(t,t+\Delta t]$, but also we have to remove the old information in $(t-T,t-T+\Delta t]$, for any time increment Δt, which is hard to implement efficiently without storing each packet $\langle f,e\rangle$ with an arrival timestamp.

In practice, traffic measurement is done in each epoch. If we let T be n epochs (where integer $n(\geq 1)$ can be arbitrarily chosen), then the length of each epoch is $\frac{T}{n}$. We define approximate T-query at time t on flow f as estimating the size or spread of f in the current epoch up to t plus the previous $n-1$ epochs. The answer to the query is based on the traffic measured in a period of length between $(t-T+\frac{T}{n},t]$ and $(t-T,t]$, depending on where t is in the current epoch.

Below we give a couple of application examples of real-time T-queries. Consider the application of detecting SYN/ACK attacks and DDoS attacks discussed earlier. It is critical to detect them in real time, so that response can be made in real time, which in turns requires answering T-queries about flows of interest in real time. In another application of traffic shaping to deal with a congestion condition, we want to identify large flows in real time, so that the congestion can be resolved quickly.

III. Collaborative Edge-Center Traffic Measurement

We propose three sketch designs for collaborative edge-center traffic measurement, which allows the measurement modules at the edge to offload their measurement data to the center (in the cloud), so as to reduce the on-die memory requirement at the edge where each measurement module must record the arrival packets at the line rates and answer T-queries in real time. *Sketch* refers to a compact data structure that records information from a large data stream, in our case, a large packet stream arriving at a high rate. We discuss the tradeoff between query approximation and communication overhead that is enabled in our designs. Finally we formally analyze query accuracy.

A. One-Sketch Design for Flow Size

We describe our one-sketch design that uses a CountMin [26] sketch to measure per-flow size in each epoch of length $\frac{T}{n}$. The same design can be easily modified to work with other sketches that measure per-flow size in each epoch [4], [18], [27], [28]. Our contribution is to build a solution with any of these sketches to answer T-queries on flow size.

Algorithm 1: One-sketch design for flow size—edge side

Input: CountMin sketches C, time t, T, n
Action: packet recording, data offloading and query
if *packet x arrives* **then**
  record x to C using the recording operation in CountMin;
end
if *reach the end of current epoch* **then**
  send C to the measurement center;
  reset all counters in C to zero;
end
if *receive A* **then**
  for $i=0$ *to* $d-1$ **do**
    for $j=0$ *to* $m-1$ **do**
      $C[i][j]=C[i][j]+A[i][j]$;
    end
  end
end
if *query on flow f* **then**
  answer query on C in the way that CountMin does;
end

Algorithm 2: One-sketch design for flow size—center side

Input: CountMin sketches $B_0,B_1...B_{k-1}$, $C_0,C_1...C_{k-1}$, time t, T, n, k
Action: data combination
if *receive C_k* **then**
  if $k<n-1$ **then**
    for $i=0$ *to* $d-1$ **do**
      for $j=0$ *to* $m-1$ **do**
        $B_k[i][j]=C_k[i][j]-C_{k-1}[i][j]$;
      end
    end
  end
  else
    for $i=0$ *to* $d-1$ **do**
      for $j=0$ *to* $m-1$ **do**
        $B_k[i][j]=$
        $C[i][j]-\sum_{k-n+2\leq l\leq k-1}B_l[i][j]$;
        $A[i][j]=\sum_{k-n+2\leq l\leq k}B_l[i][j]$;
      end
    end
    send A_k to the measurement edge;
  end
  $k=k+1$;
end

We first review the CountMin sketch [26], which uses a two-dimensional array of counters, denoted as $C[i][j]$, $0 \leq i < d$ and $0 \leq j < m$. It has d rows, each of m counters. When receiving a packet from flow f, we record the packet by hashing f to one counter in each row and increasing that counter by one, i.e.,

$$C[i][H_i(f)] = C[i][H_i(f)] + 1,$$

where H_i, $0 \leq i < d$, are independent hash functions and the modulo operation is assumed to keep the hash output in the range $[0, m)$. When we query the size of flow f recorded in C, we take the minimum value of those d counters as the estimate answer $\hat{n}_f$ for the query, i.e.,

$$\hat{n}_f = \min\{C[i][H_i(f)], 0 \leq i < d\}.$$

It has been proved [26] that the estimate from CountMin is bounded by

$$Prob(|\hat{n}_f - n_f| \geq \frac{eN}{l}) \leq \frac{m}{e^d}, \tag{1}$$

where N is the total number of packets recorded by CountMin.

Next we present our design of using a single CountMin sketch C per measurement module to support T-queries. The module is configured to record packets epoch by epoch, each of length $\frac{T}{n}$. The counters in C are initialized to zeros at the beginning of the the 0th epoch. Starting from the $(n-1)$th epoch, their values will be reset to zeros at the beginning of each epoch.

At the end of the kth epoch, for all $k \geq 0$, the measurement module sends the content of C to the measurement center. We denoted the content of C as C_k after the kth epoch, where $k \geq 0$. The center stores the measurements from the kth epoch in a new two-dimensional array B_k as follows:

$$B_0[i][j] = C_0[i][j],\ 0 \leq i < d,\ 0 \leq j < m$$

$$B_k[i][j] = \begin{cases} C_k[i][j] - C_{k-1}[i][j], \\ \qquad k < n-1,\ 0 \leq i < d,\ 0 \leq j < m; \\ C_k[i][j] - \sum_{k-n+2 \leq l \leq k-1} B_l[i][j], \\ \qquad k \geq n-1,\ 0 \leq i < d,\ 0 \leq j < m. \end{cases} \tag{2}$$

Clearly, B_k keeps the measurements of the kth epoch, $\forall k \geq 0$.

When $k \geq n-1$, the center computes the aggregate measurements A_k from the $(k-n+2)$th epoch to the kth epoch, i.e., $\forall 0 \leq i < d,\ 0 \leq j < m$,

$$A_k[i][j] = \sum_{k-n+2 \leq l \leq k} B_l[i][j], \tag{3}$$

and sends A_k back to the measurement module, which will add A_k to its sketch array C as follows:

$$C[i][j] = C[i][j] + A_k[i][j],\ 0 \leq i < d,\ 0 \leq j < m.$$

T-queries are allowed starting from the $(n-1)$th epoch. When being queried on flow f, the module returns $\min\{C[i][H_i(f)], 0 \leq i < d\}$. The pseudo codes for the one-sketch design on the edge side and center side are given in Algorithms 1 and 2.

In our edge-center module, we make the edge light-weighted to save memory resource by only keeping the measurement of the current epoch and relieving it from storing the detailed information about earlier epochs, which is stored in the center. The center will refresh the edge with the aggregate information of the earlier $n-1$ epochs, i.e., A_k. The problem of the one-sketch design is that queries cannot be made right after the end of the kth epoch and before the module receives A_k, which is a round-trip delay between the module and the center. This may not be a serious issue if the center collocates with the measurement module and the delay is negligible. But if the center is distant in a datacenter, serving for many measurement modules scattered in a large network, the delay may become non-negligible. To address this problem, we design a two-sketch mechanism for flow-size below, which doubles the memory requirement at each measurement module.

B. Two-Sketch Design for Flow Size

Algorithm 3: Two-sketch design for flow size—edge side

```
Input: CountMin sketches C and C', time t, T, n
Action: packet recording, data offloading and query
if packet x arrives then
    record x to C and C' using the recording
      operation in CountMin;
end
if reach the end of current epoch then
    send C to the measurement center;
    C = C';
    reset all counters in C' to zero;
end
if receive A then
    for i = 0 to d − 1 do
        for j = 0 to m − 1 do
            C'[i][j] = C'[i][j] + A[i][j];
        end
    end
end
if query on flow f then
    answer query on C in the way that CountMin does;
end
```

The new design uses two CountMin sketches, C and C' of the same dimensions. Initially, all counters in C and C' are set to zeros. In each epoch, all packets will be recorded in both C and C' in the same way as the one-sketch design does to C. At the end of the kth epoch, $\forall k \geq 0$, still denoting the content of C as C_k, the measurement module will send C_k to the center, copy C' to C, and reset C' to zeros.

At the end of the kth epoch with $k \geq n-1$, our design

Algorithm 4: Two-sketch design for flow size—center side

Input: CountMin sketches $B_0, B_1 ... B_{k-1}$, $C_0, C_1 ... C_{k-1}$, time t, T, n, k
Action: data combination
if *receive* C_k **then**
  if $k < n-2$ **then**
    for $i = 0$ *to* $d-1$ **do**
      for $j = 0$ *to* $m-1$ **do**
        $B_k[i][j] = C_k[i][j] - C_{k-1}[i][j]$;
      end
    end
  end
  else
    for $i = 0$ *to* $d-1$ **do**
      for $j = 0$ *to* $m-1$ **do**
        $B_k[i][j] = C_k[i][j] - \sum_{k-n+2 \le l \le k-1} B_l[i][j]$;
        $A[i][j] = \sum_{k-n+3 \le l \le k} B_l[i][j]$;
      end
    end
    send A_k to the measurement edge;
  end
  $k = k+1$;
end

ensures that, $\forall 0 \le i < d,\ 0 \le j < m$,

$$C[i][j] = \sum_{k-n+1 \le l \le k} B_l[i][j] \tag{4}$$

$$C'[i][j] = \sum_{k-n+2 \le l \le k} B_l[i][j], \tag{5}$$

where B_l is the measurements for the lth epoch, as is defined in Section III-A. Hence, C contains the measurements for n epochs, from the $(k-n+1)$th epoch to the kth epoch. C' contains the measurements for $n-1$ epochs, from the $(k-n+2)$th epoch to the kth epoch.

After sending the content of C to the center, copying C' to C and setting C' to zeros, the measurement module will have

$$C[i][j] = \sum_{k-n+2 \le l \le k} B_l[i][j] \tag{6}$$

$$C'[i][j] = 0. \tag{7}$$

C will contain the measurements for $n-1$ epochs, from the $(k-n+2)$th epoch to the kth epoch, ready to support T-queries from the beginning of the $(k+1)$th epoch. The measurement module will now start the $(k+1)$th epoch by recording each packet in both C and C'.

When the center receives C_k, it first computes B_k and A_k as follows: $\forall 0 \le i < d,\ 0 \le j < m$,

$$B_0[i][j] = C_0[i][j],\ 0 \le i < d,\ 0 \le j < m$$

$$B_k[i][j] = \begin{cases} C_k[i][j] - C_{k-1}[i][j], \\ \qquad k < n-1,\ 0 \le i < d,\ 0 \le j < m; \\ C_k[i][j] - \sum_{k-n+2 \le l \le k-1} B_l[i][j], \\ \qquad k \ge n-1,\ 0 \le i < d,\ 0 \le j < m. \end{cases} \tag{8}$$

Starting from the $(n-2)$th epoch, it will compute

$$A_k[i][j] = \sum_{k-n+3 \le l \le k} B_l[i][j], \tag{9}$$

and send A_k to the measurement module, which will add A_k to C'_k as follows:

$$C'[i][j] = C'[i][j] + A_k[i][j],\ 0 \le i < d,\ 0 \le j < m.$$

This will ensure that at the end of the $(k+1)$th epoch, C' will contain the measurements of $n-1$ epochs, from the $((k+1)-n+2)$th epoch to the $(k+1)$th epoch, while C will contain the measurements for n epochs, from the $((k+1)-n+2)$th epoch to the $(k+1)$th epoch, which are both consistent with (5).

When being queried on flow f, the module returns $\min\{C[i][H_i(f)], 0 \le i < d\}$. The pseudo codes for the two-sketch design on the edge side and center side are given in Algorithms 3 and 4.

C. Three-Sketch Design for Flow Spread

We describe our three-sketch design that uses vSkt(HLL) [17] to measure per-flow spread in each epoch of length $\frac{T}{n}$. The same design can be easily modified to work with other sketches that measure per-flow spread [19], [29]. Our contribution is to build a solution on top of vSkt to answer T-queries on flow spread.

We first review the vSkt(HLL) [17]. Its data structure is a two-dimensional array of HLL registers [30], denoted as $C[i][j]$, $0 \le i < d$ and $0 \le j < m$. An HLL register is a counter of r bits which can store a value in the range of $[0, 2^r - 1]$. For example, if we assign each HLL register 5 bits, then its range is is $[0, 31]$. When receiving a packet $\langle f, e \rangle$, we record it by hashing $\langle f, e \rangle$ to one HLL register in the whole sketch and update the register value as follows :

$$C[H'(e)][H_{H'(e)}(f)] = \max\{C[H'(e)][H_{H'(e)}(f)], G(e)\} \tag{10}$$

where H' is a uniform hash function with an output in the range of $[0, d)$; H_i, $0 \le i < d$, are independent hash functions of range $[0, m)$; G is a geometric hash function whose value is v with probability 2^{-v} for $v \ge 1$. When querying, we first estimate the total number of distinct elements recorded by vSkt(HLL) as

$$\hat{N} = \alpha_d d^2 / (\sum_{i=0}^{d-1} 2^{-\max\{C[i][j], 0 \le j < m\}}),$$

Algorithm 5: Three-sketch design for flow spread—edge side

Input: vSkt(HLL) sketches B, C and C', time t, T, n
Action: packet recording, data offloading and query
if *packet x arrives* **then**
  record x to B, C and C' using the recording operation in vSkt(HLL);
end
if *reach the end of current epoch* **then**
  $C = C'$;
  reset all registers in C' to zero;
  send B to the measurement center;
  reset all registers in B to zero;
end
if *receive A* **then**
  for $i = 0$ *to* $d-1$ **do**
    for $j = 0$ *to* $m-1$ **do**
      $C'[i][j] = \max\{C'[i][j], A[i][j]\}$;
    end
  end
end
if *query on flow f* **then**
  answer query on C in the way that vSkt(HLL) does;
end

Algorithm 6: Three-sketch design for flow spread—center side

Input: vSkt(HLL) sketches for different epochs $B_0, B_1 ... B_k$, time t, T, n, k
Action: data combination
if *receive B_k* **then**
  if $k \geq n-2$ **then**
    for $i = 0$ *to* $d-1$ **do**
      for $j = 0$ *to* $m-1$ **do**
        $A[i][j] = \max\{B_l[i][j], k-n+3 \leq l \leq k\}$;
      end
    end
    send A to the measurement edge;
  end
  $k = k+1$;
end

and then we use the following function to estimate spread of flow f as $\hat{n}_f$:

$$\hat{n}_f = \alpha_d d^2 / (\sum_{i=0}^{d-1} 2^{-C_{[i][H_i(f)]}}) - \hat{N}/m,$$

where $\alpha_d = 0.7213/(1 + 1.078/d)$. It has been proved [17] that the expectation and variance of estimate from vSkt(HLL) satisfies

$$\begin{aligned} n_f(1 - \epsilon - \frac{1}{m}) \leq E(\hat{n}_f) \leq n_f(1 + \epsilon - \frac{1}{m}) \\ Var(\hat{n}_f) = \frac{1.04^2}{d}(k + \frac{N}{m})^2 + \frac{1.04^2}{dm^2}X^2 \\ + (\frac{1.04^2}{d} + 1)\frac{N}{d}(1 - \frac{1}{m}) \end{aligned} \tag{11}$$

where $\epsilon = \delta(1 - \frac{1}{m} + \frac{2N}{mn_f})$ with $\delta \leq 5 \times 10^{-5}$ when $d \geq 16$, and N is the total number of distinct elements recorded by vSkt(HLL).

Our design uses three vSkt(HLL) sketches, denoted as B, C and C', respectively. Each of them is an array of $d \times m$ HLL registers. At the end of each epoch, the registers in B are sent to the center and then they are reset to zeros, whereas C is replaced by C', and C' will wait for its new values from the center. Then, the next epoch starts, and the packets will be recorded in B, C and C', with B tracking the spread measurements in this epoch, C used to answer T-queries, and C' preparing for the next epoch. The reason for introducing a third sketch B to record the measurements of the current epoch and send it to the center is that the center cannot compute it based on (8) because C is not a counter array but an HLL register array with a max operation (10).

Let B_k be the flow-spread measurements in the kth epoch. At the end of the kth epoch with $k \geq n-1$, our design ensures that, $\forall 0 \leq i < d,\ 0 \leq j < m$,

$$C[i][j] = \max\{B_l[i][j], k-n+1 \leq l \leq k\} \tag{12}$$
$$C'[i][j] = \max\{B_l[i][j], k-n+2 \leq l \leq k\}, \tag{13}$$

where B_l is the spread measurements of the lth epoch. Similar to the two-sketch design, but here in terms of spread instead of size, C contains the measurements from the $(k-n+1)$th epoch to the kth epoch, and C' contains the measurements from the $(k-n+2)$th epoch to the kth epoch.

Denote C and C' at the end of the kth epoch as C_k and C'_k, respectively. The measurement module will send B (i.e., B_k) to the center, set it to zeros, copy C' to C, and set C' to zeros, such that, $\forall 0 \leq i < d,\ 0 \leq j < m$,

$$C[i][j] = \max\{B_l[i][j], k-n+2 \leq l \leq k\} \tag{14}$$
$$C'[i][j] = 0. \tag{15}$$

C will contain the measurements for $n-1$ epochs, from the $(k-n+2)$th epoch to the kth epoch, ready to support T-queries from the beginning of the $(k+1)$th epoch. The measurement module will now start the $(k+1)$th epoch by recording each packet in B, C and C'.

When the center receives B_k, if $k \geq n-2$, it first computes A_k as follows: $\forall 0 \leq i < d,\ 0 \leq j < m$,

$$A_k[i][j] = \max\{B_l[i][j], k-n+3 \leq l \leq k\} \tag{16}$$

and then sends A_k to the measurement module, which will merge A_k to C'_k as follows: $\forall 0 \leq i < d,\ 0 \leq j < m$,

$$C'[i][j] = \max\{C'[i][j], A_k[i][j]\}.$$

This will ensure that at the end of the $(k+1)$th epoch, C contains the measurements from the $((k+1)-n+1)$th epoch to the $(k+1)$th epoch, and C' contains the measurements from the $((k+1)-n+2)$th epoch to the $(k+1)$th epoch, which are consistent with (13).

Upon T-query, the measurement module will answer query on C. The pseudo codes for the three-sketch design on the edge side and center side are given in Algorithms 5 and 6, respectively.

D. Tradeoff between Approximation and Communications

Built on top of the offloading model, our designs require the measurement edge to periodically interact with the measurement center. Specifically, at the end of each kth epoch, the measurement edge will offload the measurement data (C_k for flow size measurement and B_k for flow spread measurement) and the measurement center will send the aggregate data back to the measurement edge, i.e., A_k for either flow size measurement or flow spread measurement. The epoch length is $\frac{T}{n}$ seconds and the memory for the sketch is denoted as M bits. So the average bandwidth required when dividing the time period T into n epochs is

$$\frac{2Mn}{T}bps, \tag{17}$$

where the coefficient is 2 because we combine the downlink and uplink bandwidth together. As we can see, with n increasing, the communication cost will increase as well.

While a smaller n is in favor for less communication cost, a larger n enables $[t_0 - \frac{n-1}{n}T, t]$ to approximate $[t-T, t]$ more, where t_0 is the start time of the current epoch with $t_0 \le t \le t_0 + \frac{T}{n}$. For any T-query casted at time t, we expect to collect the traffic statistics during the time period of $[t-T, t]$ exactly, which can only be done if we track the time stamp of each packet, implying tremendous resource consumption. Alternatively, our solutions actually collect the traffic statistics during the time period of $[t_0 - \frac{n-1}{n}T, t]$, an approximate substitute period to the period $[t-T, t]$. We can conclude that the difference between these two periods is within $|t_0 - \frac{n-1}{n}T - (t-T)| \le \frac{T}{n}$ which indicates that a larger n is preferred for a closer approximation.

E. T-query Accuracy

Consider an arbitrary time t of T-query on a flow's size or spread over a period of $[t_0 - \frac{n-1}{n}T, t]$, where t_0 is the beginning time of the current epoch. Our two-sketch (or three-sketch) design greatly reduces memory cost by measuring in each epoch of length $\frac{T}{n}$, while not keeping the detailed measurements of the earlier epochs locally. The question is whether its query result is as accurate as what will be produced from directly applying CountMin (or vSkt) to the period of $[t_0 - \frac{n-1}{n}T, t]$ — this is the ideal case but not implementable for all possible times t. The theorems below show that our designs produce the same query results as the ideal case.

Theorem 1. *For a T-query at an arbitrary time t on an arbitrary flow, the two-sketch design produces the same result as what CountMin with the same configuration of d, m and hash functions will produce when it measures the traffic in $[t_0 - \frac{n-1}{n}T, t]$.*

Proof: C represents the CountMin sketch in our design and let C^c be the CountMin sketch that records the packet stream during time period $[t_0 - \frac{n-1}{n}T, t]$. Consider any pair of counters that have the same location in C and C^c, i.e., $C[i][j]$ and $C^c[i][j]$. They will witness the same set of packets as the hash functions and the number of columns are the same. Since for the each packet arrival the counter will increase by 1, we can conclude that $C[i][j] = C^c[i][j]$. Since their query operations are the same, C and C^c will produce the same flow size estimate for any flow. The theorem stands. ■

Theorem 2. *For a T-query at an arbitrary time t on an arbitrary flow, the three-sketch design produces the same result as what vSkt with the same configuration of d, m and hash functions will produce when it measures the traffic in $[t_0 - \frac{n-1}{n}T, t]$.*

Proof: C represents the vSkt(HLL) sketch in our design from which we answer the spread query for any flow and let C^c be the vSkt(HLL) sketch that records the packet stream during time period $[t_0 - \frac{n-1}{n}T, t]$. Consider any pair of HLL registers in the same location of C and C^c, i.e., $C[i][j]$ and $C^c[i][j]$. They will witness the same set of packets as the hash functions and the number of columns are the same, and thus produce the same geometric hash value. Since both registers $C^c[i][j]$ and $C[i][j]$ store the maximum geometric hash value, we have $C[i][j] = C^c[i][j]$. Given that C and C^c answer the spread query in the same way, they will produce the same spread estimate for any flow. The theorem stands. ■

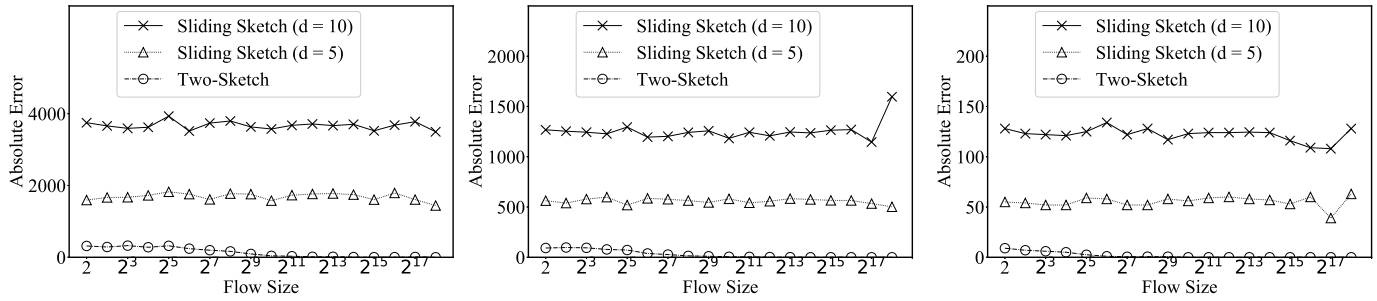
IV. Experiment and Evaluation

A. Experiment Setup

Dataset: We use the real traffic traces downloaded from CAIDA in 2018 [31]. The dataset we use lasts for 30 mins and contains 328524761 packets, 27710 different source addresses and 20791 different destination addresses.

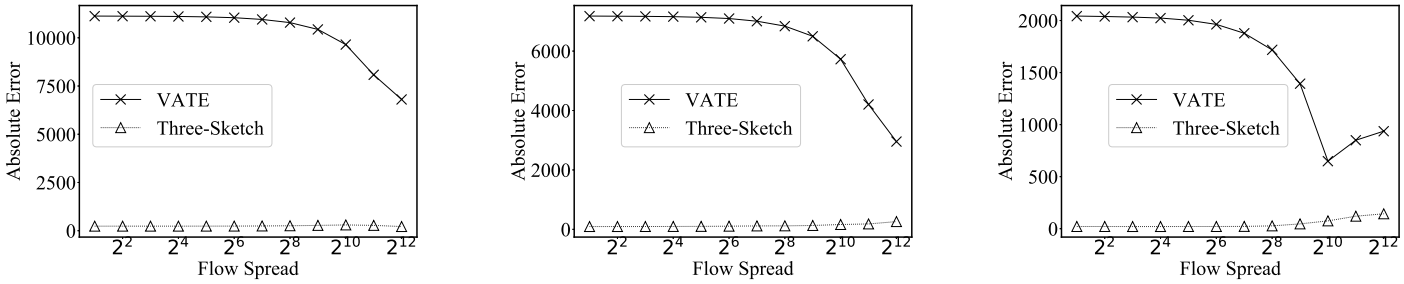
Flow size measurement: We consider the destination address as the flow label. That is, packets with the same destination address form a flow. In the CAIDA dataset, there are 20791 destination flows. T is set as 1 min and each epoch lasts for 6s, meaning that answering T query requires collecting the information of 10 consecutive epochs. We use CountMin to measure the flow size in each epoch, with the recommended setting of $d = 4$. We implement our two-sketch design in comparison with the state of the art, i.e., Sliding Sketch [22]. Sliding Sketch is based on CountMin. Its main data structure is a two-dimension array with its number of rows d equal to the number of epochs that the sliding window (T) contains. We implement two types of Sliding Sketches, one with $d = 10$ for fair comparison and the other with $d = 5$ to enhance our evaluation.

Flow spread measurement: The destination address is the flow label, which has the application of detection of DDoS



(a) 1Mb Memory (b) 2Mb Memory (c) 8Mb Memory

Fig. 1: Flow size measurement accuracy comparison of Sliding Sketch and the two-sketch design under memory allocations of 1Mb, 2Mb and 8Mb.



(a) 1Mb Memory (b) 2Mb Memory (c) 8Mb Memory

Fig. 2: Flow spread measurement accuracy comparison of VATE and the three-sketch design under memory allocations of 1Mb, 2Mb and 8Mb.

attacks [32]. T is set as 1 min and each epoch is 6s. vSkt(HLL) is used for flow spread measurement in our three-sketch design. Following the parameter setting in the original paper, we set $d = 128$. We use the classical solution, i.e., VATE [23], as the benchmark, which measures the spread of each flow based on the bitmap algorithm [5], [33], [34]. The bitmap length is the same as the original paper, i.e., 4096.

We use MacBook as the measurement edge, which has a Quad-Core Intel Core i7 (2.7Ghz), 16GB memory. The cloud center is HP Z840, which has an E5-2643v4 CPU (6-Core, 20M Cache, 3.4GHz), 256GB memory and disk space of total 9.6 TB.

In what follows, we present the experimental results using the performance metric of absolute error, defined as $|n_f - \hat{n}_f|$, where n_f is the real size/spread of f and $\hat{n}_f$ is the estimated size/spread of f.

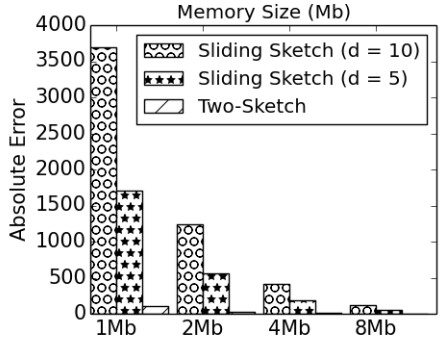


Fig. 3: Average absolute error of the two-sketch design in comparison with Sliding Sketch under different memory allocations for flow size measurement.

B. Experimental Results

Flow size measurement: Each algorithms are allocated the same amount of memory, which can be 1Mb, 2Mb, 4Mb and 8Mb. Fig. 3 shows the experimental results about the average absolute error of all the flows. Our solution reduces the average absolute error by up to 96.9% and 93.4% compared to Sliding Sketches with $d = 10$ and $d = 5$, respectively. Thanks to the offloading model, our solution only needs to store the information of the current epoch and accumulative information, while Sliding Sketch needs to store the information of recent T period locally. We also illustrate the absolute error distribution along flows with different sizes under the memory allocations of 1Mb, 2Mb and 8Mb, as shown in Figs. 1a, 1b

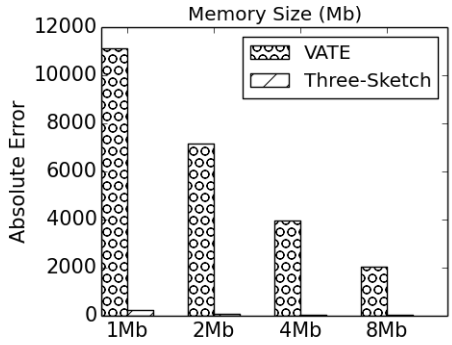


Fig. 4: Average absolute error of the three-sketch design in comparison with VATE under different memory allocations for flow spread measurement

and 1c, respectively. The figures demonstrate that our solution consistently outperforms Sliding Sketch, regardless of the flow size.

Flow spread measurement: Fig. 4 shows the absolute error with memory sizes of 1Mb, 2Mb, 4Mb and 8Mb. Thanks to the offloading model, our solution reduces the average absolute error by up to 97.8% compared to VATE. We also illustrate the absolute error distribution along flows with different sizes. The results under 1Mb, 2Mb and 8Mb memory allocations are shown in Figs. 2a, 2b and 2c, respectively. Although VATE reduces the absolute error for flows with large spreads, it is still consistently inferior to our solution, regardless of the flow spread.

V. Related Work

We review two categories of existing work on traffic measurement, one for the fix window and the other for the sliding window. For each categories, we separate them into flow size measurement and flow spread measurement.

A. Traffic Measurement for a Fixed Window

Flow size measurement: Flow size measurement for a single flow can be easily solved using counters. However, when there are numerous flows, it can incur tremendous memory overhead. Therefore, solution such as Netflow can only deal with a small subset of flows/packets, resulting in unsatisfactory quality of service. Most research has followed the path of designing efficient data structures to perform approximate estimation. Instead of using separate counters for individual flows, counting Bloom filter [35], CountMin [5] and Counter Braids [36], [37] and other solutions [2]–[6] share counters among flows to reduce the memory overhead. Among them the widely acknowledged one is CountMin, serving as a building block for a number of sophisticated measurement solutions.

Flow spread measurement: Bitmap [5], [33], [34], FM sketch [38], and HLL sketch [30], [39] can measure the spread of a single flow, called spread estimator. Because of the same reason as the solution for flow size measurement, we need sketch-based solutions to save memory. To measure the flow spread for each flow in the packet stream, CSE [20] and vHLL [21] reduce memory consumption through space sharing. bSketch [17] and vSketch [17] propose a family of sketches using plug-ins like bitmaps [40], FM sketches [38] and HLL sketches [30], [39]. Among all the solutions, vSketch using HLL sketches, denoted as vSkt(HLL), and vHLL, are the most accurate solutions.

B. Traffic Measurement for Sliding Window

Traffic measurement for sliding window cares about the same packet stream as T-query does if the window size is equal to T. However, T-query highlights the importance of answering the query in real time while some traditional solutions for sliding window cannot be implemented on online network processors. The reason is that most existing solutions based on sliding window divide the packet stream into multiple epochs, which can turn the traffic measurement problem for a sliding window to that for multiple fixed windows, but at the cost of using multiple independent sketches. The idea is directly applied by Zhou et al [24] for flow size measurement and CountMin is used to measure packets stream in one epoch. Gou et al. reduces the memory consumption by using a CountMin with n rows for a sliding window with n epochs. They propose a scanning operation to circularly clear the counters in CountMin, ensuring that counters in different rows store the traffic statistics of different numbers of epochs and the minimum value among n hashed counters can approximately estimate the size of the queried flow. Flow spread measurement for a sliding window is studied. The state of the art is VATE [23], which is based on CSE, a solution for flow spread measurement for fixed epochs. VATE expands every bit in CSE to a counter and the value of the counter (except 0) is a time stamp. For T-query, VATE only needs to collect the counters whose values locate in the latest time period of T and perform similar query to what CSE does.

VI. Conclusion

Real-time obtaining the traffic statistics in the most recent time period of T is crucial for many practical network applications. This paper proposed solutions to answering the T-query on network traffic in real time. We employ the offloading model to significantly reduce the on-die memory resource consumption, making it possible to implement the solutions on online network processors. Theoretical analysis and experimental results show that our solutions outperform existing work in terms of estimation accuracy and memory efficiency.

Acknowledgement

This work was supported in part by National Science Foundation of US under grant CNS-1909077 and a grant from Florida Center for Cybersecurity.

References

[1] Cisco, "Cisco IOS NetFlow," Online. [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[2] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," *Proc. of NSDI*, pp. 29–42, 2013.

[3] 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.

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

[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] D. Plonka, "FlowScan: A Network Traffic Flow Reporting and Visualization Tool," *Proc. of USENIX LISA*, 2000.

[8] 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.

[9] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial of Service Activity," *Proc. of USENIX Security Symposium'2001*, August 2001.

[10] P. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling High Bandwidth Aggregates in the Network," *Computer Communications Review*, vol. 32, no. 3, pp. 62–73, July 2002.

[11] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," *Proc. of NDSS*, February 2005.

[12] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and Early Warning for Internet Worms," in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 190–199.

[13] C. C. Zou, W. Gong, D. Towsley, and L. Gao, "The Monitoring and Early Detection of Internet Worms," *IEEE/ACM Transactions on networking*, vol. 13, no. 5, pp. 961–974, 2005.

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

[15] M. Feily, A. Shahrestani, and S. Ramadass, "A Survey of Botnet and Botnet Detection," in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. IEEE, 2009, pp. 268–273.

[16] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum, "BotGraph: Large Scale Spamming Botnet Detection." in *NSDI*, vol. 9, 2009, pp. 321–334.

[17] 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.

[18] 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.

[19] Q. Xiao, S. Chen, M. Chen, and Y. Ying, "Hyper-Compact Virtual Estimators for Big Network Data Based on Register Sharing," *in Proc. of ACM SIGMETRICS*, 2015.

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

[21] Q. Xiao, S. Chen, Y. Zhou, M. Chen, J. Luo, T. Li, and Y. Ling, "Cardinality Estimation for Elephant Flows: A Compact Solution based on Virtual Register Sharing," *IEEE/ACM Transactions on Networking*, 2017.

[22] X. Gou, L. He, Y. Zhang, K. Wang, X. Liu, T. Yang, Y. Wang, and B. Cui, "Sliding sketches: A framework using time zones for data stream processing in sliding windows," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1015–1025. [Online]. Available: https://doi.org/10.1145/3394486.3403144

[23] J. Xu, W. Ding, X. Hu, and Q. Gong, "Vate: A trade-off between memory and preserving time for high accurate cardinality estimation under sliding time window," *Computer Communications*, vol. 138, pp. 20–31, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014036641830625X

[24] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Per-flow counting for big network data stream over sliding windows," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, 2017, pp. 1–10.

[25] H. Wang, D. Zhang, and K. G. Shin, "SYN-dog: Sniffing SYN Flooding Sources," *Proc. of 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, July 2002.

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

[27] 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.

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

[29] M. Yoon, T. Li, S. Chen, and J. Peir, "Fit a Spread Estimator in Small Memory," *Proc. of IEEE INFOCOM*, April 2009.

[30] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," *Proc. of AOFA*, pp. 127–146, 2007.

[31] Caida anonymized 2018 internet traces. [Online]. Available: http://www.caida.org/data/overview/

[32] Wikipedia contributors, "Denial-of-service attack — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Denial-of-service_attack&oldid=1026628937, 2021, [Online; accessed 8-June-2021].

[33] 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, June 1990.

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

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

[36] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter Braids: A Novel Counter Architecture for Per-Flow Measurement," *Proc. of ACM SIGMETRICS*, June 2008.

[37] Y. Lu and B. Prabhakar, "Robust Counting Via Counter Braids: An Error-Resilient Network Measurement Architecture," *Proc. of IEEE INFOCOM*, April 2009.

[38] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for database applications," *Journal of Computer and System Sciences*, vol. 31, pp. 182–209, September 1985.

[39] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in Practice: Algorithmic Engineering of a State-of-The-Art Cardinality Estimation Algorithm," *Proc. of EDBT*, 2013.

[40] 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.