2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00010

Supporting Real-time Networkwide T-Queries in High-speed Networks

Yuanda Wang* Haibo Wang* Chaoyi Ma Shigang Chen
Department of Computer & Information Science & Engineering,
University of Florida, Gainesville, FL 32611, USA,
Email: {yuandawang, wanghaibo, ch.ma}@ufl.edu, sgchen@cise.ufl.edu

***Abstract*—Traffic measurement is key to many important network functions. Supporting real-time queries at the individual flow level over networkwide traffic represents a major challenge that has not been successfully addressed yet. This paper provides the first solutions in supporting real-time networkwide queries and allowing a local network function (for performance, security or management purpose) to make queries at any measurement point at any time on any flow's networkwide statistics, while the packets of the flow may traverse different paths in the network, some of which may not come across the point where the query is made. Our trace-based experiments demonstrate that the proposed solutions significantly outperform the baseline solutions derived from the existing techniques.**

I. Introduction

Real-time traffic measurement is key to supporting important network functions, such as identifying elephant flows [1], [2], re-routing traffic to resolve congestion and balance traffic across the network [3], [4], detecting denial-of-service attacks [5]–[7], worm propagation [8]–[10] and scanning [10]–[12], as well as profiling potential botnet activities [13]. There are two major challenges in designing traffic measurement modules that run on network devices (such as routers and switches) to collect information from the arrival packet streams: One is to support real-time queries on traffic statistics at the individual flow level [14], [15], and the other is to support a generalized flow model [2], [16] where the packets of a flow (such as all packets towards a given destination address) may pass many network paths and its measurement requires synthesizing networkwide data from multiple measurement points. The prior work has only limited success in addressing these two challenges.

To support real-time responses, it is highly desired that the traffic measurement modules are implemented on the data plane, examining packet streams at line rates directly on network processors [1], [15], [17]. But this means they have to compete for on-die processing and memory (such as SRAM) resources of the network processors with other key network functions of packet forwarding, queuing and scheduling, quality of service, etc. One solution is to implement traffic measurement with highly compact and efficient data structures called *sketches* [18]–[21], which however bring their own problems: Many sketches are efficient in recording information from a packet stream, but inefficient in answering flow-level queries. They adopt a strategy of online recording and offline queries [18], [21], [22], which does not support real-time applications. For some sketches that claim to support real-time queries, they are designed to measure network traffic and answer queries within each measurement epoch [1], [15]. At the beginning of each epoch, little information will be returned for a query. As time progresses, more information will be returned, which depends on the time of the query within the current epoch. A better approach of supporting real-time queries is based on a time window model [23]–[25], which returns the statistics of a flow within a sliding window $[t-T, t)$, where t is the time when the query is made and T is the window size. We refer such queries as T-*queries*. The challenge is that sketches do not keep a time stamp for each piece of information that is recorded in their data structures and therefore it is difficult for them to remove the stale information that should be moved out of the window. The existing solutions [23]–[25] divide the window into n smaller epoches of length $\frac{T}{n}$ and n sketches are used to store the measurement data in these epoches respectively. As a new epoch starts, the data for the oldest epoch is removed and its memory is released to store new information in the current epoch. The accuracy of this approach depends on the value of n. As we increase n, we can improve the accuracy in answering T-queries.

The above window-based solutions however have serious performance issues. They have to store n sketches, each for an epoch in the window. As n increases, their memory requirement multiplies; or under a fixed memory allocation their per-sketch memory decreases, which has significant impact on measurement accuracy for each sketch. Moreover, a query requires processing n sketches with an overhead that increases with n. Another limitation is that all prior work on T-queries is designed for answering queries on local traffic at one measurement point. No existing work addresses the practically important problem of answering real-time queries on networkwide traffic.

There exists some work on networkwide traffic measurement [19], [21]. However, their model is not window-based and does not support real-time queries. Instead, they are designed for offline queries, with all measurement points forwarding locally measured data to a centralized measurement center, where the data are combined to answer queries.

This paper attempts to provide the first solutions in supporting real-time networkwide T-queries and allowing a local network function (for performance, security or management) to make queries at any measurement point at any time on any flow's networkwide statistics, while the packets of the flow may traverse different paths in the network, some of which may not come across the point where the query is made. We support approximate T-queries, yet do not incur significant memory cost and processing overhead that linearly grow with n. Instead, we work with a fixed memory allocation at any measurement point and store only the information of the current epoch as well as the aggregate information of the previous epochs within

*equal contribution

2575-8411/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDCS54860.2022.00010

the window. We rely on a measurement center to store the detailed data from all measurement points, synthesize the data and distribute aggregate information back to the measurement points to support localized real-time queries on networkwide statistics at the individual flow level. We stress that, different from [19], [21], the proposed solution supports window-based queries in real time. In addition, the method of combining data from different measurement points in [21] assume that all points use sketches of the same size. We relax the requirement because different points (e.g., routers) have different traffic conditions and resource availability and therefore should be allowed to commit different amounts of resources (thus different sketch sizes) for local traffic measurements. The main contributions of this paper are summarized below.

- We define a new problem of supporting approximate real-time networkwide T-queries.
- We present two solutions for collecting flow-level traffic statistics locally, synthesizing the data networkwide, and answering T-queries efficiently and accurately.
- We conduct experiments based on real network traffic traces to evaluate the performance of the proposed solutions. The experimental results demonstrate that our solutions significantly outperform the baseline solutions derived from the existing techniques.

II. Preliminaries

A. System Model and Flow Model

Our traffic measurement system consists of a measurement center and a set of p (> 1) measurement points, denoted as $V = \{v_0, v_1, ..., v_{p-1}\}$ with $|V| = p$. The measurement center is hosted at a powerful server with adequate computing/memory resources for measurement data storage and synthesis. It sets up connections with all measurement points for coordination and data exchange. The measurement points can be any network devices (such as gateways, routers, switches, or firewalls) at which a traffic measurement module is deployed.

Consider the packet stream arriving at a measurement point. Each packet is abstracted as $\langle f, e\rangle$, where f is a flow label and e is an element identifier. Flow label f is typically composed from a selected subset of packet header fields, such as source address, destination address, port numbers, protocols, or others, depending on the application need. Element e may be a count (e.g., 1 for the packet itself) or a value selected from the packet headers or even the payload. All the packets with the same label f form a *flow*, referred to as flow f.

Flow size is defined as the number of elements in one flow, e.g., the number of packets. Size measurement can provide information about traffic volumes of individual flows for billing or traffic engineering and help identify elephant flows for traffic shaping, congestion control or attack detection.

Flow spread is defined as the number of *distinct elements* in a flow. Spread measurement can assist in detecting malicious network activities. For example, consider deploying measurement points at the gateway routers of an enterprise network to monitor inbound traffic. If we use source address as flow label and destination address as element identifier, then all packets from the same external source form a flow. Measuring flow spread can help detect external sources that are scanning the internal network — these sources have each contacted too many distinct internal destination addresses. In another example, if we use destination address as flow label and source address as element identifier, all external packets to the same internal destination form a flow. Measuring the spread of each internal destination, i.e., how many external sources are sending packets to an internal host, can detect distributed denial-of-service (DDoS) attacks [26], [27].

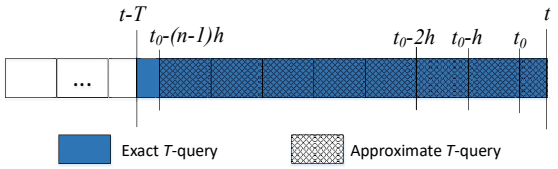


Fig. 1: Exact T-query vs. approximate T-query at one measurement point and time t.

A flow may be spatial-temporally distributed. In the above DDoS detection example, the flow of external packets to an internal destination may arrive from all over the Internet through different gateways of the enterprise network, and the size/spread of the flow changes over time. Hence, to measure this flow, we need the information from all the measurement points over time.

B. Problem of Real-Time T-query and Prior Work

The problem of *exact real-time T-query* is to answer the size (or spread) of any flow f in time window of $[t - T, t)$, where t is an arbitrary time instant and T is a pre-specified window size.

First let's consider the simple case where there is only one measurement point v_0. Let $M_0^{s,t}$ be the measurement data for all flows during the period from time s to time t at the measurement point v_0. To save memory space, the measurement data are stored in compact data structures called sketches [14], [15]. From $M_0^{t-T,t}$, we can extract the measurement of any flow f during time window $[t - T, t)$ to answer T-queries. However, maintaining $M_0^{t-T,t}$ as time t continuously advances is a difficult problem. When time t advances to $t + \Delta t$, not only do we need to add information about new packets arrived during $[t, t + \Delta t)$, but also we have to remove the information about packets arrived during $[t-T, t-T+\Delta t)$, which is hard to implement unless we store information about each packet with an arrival timestamp.

Therefore, the prior work [23]–[25] relaxes the problem for *approximate T-query*. It splits time into fixed measurement epochs of length $h = \frac{T}{n}$, where n is a pre-specified parameter. Starting from the initial time 0, epochs are $[(i - 1)h, ih)$, for $i \geq 1$. As shown in Fig. 1, we let t_0 be the ending time of the most recent measurement epoch before t. Unless $t = t_0$, the window $[t - T, t)$ contains $n - 1$ completed epochs, in addition to the current epoch and a partial epoch before those completed epochs. This partial epoch will be ignored by approximate T-query. The measurement data of the $(n - 1)$ completed epochs are $M_0^{t_0-h,t_0}$, $M_0^{t_0-2h,t_0-h}$, ..., $M_0^{t_0-(n-1)h,t_0-(n-2)h}$. The measurement data of the current epoch (up to time t) is denoted as $M_0^{t_0,t}$. Let $\bigcup$ be the operator that aggregates the measurement data from different epochs. Its exact operation depends on which sketch is used for the measurement data, as we will explain later. To answer an approximate T-query, we aggregate the measurement data of the $(n-1)$ completed epochs and the current epoch, i.e., $(\bigcup_{i=1}^{n-1} M_0^{t_0-ih,t_0-(i-1)h}) \bigcup M_0^{t_0,t}$,

from which the measurement data of individual flows can be extracted. This paper will focus on approximate T-query, which will approach exact T-query as we increase the value of n.

The prior work on approximate T-query is limited in three regards. First, they only consider a single measurement point [23]–[25] or non-real-time offline queries [21], whereas *this paper considers real-time queries on flows whose packets may pass multiple or all measurement points*. Second, due to varied traffic volumes, workloads and functions, different measurement points may commit different amounts of resources (such as memory) to store their measurement data, resulting in sketches of different sizes, which makes it difficult to aggregate them. This is an issue that is not considered in the existing single-point study. Third, the prior work has to maintain the measurement data of the $(n-1)$ completed epochs before the current epoch. This incurs high memory cost as n increases.

The new problem studied in this paper is called *approximate real-time networkwide T-query*, which allows a network function at any measurement point v_x to query on any flow f at an arbitrary time t for the flow's size (or spread) across all measurement points. Consider the previous example of an enterprise network with multiple gateways (multiple measurement points) monitoring the external sources that send packets to each internal destination. All external packets to the same internal destination form a flow; the packets of a flow may pass through all measurement points. As each gateway records the spread information of the arrival packets, it may sample the packet destinations to query about their current spreads in real time, which is *the number of distinct sources that send packets to this destination across all measurement points during* $[t-T, t)$. For an approximated answer, one may expect that we aggregate the measurement data of the recent $(n-1)$ completed epochs and the current epoch from all points. This is however not true. As traffic measurement is done epoch by epoch, the measurement data of the current epoch from other points will not be updated until the end of the epoch. Moreover, the query can be made at any time t in the current epoch, i.e., $\forall t \in [t_0, t_0+h)$, where t_0 is the ending time of the previous epoch (which is the last completed epoch). Given that $(t-t_0)$ can be arbitrarily small, we cannot guarantee that the measurement data of the previous epoch is obtained and aggregated before t. Assuming that the round-trip communication delay between the measurement center and any measurement point is bounded by h, what we can aggregate is the measurement data of the recent $(n-2)$ completed epochs before the last completed epoch from all points. Let $\tilde{M}_x$ be the measurement data that v_x can obtain to answer the approximate real-time networkwide T-query. We have

$$\tilde{M}_x = \Big(\bigcup_{v_{x'} \in V} \bigcup_{i=2}^{n-1} M_{x'}^{t_0-ih, t_0-(i-1)h}\Big) \bigcup M_x^{t_0-h, t_0} \bigcup M_x^{t_0, t}. \quad (1)$$

$\tilde{M}_x$ is plotted in Fig. 2 with crosshatch pattern, whereas the answer to the exact networkwide T-query is in blue color.

III. Challenges and Main Idea

A. Challenges

Supporting approximate real-time networkwide T-queries is challenging as the solution should achieve high estimation accuracy and in the meanwhile satisfy the following requirements.

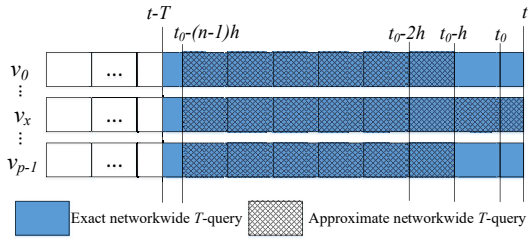


Fig. 2: Exact networkwide T-query vs. approximate networkwide T-query for any measurement point v_x and time t.

- Memory Efficiency: The traffic measurement modules are usually implemented on the data plane to record and examine the packet streaming directly on the network processors. Given that the on-die memory resource of the network processor is limited and shared by key networking functions such as packet forwarding, queuing and scheduling, etc., the solution should be memory efficient.
- Low Query Overhead: Query overhead competes for processing cycles that would otherwise be used to record the packet stream or for other network functions. High query overhead means that queries can only be performed sparsely, which is undesired for real-time applications that rely on frequent queries to catch events (such as DDoS attacks) at they happen.
- Handle Device Diversity: Different points may have different amount of available memory for the measurement module for two reasons. First, device heterogeneity. The measurement points may be different types/generations of devices as the equipment replacement/renewal is usually conducted progressively. Second, due to varied traffic volumes, workloads and functions, measurement modules among different points may be assigned different percentages of on-die memory. The solution should be able to aggregate sketches with different sizes and customize the size of the aggregated sketch for each measurement point.

As the system model described in Section II-A that the system consists of a measurement center and a set of p (> 1) measurement points is very common and classic (e.g., client-server model), one may come up with the following possible solutions.

- Naive Solution 1: Each measurement point performs both local traffic measurement and aggregation on measurements from peer points and multiple epochs. For real-time networkwide T queries, the measurement point can directly access local aggregation and produce answers instantly.
- Naive Solution 2: The measurement center collects the measurements of each epoch from all measurement points. When any measurement point receives a real-time networkwide T query on any flow f, the query will be forwarded to the center and the result produced by the center will be returned as the answer.

Naive solution 1 has a huge memory consumption that is linear to p and n, which is not scalable for large scale systems and for larger n (larger n makes the approximate T-query approach exact T-query and hence will increase measurement accuracy). We will elaborate this in the next subsection. Naive solution 2 requires a round-trip delay for answering a real-time networkwide T query,

which is significantly larger than directly accessing the local memory and producing the answer (which is what our solution will do). This is also validated by our experimental results in Section VII-D.

To address the above issues, *this paper does not attempt to build a new system/architecture*, instead, we follow the system model in Section II-A and propose a new design that carefully instructs the interaction between measurement center and points, so that the above requirements can be satisfied. Next subsection will describe the main idea of our design and explain why our design can satisfy the above requirements.

B. Main Idea

Our solutions are built on top of sketches. Consider any measurement point v_x and assume that time starts from 0. At the current time $t \in [(k-1)h, kh)$ of the kth epoch, with $k \geq n$, to answer the approximate real-time networkwide T-queries, there must be a sketch C that stores the measurement data $\tilde{M}_x$, i.e.,

$$\begin{aligned} C &= \tilde{M}_x \\ &= (\bigcup_{v_{x'} \in V} \bigcup_{i=2}^{n-1} M_{x'}^{t_0-ih, t_0-(i-1)h}) \bigcup M_x^{t_0-h, t_0} \bigcup M_x^{t_0, t} \\ &= (\bigcup_{v_{x'} \in V} M_{x'}^{(k-n)h, (k-2)h}) \bigcup M_x^{(k-2)h, t}. \end{aligned} \tag{2}$$

The above measurements are transformed from (1), where $t_0 = (k-1)h$. As t proceeds from the kth epoch to the $(k+1)$th epoch, i.e., $t \in [kh, (k+1)h)$, at the end of the kth epoch, C should be updated to

$$(\bigcup_{v_{x'} \in V} M_{x'}^{(k-n+1)h, (k-1)h}) \bigcup M_x^{(k-1)h, kh}.$$

This update should be conducted *instantly* to prepare C for the packet-recording and answering approximate real-time networkwide T-queries when the $(k+1)$th epoch starts, meaning that we need another sketch C' (with equal size) to store update information so that C just needs to copy the content of C', i.e.,

$$C' = (\bigcup_{v_{x'} \in V} M_{x'}^{(k-n+1)h, (k-1)h}) \bigcup M_x^{(k-1)h, kh}. \tag{3}$$

The key of our design is to make sure that (3) holds at the end of the kth epoch with $\forall k \geq n-1$, so that (2) holds at any time during the $(k+1)$th epoch. However, calculating (3) requires the measurement data of $(n-2)$ epochs from all points, i.e., $M_{x'}^{(k-n+1)h, (k-n+2)h}$, $M_{x'}^{(k-n+2)h, (k-n+3)h}$, ... , $M_{x'}^{(k-2)h, (k-1)h}$, $\forall v_{x'} \in V$ and local measurement data of the most recent completed epoch $M_x^{(k-1)h, kh}$, which together are $(p(n-2)+1)$ epochs. Note that we need to calculate (3) for arbitrary k with $k \geq n-1$, meaning that we have to store the measurement data of each epoch independently. This is a huge memory consumption especially when p and n are large. Recall that the on-die memory on the network processors that can be used for measurement is limited. It is impractical to store the measurement data of $(p(n-2)+1)$ epochs at the measurement point (This is also the reason why Naive Solution 1 is memory hungry). Moreover, aggregating the measurement data of $(p(n-2)+1)$ epochs is complicated, given that the sizes of sketches from different measurement points may be different (due to device diversity). Therefore, it is impractical for v_x to calculate (3) locally. Our main idea is to push the mass measurement data storage and the complex calculation to the measurement center that is hosted at a powerful server with adequate memory/computing resources, leaving measurement points the following lightweight tasks:

- Task 1: Record the measurement data of the current epoch and upload it to the center.
- Task 2: Maintain C to answer approximate real-time networkwide T-queries at the current kth epoch with $k \geq n-1$. Copy C' to C at the end of each epoch.
- Task 3: Maintain C' to ensure an instant transfer for C to answer approximate real-time networkwide T-queries in the next $(k+1)$th epoch. Wait for the center to return (3) and then update C'.

We will show that for flow spread measurement, we need an additional sketch that maintains the measurement data of the current epoch to accomplish Task 1, resulting in totally three sketches for all three tasks. While for flow size measurement, Task 1 can be accomplished by only maintaining C and C', resulting in a two-sketch design. Our design ensures that: 1) we only need constant (two for flow size and three for flow spread) number of sketches for each measurement point, which is memory efficient; 2) we only need to access C to answer queries, resulting in low query overhead; 3) the measurement center can aggregate measurements among sketches of different sizes, and return the aggregated result back to the measurement points. This allows our design to handle device diversity. One key technical challenge that is not adequately addressed in the existing literature is how to combine sketches of different sizes and do so for different types of sketches.

In the next two sections, we will first propose a sophisticated three-sketch design for answering approximate real-time networkwide T-queries for flow spread, based on which we then propose a concise two-sketch design for flow size.

IV. Three-sketch Design for Flow Spread Measurement

This section proposes three-sketch design for answering approximate real-time networkwide T-queries on the statistics of flow spread. We first review the sketch solution called rSkt2(HLL) [15] which performs per-flow spread measurement in a single epoch at a single measurement point. We then describe the proposed three-sketch design without considering device diversity. After that, we propose a method to deal with device diversity. Finally, we discuss the proposed design.

A. rSkt2(HLL)

rSkt2 [15] is a state-of-the-art framework for per-flow spread measurement, which can be plugged in different single-flow estimators, primarily including bitmap, FM, and HyperLogLog (HLL) [28] [29]. Among them, rSkt2 that uses HLL estimators, denoted as rSkt2(HLL), is the most accurate and is thus adopted in this paper. Before reviewing rSkt2(HLL), we first describe the HLL estimator, whose data structure is an array of m HLL registers, each of r (usually 5) bits storing an integer in the range of $[0, 2^r - 1]$. rSkt2(HLL) is a sketch for per-flow spread measurement. Its data structure is two-dimension HLL estimator arrays, denoted as D. D has 2 rows, $D[0], D[1]$, each with w HLL estimators. The jth HLL estimator in $D[0]/D[1]$ is denoted as $D[0][j]/D[1][j]$, $0 \leq j < w$. The lth HLL register in estimator $D[0]/D[1][j]$ is denoted as $D[0][j][l]/D[1][j][l]$, with $0 \leq l < m$.

When receiving a packet $\langle f, e\rangle$, rSkt2(HLL) first maps it to a pair of estimators, $D[0][H_0(f) \mod w]$ and $D[1][H_0(f) \mod w]$, and then hashes the packet to the $H_1(e)$th pair of HLL registers, $D[0][H_0(f) \mod w][H_1(e) \mod m]$ and $D[1][H_0(f) \mod w][H_1(e) \mod m]$, where $H_0(\cdot)$ and $H_1(\cdot)$ are independent uniform hash functions whose outputs are sufficiently large. Without confusion, we may omit the modulo operation. Let $g(f, H_1(e))$ be a pseudo-random function taking two input parameters f and $H_1(e)$ and returning a bit, 0 or 1, with equal probability. The packet will be recorded to the HLL register $D[0][H_0(f)][H_1(e)]$ (if $g(f, H_1(e)) = 0$) or $D[1][H_0(f)][H_1(e)]$ (if $g(f, H_1(e)) = 1$) as follows. $\forall 0 \le u < 2, 0 \le i < w, 0 \le j < m$,

$$D[u][i][j] = \max\{D[u][i][j], G(f \oplus e)\}$$

where $\oplus$ is the XOR operation and $G(\cdot) \in [1, 2^r - 1]$ is a geometric hash function and $G(\cdot) = x$ with probability 2^{-x}.

For $0 \le i < m$, define

$$L_f[i] = D[0][H_0(f)][i], \bar{L}_f[i] = D[1][H_0(f)][i], \text{ if } g(f,i) = 0$$
$$L_f[i] = D[1][H_0(f)][i], \bar{L}_f[i] = D[0][H_0(f)][i], \text{ if } g(f,i) = 1.$$

We know any element e of flow f must be recorded in L_f. The elements from any flow $f' \neq f$ that is hashed to the same pair of estimators, $D[0][H_0(f)]$ and $D[1][H_0(f)]$, will be recorded in either L_f or $\bar{L}_f$ with equal chance. Let $V(.)$ be the value produced by the HLL estimator. rSkt2(HLL) produces the spread estimate $\hat{s}_f$ for flow f by subtraction.

$$\hat{s}_f = V(L_f) - V(\bar{L}_f) \tag{4}$$

rSkt2(HLL) outperforms existing work in terms of estimation accuracy, recording overhead and query overhead, and thus is adopted in this paper. For details of rSkt2(HLL), refer to [15].

B. Three-sketch Design without Device Diversity

We describe our three-sketch design that uses rSkt2(HLL) [21] to measure per-flow spread in each epoch. The same design can be easily modified to work with other sketches that measure per-flow spread [18], [20]. Our contribution is to build a solution on top of rSkt2(HLL) to answer approximate real-time networkwide T-queries on flow spread. Consider any measurement point v_x, its three-sketch design uses three rSkt2(HLL) sketches, denoted as B, C and C', respectively. We begin with the case where all measurement points use the sketches of the same size. That is, each sketch is an array of $2 \times w \times m$ HLL registers. B measures the traffic in the current epoch (for Task 1 in Section III-B). C contains the aggregate measurement from which we can answer approximate real-time networkwide T-queries (Task 2 in Section III-B). C' contains the measurement that enalbes instant update of C when proceeding to the next epoch (Task 3 in Section III-B). Initially, all registers in B, C and C' are set to zeros. our design has the following three stages.

1) ***Local online recording and query****:* The packets will be recorded in B, C and C' in the same way as rSkt2(HLL) does. For query on flow f, we operate on C, which produces the spread estimate following (4).

2) ***Local periodical measurement update****:* At the end of the kth epoch with $k \ge n-1$, for any v_x, the measurement module executes three actions: 1) send B to the center; 2) copy C' to C; 3) reset C' to zeros. The second action ensures an instant transfer to the $(k+1)$th $(k+1 \ge n)$ epoch as C is ready for the online recording and query in next epoch. However, this holds with the prerequisite that (3) holds for C' at the end of the kth epoch with $k \ge n-1$, meaning that C' should obtain measurement data

$$\bigcup\nolimits_{v_{x'} \in V} M_{x'}^{(k-n+1)h,(k-1)h} = M^{(k-n+1)h,(k-1)h} \tag{5}$$

before the end of the kth epoch, which will be sent by the center ($M_x^{(k-1)h,kh}$ in (3) is possessed by C' via online recording). Next, we will describe how the center collects the local measurement data sent from each point (benefited from the first action) and returns the measurement data $M^{(k-n+1)h,(k-1)h}$ before the end of the kth epoch with $k \ge n-1$. The process is called *spatial-temporal (ST) join*.

3) ***Remote ST join****:* Denote B at v_x at the end of the kth epoch as $B_{x,k}$. During the kth epoch, the center has received $B_{x,l}, \forall v_x \in V, 1 \le l \le k-1$, which will be aggregated for each measurement point $v_x \in V$ to obtain $M_x^{(k-n+1)h,(k-1)h}$. The aggregating process is called *temporal join* and its operation is

$$M_x^{(k-n+1)h,(k-1)h} = \bigcup\nolimits_{l=k-n+2}^{k-1} M_x^{(l-1)h,lh} \tag{6}$$

where $\bigcup$ on any two measurement data M_x and $M_{x'}$ (which are two rSkt2(HLL) sketches with the same w, m) is the register-wise max operation, i.e., $\forall 0 \le u < 2, \le i < w,\ 0 \le j < m$,

$$[M_x \bigcup M_{x'}][u][i][j] = \max\{M_x[u][i][j], M_{x'}[u][i][j]\}. \tag{7}$$

Then, the center will obtain $M^{(k-n+1)h,(k-1)h}$ by aggregating all $M_x^{(k-n+1)h,(k-1)h}, \forall v_x \in V$ spatially. This aggregating process is called *uniform spatial join* and its operation is $\forall 0 \le u < 2, \le i < w,\ 0 \le j < m$,

$$M^{(k-n+1)h,(k-1)h}[u][i][j] = \max_{v_x \in V}\{M_x^{(k-n+1)h,(k-1)h}[u][i][j]\}. \tag{8}$$

The word "uniform" means that the rSkt2(HLL) sketches at different measurement points are assumed to have the same w and m. We will discuss how to aggregate $M_x^{(k-n+1)h,(k-1)h}, \forall v_x \in V$ spatially when the rSkt2(HLL) sketches at different measurement points are with different sizes in next subsection.

$M^{(k-n+1)h,(k-1)h}$ in (8) is exactly what the measurement point v_x needs for updating C' (see (5)). The center will return it to the measurement point. As we have assumed, the round-trip transmission time is smaller than h. Since the center has adequate computing resources, we also assume that the time for the ST join plus the round-trip transmission time is smaller than h. Therefore, we can guarantee that $M^{(k-n+1)h,(k-1)h}$ can be sent to the measurement point before the end of the kth epoch.

C. Handle Device Diversity with Nonuniform Spatial Join

Due to device diversity, the rSkt2(HLL) sketches at different points may be different sizes. Actually, the number of register in each HLL estimator, i.e., m is recommended to be a constant number, e.g., 128, to ensure accuracy of each estimator [28], therefore, it is reasonable to assign the same value of m to C and C' at each measurement point. In contrast, the number of estimators, i.e., w, may vary depending on the memory allocated to the measurement module.

Denote the number estimators for v_x as w_x. In this subsection, we consider the case where $w_0, w_1, w_2, ..., w_{p-1}$ are not all the same. Let $w_0 \le w_1 \le ... \le w_{p-1}$ and we assume $\frac{w_x}{w_{x-1}}$ to be the power of 2, $\forall 1 \le x < p$. In this case, the uniform spatial join that directly does register-wise max in (8) will not work and we need to aggregate $M_x^{(k-n+1)h,(k-1)h}$ across all points $\forall v_x \in V$ in a different manner, called *nonuniform spatial join*.

We propose a method called *expand-and-compress*. The center will column-wise expand each $M_x^{(k-n+1)h,(k-1)h}$ that contain w_x estimators to $M_{x,e}^{(k-n+1)h,(k-1)h}$ that contains w_{p-1} counters such that $\forall 0 \le u < 2, 0 \le i < w_{p-1}, 0 \le j < m$,

$$M_{x,e}^{(k-n+1)h,(k-1)h}[u][i][j] M_x^{(k-n+1)h,(k-1)h}[u][i \bmod w_x][j]. \tag{9}$$

After expansion, $M_{x,e}^{(k-n+2)h,kh}, \forall v_x \in V$ contains w_{p-1} estimators. The center combines $M_{x,e}^{(k-n+2)h,kh}, \forall v_x \in V$ to $M_*^{(k-n+1)h,(k-1)h}$ as follows. $\forall 0 \le u < 2, 0 \le i < w_{p-1}, 0 \le j < m$,

$$M_*^{(k-n+1)h,(k-1)h}[u][i][j] = \max_{v_x \in V} M_{x,e}^{(k-n+1)h,(k-1)h}[u][i][j]$$

For a certain measurement point v_x, the rSkt2(HLL) sketches on it are $2 \times w_x \times m$ register arrays while $M_*^{(k-n+2)h,kh}$ is $2 \times w_{p-1} \times m$. Therefore, the center needs to compress the size of $M_*^{(k-n+1)h,(k-1)h}$ from $2 \times w_{p-1} \times m$ to $2 \times w_x \times m$ and sends it back to v_x. $\forall 0 \le u < 2, 0 \le i < w_{x-1}, 0 \le j < m$,

$$\begin{aligned} &M^{(k-n+1)h,(k-1)h}[u][i][j] \\ &= \max_{0 \le l < \frac{w_{p-1}}{w_x}} \{M_*^{(k-n+1)h,(k-1)h}[u][i + l w_x][j]\}. \end{aligned}$$

Note that in Section IV-B where we do not consider device diversity, $M^{(k-n+1)h,(k-1)h}$ in (8) is a $2 \times w \times m$ counter array and will be sent to each measurement point. In this subsection, $M^{(k-n+1)h,(k-1)h}$ is customized in size for v_x and will only be sent to v_x. Since v_x is an arbitrary point, our nonuniform spatial join is applicable for each point.

D. Discussion

Recall in Section II-B that the measurement data of the last completed epoch of all peer points cannot be obtained at the beginning of this epoch but is delayed for a time that is assumed to be smaller than h. Therefore, at the kth epoch with $k \ge n-1$, the center can calculate $\bigcup_{v_{x'} \ne v_x} M_{x'}^{(k-2)h,(k-1)h}$ by performing uniform spatial join in (8) for the case where there is no device diversity or perform nonuniform spatial join for the case where there is device diversity. Upon receiving $\bigcup_{v_{x'} \ne v_x} M_{x'}^{(k-2)h,(k-1)h}$, v_x will aggregate it directly to C by counter-wise addition. From (2), we have

$$\begin{aligned} C &= (\bigcup_{v_{x'} \ne v_x} M_{x'}^{(k-2)h,(k-1)h}) \bigcup \tilde{M}_x \\ &= (\bigcup_{v_{x'} \ne v_x} M_{x'}^{(k-2)h,(k-1)h}) \bigcup (\bigcup_{v_{x'} \in V} M_{x'}^{(k-n)h,(k-2)h}) \\ &\quad \bigcup M_x^{(k-2)h,t} \\ &= (\bigcup_{v_{x'} \in V} M_{x'}^{(k-n)h,(k-1)h}) \bigcup M_x^{(k-1)h,t}. \end{aligned} \tag{10}$$

This will make the approximate real-time networkwide T-query closer to the exact networkwide T-query.

V. Two-sketch Design for Flow Size Measurement

This section proposes two-sketch design for answering approximate real-time T-queries on the other kind of flow statistics, i.e., flow size. We first review the classical sketch, i.e., CountMin [14], which performs per-flow size measurement of a single epoch and at a single measurement point. We then describe the detailed design of the proposed two-sketch design, during which we will explain why only two-sketches are required for flow size measurement. Finally, a method to deal with device diversity is proposed. We will focus on the new content and avoid duplicate description.

A. CountMin

CountMin [14] uses a two-dimensional array of counters, denoted as C. It has d rows, each of w counters. The jth counter in the ith row is denoted as $C]i][j]$, $0 \le i < d$, $0 \le j < w$. Upon receiving a packet from flow f, CountMin records the packet by hashing f to one counter in each row and increases that counter by one, i.e., $C[i][H_i(f)] = C[i][H_i(f) \mod w] + 1$, where $H_i, \forall\ 0 \le i < d$ are independent pseudo-random hash functions whose outputs are sufficiently large. Without confusion, we abbreviate $H_i(f) \mod w$ as $H_i(f)$. Upon query on flow f, CountMin takes the minimum value of those d counters as the size estimate $\hat{s}_f$, i.e.,

$$\hat{s}_f = \min\{C[i][H_i(f)], 0 \le i < d\}. \tag{11}$$

B. Two-sketch Design without Device Diversity

Our two-sketch design employs CountMin for local flow size measurement. Consider any measurement point v_x, its two-sketch design uses two CountMin sketches, C and C' of the same size, i.e., $d \times w$. Initially, all counters in C and C' are set to zeros. Similar to the three-sketch design for flow spread measurement, the two-sketch design also has the three stages.

*1) **Local online recording and query**:* Each packet that arrives at the local measurement point will be online recorded in both C and C' in the same way as CountMin does. For query, the module returns $\min\{C[i][H_i(f)], 0 \le i < d\}$.

*2) **Local periodical measurement update**:* At the end of the kth epoch, $\forall k \ge 1$, the measurement module executes three actions: 1) send C to the center; 2) copy C' to C; 3) reset C' to zeros. The later two actions are the same as those under three-sketch design. The difference is the first action that the module send C rather than B (which we do not need here) to the center. To make (3) holds for C' at the end of the kth epoch with $k \ge n-1$, the ST join performed at the center should return the measurement data $M^{(k-n+1)h,(k-1)h}$ before the end of the kth epoch with $k \ge n-1$.

*3) **Remote ST join**:* Denote C that is sent from v_x at the end of the kth epoch as $C_{x,k}$. When the center receives $C_{x,k}$, it recovers the measurement data of the kth epoch at v_x:

$$M_x^{0,h}[i][j] = C_{x,1}[i][j],\ 0 \le i < d,\ 0 \le j < w$$

$$M_x^{(k-1)h,kh}[i][j] = \begin{cases} C_{x,k}[i][j] - C_{x,k-1}[i][j], \\ 1 < k < n,\ 0 \le i < d,\ 0 \le j < w; \\ C_{x,k}[i][j] - (M_x^{(k-2)h,(k-1)h}[i][j] \\ + \sum_{v_y \in V} \sum_{l=k-n+1}^{k-2} M_y^{lh,(l+1)h}[i][j]), \\ k \ge n,\ 0 \le i < d,\ 0 \le j < w. \end{cases}$$

The above operation reveals that the center can recover the measurement of each epoch, i.e., B in Task 1 of Section III-B from C. *This explains why for flow size we only need two sketches in each measurement point*. After measurement recovery of each epoch from the above equation, the center performs *temporal join* and obtains $M_x^{(k-n+1)h,(k-1)h}$ for each measurement point $v_x \in V$ if $k \geq n-1$. The temporal join here is counter-wise addition rather than register-wise max in the three-sketch design i.e., $\forall 0 \leq i < d,\ 0 \leq j < w$,

$$[M_x \bigcup M_{x'}][i][j] = M_x[i][j] + M_{x'}[i][j] \tag{12}$$

Then, the center will obtain $M^{(k-n+1)h,(k-1)h}$ by performing *spatial join*, i.e., $\forall 0 \leq i < d,\ 0 \leq j < w$,

$$M^{(k-n+1)h,(k-1)h}[i][j] = \sum\nolimits_{v_x \in V} M_x^{(k-n+1)h,(k-1)h}[i][j].$$

Similar to the three-sketch design, w here can be different and we will discuss how to perform *nonuniform spatial join* later. The center will return $M^{(k-n+1)h,(k-1)h}$ to the measurement point.

C. Handle Device Diversity with Nonuniform Spatial Join

We can also apply the *expand-and-compress* method for flow size measurement. The workflow is similar but the detailed operations vary. The expand operation in (9) for flow spread is modified to $\forall 0 \leq i < d, 0 \leq j < w_{p-1}$

$$M_{x,e}^{(k-n+1)h,(k-1)h}[i][j] = M_x^{(k-n+1)h,(k-1)h}[i][j \mod w_x].$$

After expansion, $M_{x,e}^{(k-n+2)h,kh}, \forall v_x \in V$ contains w_{p-1} counters in each row. The server combines all $M_{x,e}^{(k-n+2)h,kh}, \forall v_x \in V$ to $M_*^{(k-n+1)h,(k-1)h}$ as follows. $\forall 0 \leq i < d, 0 \leq j < w_{p-1}$,

$$M_*^{(k-n+1)h,(k-1)h}[i][j] = \sum_{v_x \in V} M_{x,e}^{(k-n+1)h,(k-1)h}[i][j].$$

The center needs to compress the size of $M_*^{(k-n+2)h,kh}$ from $d \times w_{p-1}$ to $d \times w_x$ and sends it back to v_x. $\forall 0 \leq i < d, 0 \leq j < w_{x-1}$,

$$M^{(k-n+1)h,(k-1)h}[i][j] = \max_{0 \leq l < \frac{w_{p-1}}{w_x}} \{M_*^{(k-n+1)h,(k-1)h}[i][j+w_x l]\}$$

Again, we stress that $M^{(k-n+1)h,(k-1)h}$ sent to each measurement point $v_x \in V$ is customized in size for v_x and will only be sent to v_x. Since v_x is an arbitrary point, our nonuniform spatial join is applicable for each point.

VI. Analysis

Denote C at measurement point v_x as C_x. For any measurement point $v_x \in V$ and any time t, the approximate real-time networkwide T-query focuses on the packet stream that appears during period $[t_0-(n-1)h, t_0-h)$ at peer points and that appears during $[t_0-(n-1)h, t)$ locally, which together are called *approximate networkwide T-stream*. For any flow f, let s_f be the actual size/spread estimate of flow f in the approximate networkwide T-stream and $\hat{s}_{f,x}$ be the estimate produced by C_x. We compare our design with an ideal case where we use one sketch (CountMin for size and rSkt2(HLL) for spread) to record the approximate networkwide T-stream. We assume a sketch C'_x, $\forall v_x \in V$ that has the same data structure as C_x, i.e., CountMin with size $d \times w_x$ for flow size and rSkt2(HLL) with size $2 \times w_x \times m$ for flow spread, and records the approximate networkwide T-stream. Let $\hat{s}'_{f,x}$ be the estimate produced by C'_x. Without device diversity, $\hat{s}'_{f,0}$ = $\hat{s}'_{f,1}$=...=$\hat{s}'_{f,p-1}$ as w_0=w_1=...w_{p-1}. We have the following theorems for our designs.

Analysis for three-sketch design:

Theorem 6.1: If $w_0 = w_1 =, ..., = w_{p-1}$ (in the case where there is no device diversity), we have $\hat{s}_{f,x}$=$\hat{s}'_{f,x}$, $\forall v_x \in V$.

Theorem 6.2: If $w_0 \leq w_1 \leq, ..., \leq w_{p-1}$ (in the case where there is device diversity), we have

$$|E(\hat{s}_{f,x}) - s_f| \leq |E(\hat{s}'_{f,0}) - s_f| \tag{13}$$

$$Var(\hat{s}_{f,x}) \leq Var(\hat{s}'_{f,0}) \tag{14}$$

Analysis for two-sketch design:

Theorem 6.3: If $w_0 = w_1 =, ..., = w_{p-1}$ (in the case where there is no device diversity), we have $\hat{s}_{f,x}$=$\hat{s}'_{f,x}$, $\forall v_x \in V$.

Theorem 6.4: If $w_0 \leq w_1 \leq, ..., \leq w_{p-1}$ (in the case where there is device diversity), we have $\hat{s}'_{f,p-1} \leq \hat{s}_{f,x} \leq \hat{s}'_{f,0}$

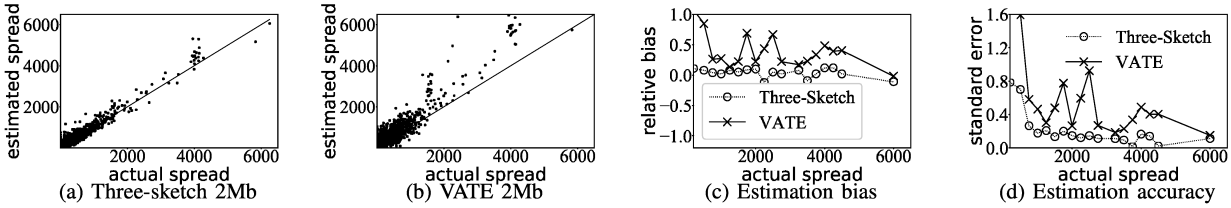
The proof of the above theorems are provided in [30] due to space limit.

VII. Performance Evaluation

A. Experiment Setup

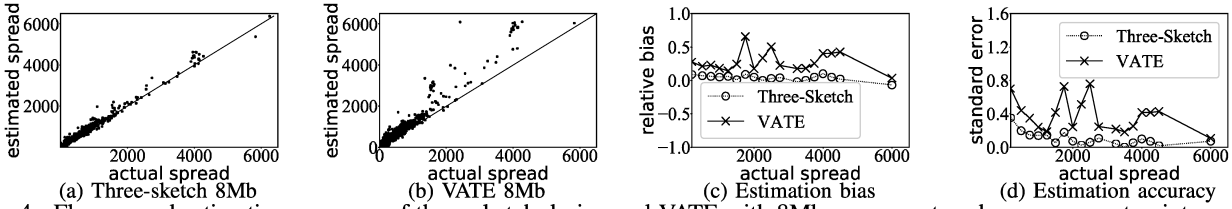
Our experiments support approximate real-time networkwide T-queries by running the two-sketch design for flow size and three-sketch design for flow spread at three measurement points, denoted as v_0, v_1, v_2, simulating a scenario where a network has three gateways to three ISPs (for robustness of Internet connection against ISP failure). T = 1 minute and each epoch lasts for 6s by default. We will evaluate the impact of the length of epoch. Since we are the first that proposes solutions for answering the approximate real-time networkwide T-queries, there is no prior work that can be used directly as baselines. To enhance our evaluation, we use the state-of-the-art work that support approximate T-queries (for one measurement point) as baselines. Specifically, each measurement point is embedded with the baseline solution. When answering real-time networkwide T-queries at an arbitrary measurement point $v_x = v_0, v_1, v_2$, each local measurement point communicates with the other two measurement points and adds up all the three measurement results locally, which serves as the answer to the query. For flow size measurement, we use Sliding Sketch as the baseline [31], whose main data structure is a two-dimension array. The number of rows is 10 in the original paper and we use the same parameter. For flow spread measurement, we use VATE [24] as the baseline, which measures the spread of each flow based on the bitmap algorithm [20], [32]. We set the length of bitmap for each flow to 2048, which can satisfy our measurement requirements and is consistent with the setting in VATE.

We use the real traffic traces downloaded from CAIDA in 2018 [33]. The dataset we use lasts for 30 mins. It contains 1018839925 packets and 5149105 different source IP addresses and 3306781 different destination IP addresses. The destination address is adopted as the flow label, which has the application of detecting DDoS attacks if our design is used to answer approximate real-time networkwide T-queries on flow spread. In the experiment, we divide the dataset into three packet streams,



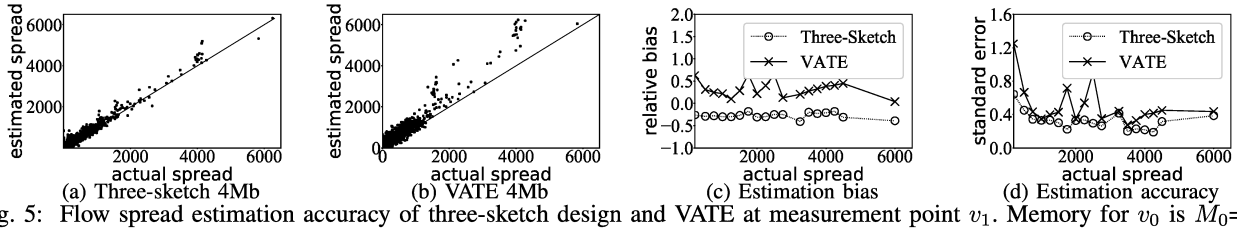
(a) Three-sketch 2Mb (b) VATE 2Mb (c) Estimation bias (d) Estimation accuracy

Fig. 3: Flow spread estimation accuracy of three-sketch design and VATE with 2Mb memory at each measurement point v_0, v_1, v_2.



(a) Three-sketch 8Mb (b) VATE 8Mb (c) Estimation bias (d) Estimation accuracy

Fig. 4: Flow spread estimation accuracy of three-sketch design and VATE with 8Mb memory at each measurement point v_0, v_1, v_2.



(a) Three-sketch 4Mb (b) VATE 4Mb (c) Estimation bias (d) Estimation accuracy

Fig. 5: Flow spread estimation accuracy of three-sketch design and VATE at measurement point v_1. Memory for v_0 is M_0=2Mb, for v_1 is M_1=4Mb, and for v_2 is M_2=8Mb.

each for a measurement point. Each packet in the dataset is randomly selected to one of the three packet streams.

Our evaluation is based on five metrics, categorized into three groups: 1) Estimation Accuracy: As we have explained in Section II-B and plotted in Fig. 2, the measurement module $v_x = v_0, v_1, v_2$ answers the approximate real-time networkwide T-queries rather than the exact networkwide T-queries (which cannot be done in real-time and is resource-hungry). Therefore, the true size/spread for any flow is its size/spread in the approximate neworkwide T-stream that has been defined in Section VI. There are three metrics to evaluate the estimation accuracy. 1) Absolute error, defined as $|\bar{s}_f - s_f|$, where s_f is the actual size/spread of f and $\hat{s}_f$ is the estimated size/spread of f. The average absolute error is defined as $\frac{\sum_{f\in\Gamma}|\bar{s}_f - s_f|}{|\Gamma|}$, where Γ is a flow set of $|\Gamma|$ flows; 2) Relative bias, defined as $\frac{\bar{s}_f - s_f}{s_f}$. It evaluates how the flow's estimated size/spread deviates its actual size/spread; 3) Relative standard error, defined as $\sqrt{\frac{\sum_{f\in\Gamma}((\hat{s}_f/s_f)-1)^2}{|\Gamma|}}$ for a flow set Γ of $|\Gamma|$ flows. 2) Online query overhead, defined as average time it takes to answer one approximate real-time networkwide T-query at the measurement point. The smaller it is, the more flows the measurement module can query on in one unit time. 3) Throughput. It represents the average number of packets recorded per second. The unit is one packet per second, which can be transformed to Abps if the average packet size is A (e.g., 1k) bits. High throughput ensures that the measurement module can be used to record the packet stream arriving at high rate.

We implement our designs and baselines in Java. We use three computers performing as three measurement points, each with a Quad-Core Intel Core i7 (2.7GHz), 16GB memory. The server is HP Z840, which has an E5-2643v4 CPU (6-Core,20M Cache, 3.4GHz), 256GB memory and around 10 TB disk storage.

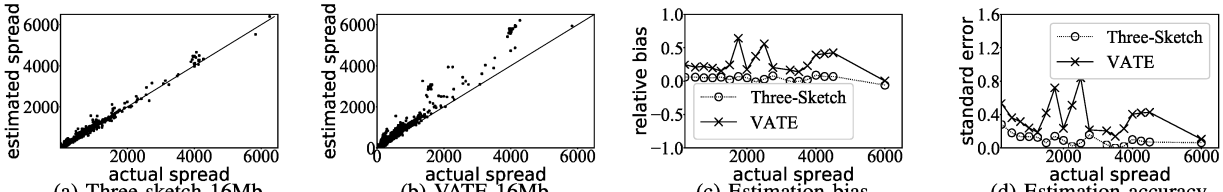
B. Estimation Accuracy for Flow Spread Measurement

We plot the same styles of figures as what we do for flow size measurement. The length of each epoch is 6s. We first conduct experiments for the case where each algorithm is allocated the same amount of memory. The results under 2Mb memory allocation for each algorithm are shown in Fig. 3 and under 8Mb memory allocation for each algorithm are shown in Fig. 4. Both figures clearly show that our three-sketch design for flow spread measurement outperforms VATE significantly.

Next we evaluate the performance of the three-sketch design when handling device diversity. For fair comparison VATE at different points will be allocated to the same memory as three-sketch. We allocate different amount of memory for different measurement points by setting M_1=2Mb, M_2=4Mb and M_3=8Mb. The results produced by the three-sketch design and VATE at the measurement point v_1 with 4Mb memory allocation are shown in Fig. 5. The results of the three-sketch design at v_0 and v_2 are shown in Figs. 7(a)-(b), respectively. The results demonstrate the advantages of the three-sketch design over VATE in terms of estimation accuracy. We increase the memory allocations to M_1=8Mb, M_2=16Mb and M_3=32Mb. The results produced by the three-sketch design and VATE at the measurement point with M_2=16Mb are shown in Fig 6. The results of the three-sketch design at measurement points v_0 with M_1=8Mb and v_2 with M_3=32Mb are shown in Figs. 7(c)-(d), respectively. The similar conclusion can also be drawn.

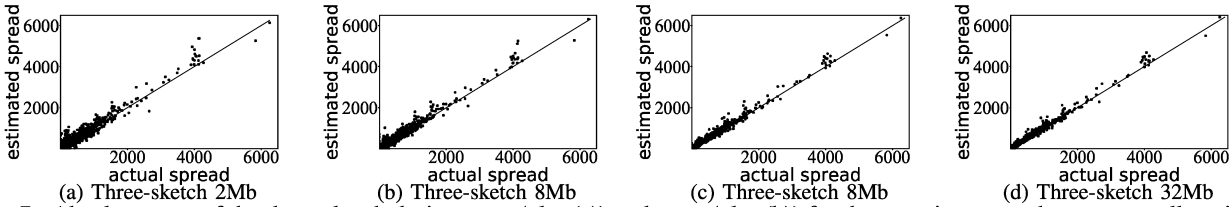
C. Estimation Accuracy for Flow Size Measurement

We plot scatter figures to illustrate the absolute error of each flow, where the x-axis is the real value (flow size in this subsection) and y-axis is the estimate. We also plot the line $y = x$. Each point in the figure represents a flow. The closer to $y = x$ the point is, the more accurate the estimate of the flow



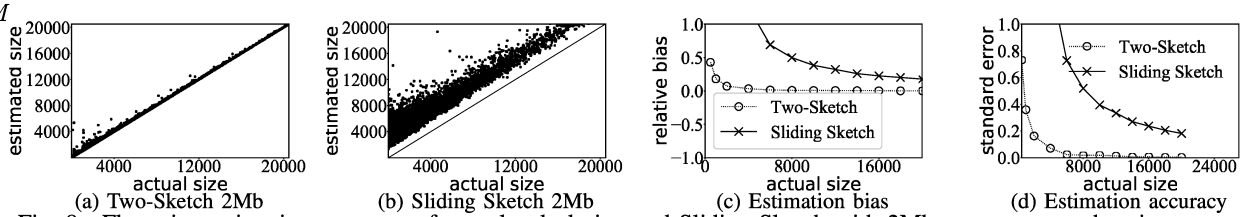
(a) Three-sketch 16Mb (b) VATE 16Mb (c) Estimation bias (d) Estimation accuracy

Fig. 6: Flow spread estimation accuracy of the three-sketch design and VATE at measurement point v_1. Memory allocations for v_0 is M_0=8Mb, for v_1 is M_1=16Mb, and for v_2 is M_2=32Mb.



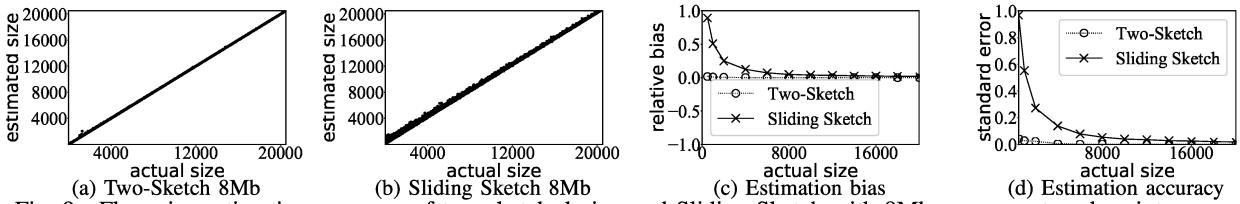
(a) Three-sketch 2Mb (b) Three-sketch 8Mb (c) Three-sketch 8Mb (d) Three-sketch 32Mb

Fig. 7: Absolute error of the three-sketch design at v_0 (plot (a)) and at v_2 (plot (b)) for the experiments under memory allocations of M_0=2Mb, M_1=4Mb, M_2=8Mb, and at v_0 (plot (c)) and at v_2 (plot (d)) for the experiments under memory allocations of M_0=8Mb, M



(a) Two-Sketch 2Mb (b) Sliding Sketch 2Mb (c) Estimation bias (d) Estimation accuracy

Fig. 8: Flow size estimation accuracy of two-sketch design and Sliding Sketch with 2Mb memory at each point v_0, v_1, v_2.



(a) Two-Sketch 8Mb (b) Sliding Sketch 8Mb (c) Estimation bias (d) Estimation accuracy

Fig. 9: Flow size estimation accuracy of two-sketch design and Sliding Sketch with 8Mb memory at each point v_0, v_1, v_2.

is. We also plot figures to show the distribution of the relative bias and relative standard error along with the actual flow size. We set the length of each epoch to be 6s.

We first conduct experiments for the case where each algorithm is allocated the same amount of memory. In this case, the results at v_0, v_1, v_2 will be the same. The results under 2Mb memory allocation are shown in Figure 8. The two-sketch design produces the estimates that are closer to the line $y = x$ in the scatter plot (Fig. 8(a)) than Sliding Sketch (Fig. 8(b)). As shown in Fig. 8(c), the relative bias of the two-sketch design is less than 5.0% of that of Sliding Sketch. The relative standard error of the two-sketch design is within 10% of that of Sliding Sketch. We allocate more memory for each algorithm, i.e., 8Mb. The results in Fig. 9 show Sliding Sketch's estimation accuracy is improved, but the two-sketch design still outperforms it significantly.

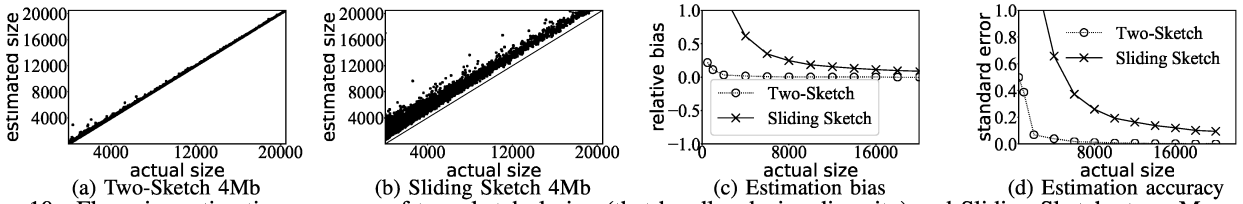
We also conduct experiments for the case where the algorithms at v_0, v_1, v_2 are allocated with different memories M_0, M_1, M_2, respectively, simulating the scenario with device diversity. In this case, we need the expand-and-compress method in the two-sketch design. Note that the two-sketch design at v_0, v_1, v_2 will produce different estimates while Sliding Sketch will produce the same estimate. We first Let M_0=2Mb, M_1=4Mb and M_2=8Mb. The results produced by the two-sketch design and Sliding Sketch at v_1 are shown in Fig. 10. The results of the two-sketch design at v_0, v_2 are shown in Figs. 12(a)-(b), respectively. Figures show that the two-sketch design (even the two-sketch design at v_0) perform much better than Sliding Sketch. Let M_0=8Mb, M_1=16Mb and M_2=32Mb. The results produced by the two-sketch design and Sliding Sketch at v_1 are shown in Fig 11. The results of the two-sketch design at v_0, v_2 are shown in Figs. 12(c)-(d), respectively. The similar conclusion can be drawn.

Two-Sketch	Sliding Sketch	Three-Sketch	VATE
0.068	1057	0.53	4104

TABLE I: Online query overheads (us) of the two-sketch design, three-sketch design, Sliding Sketch and VATE.

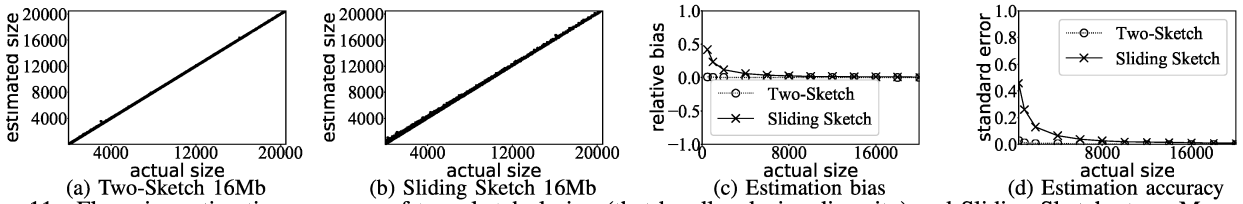
D. Online Query Overhead

The online query overheads of the two-sketch design and Sliding Sketch for flow size measurement and the three-sketch design and VATE for flow spread measurement are shown in Tbl. I. We stress that the results will be not affected by the memory allocation. For flow size measurement, the two-sketch design only needs $0.068us$ to answer one approximate real-time networkwide T-query, while Sliding Sketch needs $1057us$. For flow spread measurement, the three-sketch design's online query overhead is $0.53us$. In contrast, VATE's online query overhead is $4104us$, 6514 times larger. The reason is that our designs only need to access local sketches for answering queries while



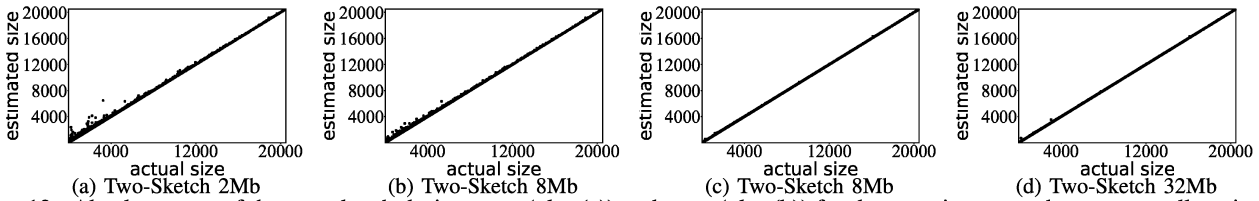
(a) Two-Sketch 4Mb (b) Sliding Sketch 4Mb (c) Estimation bias (d) Estimation accuracy

Fig. 10: Flow size estimation accuracy of two-sketch design (that handles device diversity) and Sliding Sketch at v_1. Memory for v_0 is M_0=2Mb, for v_1 is M_1=4Mb, and for v_2 is M_2=8Mb.



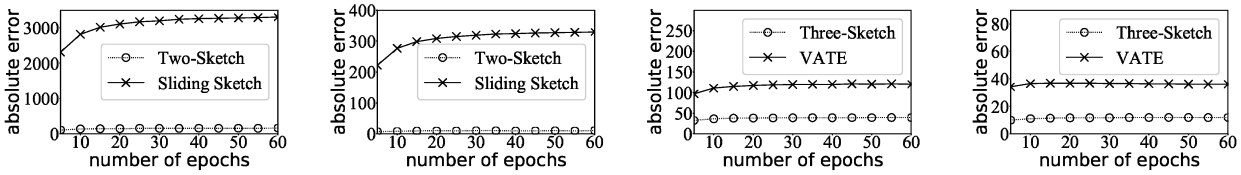
(a) Two-Sketch 16Mb (b) Sliding Sketch 16Mb (c) Estimation bias (d) Estimation accuracy

Fig. 11: Flow size estimation accuracy of two-sketch design (that handles device diversity) and Sliding Sketch at v_1. Memory for v_0 is M_0=8Mb, for v_1 is M_1=16Mb, and for v_2 is M_2=32Mb.



(a) Two-Sketch 2Mb (b) Two-Sketch 8Mb (c) Two-Sketch 8Mb (d) Two-Sketch 32Mb

Fig. 12: Absolute error of the two-sketch design at v_0 (plot (a)) and at v_2 (plot (b)) for the experiments under memory allocations of M_0=2Mb, M_1=4Mb, M_2=8Mb, and at v_0 (plot (c)) and at v_2 (plot (d)) for the experiments under memory allocations of M_0=8Mb, M_1=16Mb, M_2=32Mb.



(a) 2Mb (b) 8Mb (c) 2Mb (d) 8Mb

Fig. 13: Average absolute error of two-sketch and Sliding Sketch for flow size measurement under different memory: plots(13a) and (13c); of three-sketch and VATE for flow spread measurement under different memory: plots(13b) and (13d).

Two-Sketch	Sliding Sketch	Three-Sketch	VATE
4.03	2.9	5.34	2.99

TABLE II: Throughputs (10^6 packets per second) of the two-sketch design, three-sketch design, Sliding Sketch and VATE.

baselines need extra round-trip transmission time to fetch the results on other measurement points, and extra time to aggregate local measurement data of recent $(n-1)$ epochs and the current epoch. We want to stress that the improvement is significant because query overhead competes for processing cycles that would otherwise be used to record the packet stream. High query overhead means that queries can only be performed sparsely, which is undesired for real-time applications that rely on frequent queries to catch events (such as worm attacks) at they happen.

E. Throughputs

Previous results have shown that our designs can answer approximate real-time networkwide T-queries accurately. This part will show that our design will not degrade the online packet-recording speed. Tbl. II shows the throughput results of all algorithms. All algorithms can record millions of packets per second. Among them our designs are the fastest as we push the complex ST join to the server, leaving local measurement points lightweight operations.

F. Average Absolute Error under Different Epoch Lengths

We change the value of n from 5 to 60, corresponding to the epoch length from 12s to 1s. The x-axis is the value of n. The y-axis is the average absolute error for all flows in the dataset. We first conduct experiments for measuring flow size, where all the measurement points share the same memory size. The results under 2Mb are shown in Fig. 13a. The average absolute error of the two-sketch design keeps at a low value, which ranges from 110 to 166. In the same figure, the average absolute error of Sliding Sketch increases from 2313 to 3303, meaning we have reduced the absolute error by 94.97% to 95.67%. This happens because the baseline needs to keep the measurement data of n epochs, resulting in the memory allocated for each epoch much smaller, while our two-sketch design only needs to keep two sketches at the measurement point. We allocate more memory to each algorithm in Fig. 13b (i.e., 8Mb). The estimation accuracy of both Sliding Sketch and two-sketch has been improved, but our design is still a lot more accurate.

We also conduct experiments for measuring flow spread, where all the three measurement points (v_0, v_1, v_2) are allocated the same memory size. The same as flow size measurement, We change the value of n from 5 to 60. The results under 2Mb are shown in Fig. 13c. The average absolute error of three-sketch is low, ranging from 33 to 39. In the same figure, the average absolute error of VATE increases from 97 to 120, meaning we have reduced the absolute error by 65.98% to 67.5%. The reason is the same as that for the flow size estimation accuracy improvement of two-sketch over Sliding Sketch. We allocate more memory for each algorithm in Fig. 13d (i.e., 8Mb). Both estimation accuracy of VATE and three-sketch has been improved, but our design still performs better than VATE.

VIII. Conclusion

This paper proposes two solutions, i.e., the two-sketch design and three-sketch design to answer the approximate real-time networkwide T-queries on flow size and flow spread, respectively. Our designs are built on top of existing sketches. They utilize the huge resources on the server and only store aggregated networkwide traffic statistics at the local measurement point, so that the queries can be answered by only accessing the local data. We also propose an expand-and-compress method to deal with device diversity of all measurement points. Our trace-driven experiments demonstrate that our designs outperform the baselines significantly.

Acknowledgments

This work is funded by NSF grant CSR 1909077.

References

[1] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 164–176.

[2] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide Heavy Hitter Detection with Commodity Switches," in *Proceedings of the Symposium on SDN Research*, 2018, pp. 1–7.

[3] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A Load-balancing Mechanism for Distributed SDN Control Plane Using Response Time," *IEEE transactions on network and service management*, vol. 15, no. 4, pp. 1197–1206, 2018.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 254–265.

[5] 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.

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

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

[8] C. Zou, W. Gong, and D. Towsley, "Code Red Worm Propagation Modeling and Analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 138–147.

[9] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated Worm Fingerprinting," in *OSDI*, vol. 4, 2004, pp. 4–4.

[10] S. Chen and Y. Tang, "DAW: A Distributed Anti-Worm System," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, July 2007.

[11] 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.

[12] 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.

[13] 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.

[14] G. Cormode and S. Muthukrishnan, "Space Efficient Mining of Multigraph Streams," in *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2005, pp. 271–282.

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

[16] L. Tang, Q. Huang, and P. Lee, "SpreadSketch: Toward Invertible and Network-Wide Detection of Superspreaders," 2020.

[17] N. Ivkin, Z. Yu, V. Braverman, and X. Jin, "Qpipe: Quantiles sketch fully in the data plane," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 285–291.

[18] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact Virtual Estimators for Big Network Data Based on Register Sharing," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1. ACM, 2015, pp. 417–428.

[19] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic Sketch: Adaptive and Fast Network-wide Measurements," in *Proceedings of the 2018 ACM SIGCOMM Conference*, 2018, pp. 561–575.

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

[21] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized Sketch Families for Network Traffic Measurement," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–34, 2019.

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

[23] G. Cormode and K. Yi, "Tracking Distributed Aggregates over Time-based Dliding Windows," in *International Conference on Scientific and Statistical Database Management*. Springer, 2012, pp. 416–430.

[24] J. Xu, W. Ding, X. Hu, and Q. Gong, "VATE: A trade-off BetweenMemory and Preserving Time for High Accurate Cardinality Estimation under SlidingTime Window," *Computer Communications*, vol. 138, pp. 20–31, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014036641830625X

[25] 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*, 2020, pp. 1015–1025.

[26] H. Wang, C. Ma, S. Chen, and Y. Wang, "Fast and Accurate Cardinality Estimation by Self-Morphing Bitmaps," *IEEE/ACM Transactions on Networking*, 2022.

[27] C. Ma, H. Wang, O. O. Odegbile, and S. Chen, "Virtual Filter for Non-duplicate Sampling," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–11.

[28] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the Analysis of a Near-optimal Cardinality Estimation Algorithm," in *Discrete Mathematics and Theoretical Computer Science*. Discrete Mathematics and Theoretical Computer Science, 2007, pp. 137–156.

[29] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 2013, pp. 683–692.

[30] Y. Wang, H. Wang, C. Ma, and S. Chen, "Supporting Real-time Networkwide T-Queries in High-speed Networks," https://www.dropbox.com/s/yjw7zh7epzec4ku/icdcsfullversion.pdf?dl=0, 2022.

[31] 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 *KDD '20: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, August 2020, pp. 1015–1025.

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

[33] UCSD., "Caida anonymized 2018 internet traces," https://catalog.caida.org/details/dataset/passive_2018_pcap, 2018.