I Know What You Did Last Summer: Network Monitoring using Interval Queries

NIKITA IVKIN, Amazon, USA *
RAN BEN BASAT, Harvard University, USA
ZAOXING LIU, Carnegie Mellon University, USA
GIL EINZIGER, Ben Gurion University, Israel
ROY FRIEDMAN, Technion, Israel
VLADIMIR BRAVERMAN, Johns Hopkins University, USA **

Modern network telemetry systems collect and analyze massive amounts of raw data in a space efficient manner. These require advanced capabilities such as drill down queries that allow iterative refinement of the search space. We present a first integral solution that (*i*) enables multiple measurement tasks inside the same data structure, (*ii*) supports specifying the time frame of interest as part of its queries, and (*iii*) is sketch-based and thus space efficient. Namely, our approach allows the user to define both the measurement task (e.g., heavy hitters, entropy estimation, count distinct, etc.) and the time frame of relevance (e.g., 5PM-6PM) at query time. Our approach provides accuracy guarantees and is the only space-efficient solution that offers such capabilities. Finally, we demonstrate how our system can be used for accurately pinpointing the start of a realistic DDoS attack.

CCS Concepts: • **Networks → Data path algorithms**; *Network algorithms*;

Keywords: Interval queries; sliding window; sketches; L2 heavy hitters; entropy

ACM Reference Format:
Nikita Ivkin, Ran Ben Basat, Zaoxing Liu, Gil Einziger, Roy Friedman, and Vladimir Braverman. 2019. I Know What You Did Last Summer: Network Monitoring using Interval Queries. In *Proc. ACM Meas. Anal. Comput. Syst.*, Vol. 3, 3, Article 61 (December 2019). ACM, New York, NY. 28 pages. https://doi.org/10.1145/3366709

1 INTRODUCTION

Network monitoring is at the heart of many networking protocols and network functions, such as traffic engineering [14], load balanced routing [53, 66], attack and anomaly detection [12, 33, 55, 62], and forensic analysis [46, 63]. Over the years, a large number of metrics have been defined, including *per-flow frequency* [32, 65], *heavy hitter detection* [11, 28], *distinct heavy hitters* [35], *cardinality estimation* [27, 37, 41, 42], *change detection* [39], *entropy estimation* [6, 59], *quantiles* [44, 45, 67] and more. It is often infeasible to compute these statistics at line-rate due to limited memory and computing resources on network devices. Thus, approximate results are usually a reasonable choice [65]. In particular, sketches for the L_2 norm can efficiently estimate all the above measurement tasks. Further, the seminal UnivMon work [51] captures all these metrics within a single L_2-norm based sketch.

*This work was done while the author was at Johns Hopkins University, USA.
**This work was done [in part] while the author was visiting the Simons Institute for the Theory of Computing.

2476-1249/2019/12-ART61 $15.00
https://doi.org/10.1145/3366709

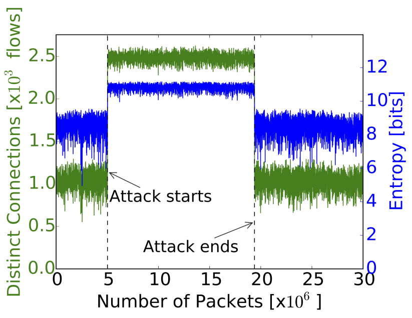


Fig. 1. The impact of a DDoS attack on the Distinct Count (Green), and the Entropy (Blue) metrics.

The current network conditions are typically more significant for network applications than old historical data. For example, in Figure 1 an attack detection application may wish to identify the beginning and the end of a DDoS attack. Such detection is possible by monitoring the number of distinct flows, or the Entropy, and by identifying a significant change in these metrics. Yet, such an attack cannot be detected if the frame of reference is too big. The *sliding window* model [6, 8, 30] maintains the measurement with respect to a fixed-sized window of streamed data.

However, it is unclear what the 'right' window size is, and once the window size is determined we have no visibility into smaller intervals contained within that window. For example, measurement over a 1-minute window may fail to detect a sub-second pattern (e.g., a microburst [26]).

Further, in typical network security scenarios, such as a SIEM-SOC setting, whenever there is a suspicion for a network anomaly, analyst teams must review long logs of recorded data. Generating these logs requires tremendous amounts of storage, whose accumulated maintenance costs are mounting. Worse yet, during an attack event, analysts need to manually scrutinize the logs when they scramble to identify the attack and offer counter-measures quickly. Hence, the ability to automatically pinpoint the beginnings and endings of anomaly events is desired. Such an ability allows log files to be pruned only to maintain important events, and analysts can be rapidly directed to the relevant parts of the log files when needed. Another benefit of automatic pinpointing of such beginnings and endings of unusual behavior is in unsupervised learning; such real-time pinpointing can give continuous feedback to the learning process.

In this context, the *Interval Query* (IQ) [9, 60] model generalizes sliding window to provide visibility to any interval inside the window. In the IQ model, the user provides the interval of reference at query time, and the metric is estimated over that interval. It is suitable to accurately pinpoint the beginning of an attack, or an important traffic pattern, and for identifying short patterns. For instance, we can use the IQ model to perform drill-down queries and detect a small interval that contains the start of the attack (in Figure 1).

In this paper, we present a measurement framework that supports queries with L_2 guarantees in the IQ model. To the best of our knowledge, our work is the first to study the IQ model with L_2 guarantees. By supporting L_2, we can port general sketches (e.g., UnivMon [51]), that support a variety of measurement queries. Previous work [9, 60] suggests algorithms that are limited to L_1 guarantee. In particular, this limitation prevents existing approaches from supporting the Entropy and Distinct flow count which are useful for attack detection.

In designing our algorithms, we explore the Exponential Histogram (EH) [30] and Smooth Histogram (SH) [22] in the IQ model. We gradually build our framework with increasingly sophisticated algorithms that gradually improve on each other. Such a presentation exposes the design space, trade-offs, and challenges in developing such a system. In particular, we design new algorithms in the IQ model for L_2 heavy hitters and extend them to support a broader class of functions with UnivMon [51].

Next, we study an attack localization problem that pinpoints the beginning of a network anomaly or an attack, using our framework. While the ability to automatically detect such anomalies is not new, the ability to automatically localize the attack with more fine-grained beginning and end is novel. Our algorithm relies on interval queries on the entropy of the flows, and our framework is the first to serve such queries in the IQ model.

In summary, we make the following contributions:

- We introduce the first set of measurement algorithms with an L_2 estimation guarantee in the IQ model, using sub-linear space.
- We extend a universal sketch (UnivMon [51]) to support a wide variety of queries in the IQ model, e.g., L_2 heavy hitters, entropy estimation, and distinct flow counting.
- We exemplify the usefulness of our framework by introducing an attack localization algorithm that automatically pinpoints the beginning of a traffic anomaly.
- We evaluate our framework using a set of real-world Internet traces [2], and demonstrate good accuracy using acceptable memory space. In particular, we evaluate our attack localization algorithm by simulating a DDoS attack scenario within a real-world Internet trace. We showcase the ability to pinpoint the beginning of the attack automatically. To the best of our knowledge, we are the first to provide such a capability.

Paper Roadmap: The rest of this paper is organized as follows: The basic terminology and problem statement are provided in Section 2. We present our novel algorithms with their proofs and analysis in Section 3. The evaluation results are described in Section 4. We conclude with a discussion in Section 5.

2 BACKGROUND

2.1 Streaming Computation Models

Streaming Model: The streaming model as a computational model for processing large data sets was first formally introduced in [5]. It targets the applications where the data items arrive sequentially in a streaming fashion, where each item can only be accessed at arrival time. More formally, given a stream of updates $S = \{s_1, \ldots, s_m\}$, where $s_i \in \{1, \ldots, N\}$, the goal is to compute a target function $F(S)$ while using space which is sublinear in m and N. Space constraints often render the exact computation of a function infeasible; instead, streaming algorithms usually provide an (ε, δ)-approximation scheme. That is, randomized algorithms that return $\hat{F}(S) \in (1 \pm \varepsilon)F(S)$ with probability at least $1 - \delta$. The stream of updates (S) can be a list of updates per time unit for time-based intervals (or a single update per time unit for packet-based intervals). The target function F can be the frequency of a certain identifier (e.g., per-flow frequency), the distinct count of identifiers, the entropy, and so forth. The notation $F(S)$ refers to calculating the function F over the entire stream of updates (S), while the notation $S(t_1, t_2)$ is a substream of updates that starts at t_1 and ends at t_2. For more details on the streaming model and its variations refer to [4, 57].

Sliding Window Model: In many applications, the stream of data is considered to be infinite, and a target function should be computed only on the last n updates and "forget" older ones. The *Sliding Window* model [30] addresses the pool of such problems. Formally, given a stream of updates $S = \{s_1, \ldots, s_t, \ldots\}$ and $s_i \in \{1, \ldots, N\}$, the goal of a sliding window algorithm is to report $F(t - n, t) = F(S(t - n, t)) = F(\{s_{t-n}, \ldots, s_t\})$ at any given moment t. Similarly, the algorithm should use space sublinear in n and N and follow the approximation scheme

Table 1. Notations and abbreviations.

N	dictionary size	SW	Sliding Window model
n	window size	IQ	Interval Query model
(t_1, t_2)	query interval	SH	Smooth Histograms
t	current moment	EH	Exponential Histograms
f_i	frequency of i		

$\hat{F}(t-n,t) \in (1 \pm \varepsilon)F(t-n,t)$. The numbers t, $t-n$ are natural numbers that represent the start and end of the interval in some units. The unit can be a time unit (e.g., seconds) or simply packets.

Interval Query Model: In this work, we address typical measurement tasks in the IQ model. First considered in [49], its goal is estimating a function over the interval (t_1, t_2) (of the stream S) that is specified at query time. Given a stream of updates S, the goal of an algorithm in the IQ model is to compute $F(t_1,t_2) = F(S(t_1,t_2))$ at any moment t, and any given interval $(t_1,t_2) \subset (t-n,t)$, while using space sublinear in n and N. In section 3.1, we show that achieving approximation $\hat{F}(t_1,t_2) = (1 \pm \varepsilon)F(t_1,t_2)$ is infeasible for some functions as it requires $\Omega(n)$ bits of memory. In this paper, we call an IQ algorithm (ε,δ)-approximate if it returns $\hat{F}(t_1,t_2) = F(t_1,t_2) \pm \varepsilon F(t_1,t)$ with probability at least $1-\delta$. That is, the allowed error is an ϵ fraction of the value of the function when applied on the suffix (t_1,t) and not only on (t_1,t_2). Specifically, this means that if t_2 is the current time, we get a multiplicative $(1+\varepsilon)$-approximation for a t_1-sized window whose size is given at query time.

2.2 L_1 and L_2 Heavy Hitters

Finding heavy hitters in streaming data is a well-studied problem in the analysis of large datasets; optimal or nearly optimal results were achieved in different models [15–17, 21, 25, 29, 52, 54, 67]. In this paper, we give a brief overview of the heavy hitters problem, including the formal problem statement and major differences between L_1 and L_2 settings. For more details on the problem, please refer to [4, 28, 57].

Item i is an (ε, L_p)-heavy hitter in the stream $S = \{s_i\}_{i=1}^m$, $s_i \in [N] = \{1,\dots,N\}$, if $f_i \geq \varepsilon L_p(f)$, where $f_i = \#\{j | s_j = i\}$ is the number of occurrences of item i in the stream S, and $L_p(f) = \sqrt[p]{\sum f_i^p}$ is the L_p norm of frequency vector f, and j-th coordinate in the vector equals f_j. An approximation algorithm for L_p heavy hitters returns all the items that appear at least εL_p times and no item that appears less than $\frac{\varepsilon}{2} L_p$ times and errs with probability at most δ. It was shown in [7, 24] that for $p > 2$ any algorithm will require the space at least polynomial in m and N. Therefore, the central interest is around finding L_1 and L_2 heavy hitters. Note that finding L_2 is provably more difficult compared to L_1 heavy hitters. While to be an (ε, L_1)-heavy hitter an item needs to appear in a constant fraction of the stream updates, in some cases to be an (ε, L_2)-heavy hitter the item can appear just in $O(1/\sqrt{N})$ fraction of updates. Note that to catch such L_1 heavy hitters using uniform sampling, one will need to sample at most $O(1/\varepsilon^2)$ items while catching L_2 heavy hitters will require the number of samples to be polynomial in N. Moreover, any L_2 algorithm can find all L_1 heavy hitters while the opposite is not always the case. The L_1 heavy hitters problem has optimal algorithms in both the cash register [52, 54] and turnstile [29] streaming models and was considered in both sliding windows [30] and Interval Query [9] computational models. The L_2 heavy hitters problem has tight results for both cash register [17] and turnstile [25] streaming models.

Recent results on L_2 heavy hitters in the sliding window were shown to be space optimal [21]; however, to the best of our knowledge, the problem has not been considered in the Interval Query

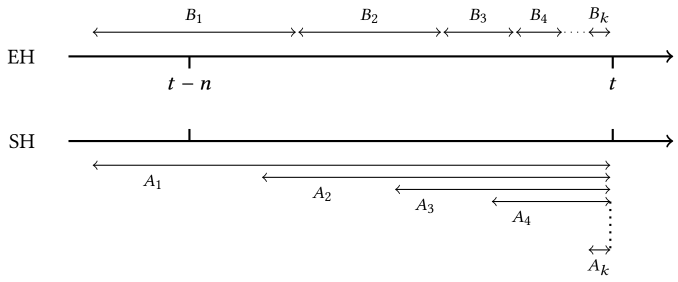


Fig. 2. Interval (bucket) structure for EH and SH.

model. In this work, we consider the following approximation L_2 heavy hitters problem in the Interval Query model.

Definition 2.1 ((ε, L_2)-heavy hitters problem in IQ). For $0 < \varepsilon < 1$ and a query $(t_1, t_2) \subseteq (t-n, t)$ given at time t, output a set of items $T \subset [N]$, such that T contains all items with $f_i(t_1, t_2) \geq \varepsilon L_2(t_1, t)$ and no items with $f_i(t_1, t_2) \leq \frac{\varepsilon}{2} L_2(t_1, t)$.

Definition 2.1 is a natural extension of heavy hitters to the IQ model but the error parameter (ϵ) applies to $L_2(t_1, t)$. In other words, the error is proportional to the L_2 norm over a t_1 sized window, regardless of the selected interval within that window. While we may want to apply ϵ to $L_2(t_1, t_2)$, we cannot do so in sublinear space (see Theorem 3.1). The error definition implies that for a given ϵ we may fail to detect any heavy hitter when the interval is too small. That is, when we reduce ϵ we gain the ability to monitor shorter intervals. In [64] authors suggest a similar model but approximate the function $F(t_1, t_2)$ with a mixture of relative $\varepsilon F(t_1, t_2)$ and additive Δ error components. For the sake of completeness we compare our results with [64] in Appendix B.

2.3 Sliding Window Frameworks

There is a strong connection between the Interval Query (IQ) and Sliding Window (SW) models: any algorithm that solves the problem in the IQ model can answer SW queries as well; one only needs to query the largest permitted interval, i.e., $(t_1, t_2) = (t-n, n)$. Therefore, we expect the current SW approaches to be useful for understanding the IQ model. Further, we introduce some background on SW and (ε, L_2)-heavy hitters algorithms in it.

Currently, two general SW frameworks are known: Exponential Histogram [30] and Smooth Histogram [22]. We now provide a brief overview of the core techniques of those frameworks.

2.3.1 *Exponential Histograms (EH).*

In [30], the authors suggest to break the sliding window $W = (t-n, n)$ into a sequence of k non-overlapping intervals $B_1, B_2, \ldots, B_k$, as depicted in Fig. 2. Window W is covered by $\cup_{i=1}^{k} B_i$, and contains all B_i except the first one. Then, if a target function f admits a composable sketch, maintaining such a sketch on each bucket can provide us with an estimator for f on a window $W' = \cup_{i=2}^{k} B_i$. Note that $f(W)$ is "sandwiched" between $f(W')$ and $f(B_1 \cup W')$. Therefore, a careful choice of each bucket endpoints provides control over the difference between $f(W)$ and $f(W')$, thereby making $f(W')$ a good estimator for $f(W)$. As the window slides, new buckets are introduced, expired buckets are deleted, and buckets in between are merged. The EH approach admits non-negative, polynomially bounded functions f which in turn enable a composable sketch and are weakly additive, i.e., $\exists C_f \geq 1$, such that $\forall S_1, S_2$:

$$f(S_1) + f(S_2) \leq f(S_1 \cup S_2) \leq C_f(f(S_1) + f(S_2)).$$

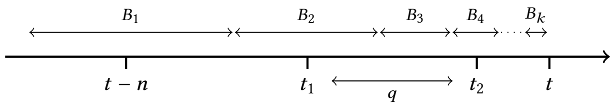


Fig. 3. Interval query in the prism of EH.

For such functions, [30] can ensure that $k = O(C_f^2 \log n)$ by maintaining two invariants:

$$f(B_j) \leq \frac{C_f}{k} \sum_{i=j+1}^{k} f(B_i) \quad \text{and} \quad f(B_{j-1}) + f(B_j) \geq \frac{1}{k} \sum_{i=j+1}^{k} f(B_i)$$

.

2.3.2 *Smooth Histograms (SH).*

In [22], the authors relax the weak additivity to more general property of smoothness:

$$\exists 0 < \beta \leq \alpha \leq \varepsilon \;\; \forall S_1, S_2, S_3 : (1-\beta) f(S_1 \cup S_2) \leq f(S_2) \Rightarrow (1-\alpha) f(S_1 \cup S_2 \cup S_3) \leq f(S_2 \cup S_3).$$

Additionally, in SH buckets $A_1, \ldots, A_k$ overlap; therefore, [22] extends the class of admitted target functions even further by relaxing the composability requirement. Similarly to [30], $f(W)$ is "sandwiched" between $f(A_1)$ and $f(A_2)$, see Fig. 2. The memory overhead is $O(\frac{1}{\beta} \log n)$ and the maintained invariants are $(1-\alpha) f(A_i) \leq f(A_{i+1})$ and $f(A_{i+2}) < (1-\beta) f(A_i)$.

2.4 Interval queries on EH and SH

The IQ model is a generalization of the SW model, with additional drill down into the monitoring window. However, in this paper, we suggest using similar building blocks as the SW model. To illustrate the idea we refer the reader to Fig. 3 which depicts query interval $q = (t_1, t_2)$ and buckets of Exponential Histogram with window of size n. Note that q is "sandwiched" between $B_2 \cup B_3 \cup B_4$ and B_3, while $f(\cup_{j=2}^{k} B_j) = (1 \pm \varepsilon) f(t_1, t)$ and $f(\cup_{j=4}^{k} B_j) = (1 \pm \varepsilon) f(t_2, t)$. Intuitively, one can expect that $f(t_1, t_2)$ can be approximated by $f(B_2 \cup B_3)$ with an additive error of $\pm \varepsilon f(t_1, t)$. A similar approach can be applied to Smooth Histograms if the sketches preserve approximation upon subtraction. We later show this formally.

3 INTERVAL ALGORITHMS

In this section, we introduce our algorithms that handle interval queries. We begin by giving a lower bound on the required number of memory bits that is needed for any algorithm in the IQ model. Then, we utilize the tools of Exponential Histogram and Smooth Histogram to design our L_2 Heavy Hitter algorithms in the IQ model. We start our first L_2 heavy hitter algorithm by adopting SH into the IQ model and gradually improve it with EH and enhanced SH. After careful analysis, we extend our approach to support a broad spectrum of functions (that UnivMon supports) and time-based interval queries.

3.1 Lower bound

Definition 2.1 suggests that our algorithm can have an error proportional to $t - t_1$, i.e., all elements from the start of the interval and until the current time. Intuitively, one could hope for an algorithm whose error only depends on the interval elements themselves (i.e., on $L_2(t_1, t_2)$). However, the following lower bound shows that no such sublinear algorithm exists, which motivates our problem goal.

THEOREM 3.1. *Any algorithm which maintains a sketch over the stream and at any moment t: $\forall t_1, t_2 \in (t-n, t)$ outputs $\hat{L}_2(t_1, t_2) = (1 \pm \varepsilon) L_2(t_1, t_2)$ requires $\Omega(n)$ bits of memory.*

Proof. The proof uses a reduction from the INDEX problem in communication complexity. Alice is given a string $x = \{0,1\}^n$ and Bob gets an index $k \in \{1,\dots,n\}$. Alice sends a message to Bob, and he should report the value of x_k. Bob can err with probability at most δ. A known lower bound on the size of the message, even for randomized algorithms, is $\Omega(n)$ [47].

Suppose there exists an algorithm that maintains a sketch over the stream and at any moment t, for any interval $(t_1, t_2) \subset (t-n, t)$, outputs $\hat{L}_2(t_1,t_2) = (1 \pm \varepsilon) L_2(t_1,t_2)$, while using only $o(n)$ bits of space. Such an algorithm can distinguish between L_2 and $L_2(1+3\varepsilon)$; denote $p = (1+3\varepsilon)^2$. In this case, Alice encodes the string x as a stream of updates $f(x_1), f(x_2), \dots, f(x_n)$, where $f(0) = 1,\dots,1$ and $f(1) = 1,2,\dots,p$ with both consisting of p updates. Therefore, the entire stream has length pn, $L_2(f(0)) = \sqrt{p}$, and $L_2(f(1)) > p$. Hence, $\frac{L_2(f(1))}{L_2(f(0))} > \sqrt{p}$ and the algorithm can distinguish them. After running the stream through the algorithm, Alice sends the content of the data structure to Bob. Bob queries interval $(t - pn + p(k-1), t - pn + pk)$ and due to approximation guarantees of the algorithm can infer whether $x_k = 0$ or 1. Therefore the INDEX problem was resolved with a message size of $o(n)$ bits, which contradicts the $\Omega(n)$ lower bound. □

3.2 Smooth Histogram based algorithm

Braverman, Gelles, and Ostrovsky, in [20], presented the first L_2 heavy hitter algorithm in the SW model. They apply the SH framework described earlier to get a $(1+\frac{\varepsilon}{2})$-approximation of L_2 norm. In addition, each bucket of the SH maintains an instance of the Count Sketch algorithm [25].

To understand the underlying intuition, refer to Figure 2. If item i is at least (ε, L_2)-heavy on the window $(t-n, t)$, then it must have appeared at least $\frac{\varepsilon}{2} L_2(t-n, n)$ times in the bucket A_2, otherwise $L_2(A_2) > (1-\frac{\varepsilon}{2}) L_2(t-n,t)$, which contradicts to the guarantee of the SH for L_2. Therefore, this item will be detected and reported by the Count Sketch algorithm associated with the bucket A_2.

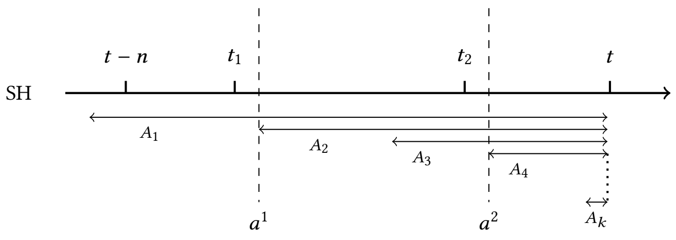


Fig. 4. Query intuition for the Algorithm 1.

Our algorithm utilizes a similar technique and maintains:

(1) *SH* for L_2 with $(\alpha, \beta) = \left(\frac{\varepsilon^2}{2^7}, \frac{\varepsilon^4}{2^{15}}\right)$.
(2) Count Sketch on each bucket of *SH*.

Later we will show that *SH* for L_2 with given (α,β) for any (t_1,t_2) returns $L_2(t_1,t_2) \pm \frac{\varepsilon}{2} L_2(t_1,t)$. Then, using similar argument, Count Sketch will detect all items heavy on the interval (t_1,t_2). Next we will go over the query process depicted on Figure 4. When querying with an interval (t_1,t_2), our algorithm finds the largest SH suffix (a^1, t) contained in (t_1, t) and the largest suffix (a^2, t) contained in (t_2, t). It then calculates Count Sketch of interval (a^1, a^2) as the difference of two Count Sketches: (a^1, t) and (a^2, t). Recall that Count Sketch is a linear operator on the frequency vector and thus

$$CS_{(a^1,a^2)} = CS_{(a^1,t)} - CS_{(a^2,t)}.$$

The frequencies on the interval (t_1,t_2) can be approximated by $CS_{(a^1,a^2)}$, as we will show later. Therefore, intuitively, if an item is heavy on the interval (t_1,t_2) it will be reported by $CS_{(a^1,a^2)}$. Detailed pseudo-code is presented in Algorithm 1.

First, we prove that the described technique can provide the desired approximation for $L_2(t_1, t_2)$.

Lemma 3.2. *Let A be an SH construction for (α, β)-smooth L_2 and $A_i = (a_i, t)$ are the suffixes of A (see Fig. 2). Then for any interval (t_1, t_2):*

$$L_2(a^1, a^2) = L_2(t_1, t_2) \pm \sqrt{2\alpha} L_2(t_1, t),$$

where $a^1 = \min a_i \geq t_1$ and $a^2 = \min a_i \geq t_2$.

Proof. From SH invariant [22] $L_2(a^1, t) > (1 - \alpha) L_2(t_1, t)$:

$$L_2^2(t_1, t) \geq L_2^2(t_1, a^1) + L_2^2(a^1, t) \Rightarrow L_2(t_1, a^1) \leq \sqrt{2\alpha} L_2(t_1, t)$$

and similarly for interval (t_2, a^2):

$$L_2(t_2, a^2) \leq \sqrt{2\alpha} L_2(t_2, t) \leq \sqrt{2\alpha} L_2(t_1, t).$$

Due to triangle inequality and monotonicity of L_2:

$$L_2(t_1, t_2) \leq L_2(t_1, a^1) + L_2(a^1, t_2) \leq L_2(t_1, a^1) + L_2(a^1, a^2),$$

$$L_2(a^1, a^2) \geq L_2(t_1, t_2) - \sqrt{2\alpha} L_2(t_1, t).$$

Repeating the argument for the different triangle:

$$L_2(t_1, t_2) + L_2(t_2, a^2) \geq L_2(t_1, a^2)$$

$$L_2(t_1, t_2) \geq L_2(t_1, a^2) - L_2(t_2, a^2) \geq L_2(a^1, a^2) - L_2(t_2, a^2)$$

$$L_2(a^1, a^2) \leq L_2(t_1, t_2) + \sqrt{2\alpha} L_2(t_1, t). \quad \square$$

Algorithm 1 L_2 heavy hitter algorithm based on [20]

```
1:  function Update(item)
2:      maintain $SH$ for $L_2$ with $(\alpha, \beta) = \left(\frac{\varepsilon^2}{2^7}, \frac{\varepsilon^4}{2^{15}}\right)$
3:      on each bucket $A_i$ maintain Count Sketch $CS_i$
4:  function Query($t_1, t_2$)
5:      $a^1 = \min a_i \geq t_1$ and $a^2 = \min a_i \geq t_2$
6:      subtract sketches $CS = CS_{(a^1,t)} - CS_{(a^2,t)}$
7:      query $L_2$ of suffix: $\hat{L}_2(a^1, t) = SH$.query$(a^1, t)$
8:      for each item $i$ in $CS_{(a^1,t)}$.heap do
9:          $\hat{f}_i(t_1, t_2) = CS$.estimateFreq$(i)$
10:         if $\hat{f}_i(t_1, t_2) > \frac{3\varepsilon}{4} \hat{L}_2(a^1, t)$ then report $\left(i, \hat{f}_i(t_1, t_2)\right)$
```

Theorem 3.3. *Algorithm 1 solves (ε, L_2)-heavy hitters in the IQ model (Definition 2.1) using $O\left(\varepsilon^{-6} \log^3 n \log \delta^{-1}\right)$ memory bits.*[1]

Proof. From the definition of a^1 and a^2 in line 5 of Algorithm 1:

$$\forall i : f_i(a^1, a^2) = f_i(t_1, t_2) - f_i(t_1, a^1) + f_i(t_2, a^2).$$

Note that $f_i(t_1, a^1) \leq L_2(t_1, a^1)$ and $f_i(t_2, a^2) \leq L_2(t_2, a^2)$, then from Lemma 3.2 and due to the choice of SH parameters:

$$f_i(t_1, a^1) \leq \frac{\varepsilon}{8} L_2(t_1, t), \quad f_i(t_2, a^2) \leq \frac{\varepsilon}{8} L_2(t_1, t).$$

[1]Here and further, we omit terms $\log \varepsilon^{-1}$ and $\log \log n$ for compactness.

For a given stream S Count Sketch approximates the counts as follows: $\hat{f}_i(S) = f_i(S) \pm \frac{\varepsilon}{8} L_2(S)$ and its heap contains all $(\frac{\varepsilon}{8}, L_2)$-heavy hitters. Thus, the sketch $CS = CS_{(a^1,t)} - CS_{(a^2,t)}$ will estimate the count of any queried item i as

$$\hat{f}_i(a^1, a^2) = f_i(a^1, a^2) \pm \frac{\varepsilon}{8} L_2(t_1, t) = f_i(t_1, t_2) \pm \frac{\varepsilon}{4} L_2(t_1, t),$$

where the last equality follows from the derivations above.
To prove the correctness of the algorithm, we need to show that:

(1) every (ε, L_2)-heavy on (t_1, t_2) item will appear in the heap of $CS_{(a^1,t)}$ (line 8) and will be reported (line 10).
(2) no items $f_j(t_1, t_2) \le \frac{\varepsilon}{2} L_2(t_1, t)$ will be reported.

Note that if item i is (ε, L_2)-heavy on (t_1, t_2), then $f_i(t_1, t_2) > \varepsilon L_2(t_1, t)$ which implies $f_i(a^1, a^2) > \frac{7\varepsilon}{8} L_2(t_1, t)$ and $f_i(a^1, t) > \frac{7\varepsilon}{8} L_2(t_1, t)$. The latter one implies that item i will be in the heap of $CS_{(a^1,t)}$. The former one given that

$$\hat{f}_i(a^1, a^2) > f_i(a^1, a^2) - \frac{\varepsilon}{8} L_2(t_1, t) > \frac{3\varepsilon}{4} L_2(t_1, t)$$

and $\hat{L}_2(a^1, t) = (1 \pm \frac{\varepsilon^2}{2^7}) L_2(t_1, t)$ guarantees that heavy item i will be reported in line 10.
Similarly for the light items:

$$\hat{f}_i(a^1, a^2) < f_i(a^1, a^2) + \frac{\varepsilon}{8} L_2(t_1, t) < \frac{3\varepsilon}{4} L_2(t_1, t),$$

therefore no item j with $f_j(t_1, t_2) \le \frac{\varepsilon}{2} L_2(t_1, t)$ will be reported in line 10.

According to Theorem 3 from [22] (see Appendix Theorem A.1), the SH approach requires $O(\frac{1}{\beta} g(\varepsilon, \frac{\delta\beta}{n}) \log n)$ bits of memory. Here, $g(\varepsilon, \delta)$ is the amount of memory needed for a sketch to get ε approximation of target function with failure probability at most δ. Note that to avoid mistaken deletions of suffixes in the SH construction, every target function sketch should succeed. Therefore L_2 sketch should fail with probability at most $O(\frac{\delta}{n})$. At the same time, Count Sketch is needed only at the moment of a query, therefore it should fail with probability at most $O(\frac{\delta}{\log n})$. Each amplified L_2 sketch, according to [5], requires $O(\frac{1}{\varepsilon^2} \log^2 n)$ bits of space – same as the amount of memory that is needed for each Count Sketch. Thus, in total, the algorithm requires $O(\varepsilon^{-6} \log^3 n \log \delta^{-1})$ memory bits. □

3.3 Exponential Histogram based algorithm

A natural question to ask is whether the Exponential Histogram framework can be used instead of Smooth Histograms. Recall that the core idea is to maintain $(1 \pm \varepsilon)$ approximation for the $L_2(t-n, t)$, which in case of Algorithm 1 guaranteed that we do not loose the heavy items. Target function $L_2^2(\cdot)$ admits a composable sketch [5] and is weakly additive with $C_f = 2$. According to Theorem 7 of [30] (see Appendix Theorem A.2), an admittable target function f can be estimated with the relative error

$$E_r \le \frac{(1+\varepsilon)}{k} C_f^2 + C_f - 1 + \varepsilon,$$

where ε is the relative error of the L_2^2 sketch and k is a parameter of the EH framework. Note that for $C_f = 2$, no k can get an error better than $E_r = 1 + o(1)$, which only implies a 2-approximation of the L_2-norm on the sliding window. The IQ model is more general than SW. Therefore, the same idea would not work out of the box, as we can not get $(1 \pm \varepsilon)$ approximation for $L_2(t-n, t)$. Further, we suggest two ways to overcome this issue. First, we suggest the following decoupling tweak to the weak additivity requirement:

$$\exists C_f, C_f' \ \forall S_1, S_2 : f(S_1 \cup S_2) \le C_f f(S_1) + C_f' f(S_2).$$

Keeping the rest of the framework the same and repeating the argument as in Theorem 7 of [30], the relative error becomes:

$$E_r \le \frac{(1+\varepsilon)}{k} C_f^2 + C'_f - 1 + \varepsilon.$$

To find appropriate constants C_f and C'_f for L_2^2, note that for any positive integers a and b and any $\varepsilon \in (0,1)$:

$$(a+b)^2 \le \frac{2}{\varepsilon}a^2 + (1+\varepsilon)b^2 - \left(\frac{1}{\sqrt{\varepsilon}}a - \sqrt{\varepsilon}b\right)^2 \le \frac{2}{\varepsilon}a^2 + (1+\varepsilon)b^2.$$

Therefore, applying to L_2^2:

$$\forall S_1, S_2 : L_2^2(S_1 \cup S_2) \le \frac{1}{2\varepsilon}L_2^2(S_1) + (1+\varepsilon)L_2^2(S_2),$$

i.e., $C_f = \frac{2}{\varepsilon}$ and $C'_f = 1+\varepsilon$. Setting $k = O(\frac{1}{\varepsilon^3})$ gives a $(1+\varepsilon)$-approximation of L_2^2 on sliding windows using the EH framework. Note that the described tweak will work for any admittable function f for which $f(S_1 \cup S_2) \le C_f f(S_1) + C'_f f(S_2)$, as no other properties of L_2 were used in the proof.

Further, we argue that the same approximation can be achieved with a smaller k. We maintain EH histogram framework for $f = L_2^2$ as proposed in [30] with $C_f = 2$. However, when queried, we output $\sqrt{f} = L_2$ rather than $f = L_2^2$. Note that due to the triangle inequality, $\sqrt{f(S_1+S_2)} \le \sqrt{f(S_1)} + \sqrt{f(S_2)}$. Denote $t_0 = t - n$; then the core derivation for the relative error E_r from Theorem 7 of [30] (see Appendix Theorem A.2) can be rewritten as follows:

$$E_r = \frac{\sqrt{f(t_0,t)} - \sqrt{f(b_1,t)}}{\sqrt{f(t_0,t)}} \le \sqrt{\frac{f(t_0,b_1)}{f(t_0,t)}} \le \sqrt{\frac{f(b_0,b_1)}{f(b_1,t)}} \le \sqrt{\frac{C_f \sum_{i>0} f(B_i)}{kf(b_1,t)}} \le \sqrt{\frac{C_f f(b_1,t)}{kf(b_1,t)}} \le \sqrt{\frac{C_f}{k}}.$$

Setting $k = O(\frac{1}{\varepsilon^2})$ provides necessary ε-approximation for L_2 on the sliding window. Note, that it is a polynomial improvement in terms of ε compared to the tweak introduced earlier.

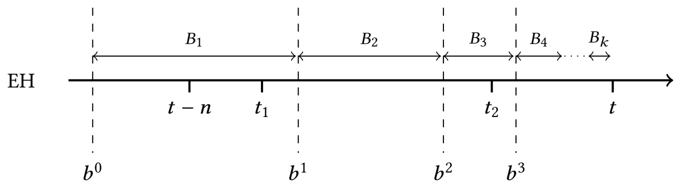


Fig. 5. Query intuition for the Algorithm 2.

Our algorithm utilizes this approach and maintains:

(1) *EH* for L_2 with $k = O(\frac{1}{\varepsilon^2})$ and $C_f = 2$.
(2) Count Sketch on each bucket of *EH*.

Later we will show formally that *EH* for L_2 with given k and C_f returns $L_2(t_1,t_2) \pm \frac{\varepsilon}{2}L_2(t_1,t)$. Then, following the similar argument Count Sketch will detect all the items which are heavy on the interval (t_1,t_2).

Here we will go over the query process depicted on Figure 5. When querying with an interval (t_1,t_2), EH based algorithm will find buckets $B_i = (b^0,b^1)$ and $B_j = (b^2,b^3)$ such that: $t_1 \in B_i$ and $t_2 \in B_j$, then it computes the Count Sketch $CS(b^1,b^3)$ as a sum of sketches on the buckets $B_{i+1},\dots,B_j$. We will show later that $CS(b^1,b^3)$ provides sufficient approximation for items' frequencies on interval (t_1,t_2), therefore it will detect items which are heavy on (t_1,t_2). Detailed pseudo-code is presented in Algorithm 2.

Lemma 3.4. *Let B be an EH construction for L_2^2 with parameters C_f, k and let $B_i = (b_i, b_{i+1})$ denote the buckets of B (see Fig. 2). Then for any interval (t_1, t_2):*

$$L_2(b^1, b^3) = L_2(t_1, t_2) \pm \sqrt{C_f/k}\, L_2(t_1, t),$$

where (b^0, b^1) is the bucket containing t_1 and (b^2, b^3) is the bucket containing t_2.

Proof. By the monotonicity of L_2^2 and the EH invariant:

$$L_2^2(t_1, b^1) \leq L_2^2(b^0, b^1) \leq \frac{C_f}{k} \sum_{b_i > b^0} L_2^2(B_i) \leq \frac{C_f}{k} L_2^2(b^1, t).$$

Repeating the argument for (t_2, b^3) and taking the square root:

$$L_2(t_1, b^1) \leq \sqrt{C_f/k}\, L_2(t_1, t) \quad \text{and} \quad L_2(t_2, b^3) \leq \sqrt{C_f/k}\, L_2(t_1, t).$$

Applying triangle inequality the same way as in the proof of Lemma 3.2 leads to the statement of the current lemma. □

Algorithm 2 L_2 heavy hitter algorithm based on EH

```
1: function Update(item)
2:     maintain $EH$ for $L_2^2$ with $k = O\left(\frac{1}{\varepsilon^2}\right)$ and $C_f = 2$.
3:     on each bucket $B_i$ maintain Count Sketch $CS_i$
4: function Query($t_1, t_2$ )
5:     find $B_i = (b^0, b^1)$ and $B_j = (b^2, b^3)$ s.t.: $t_1 \in B_i$ and $t_2 \in B_j$
6:     compute the union sketch $CS = \cup_{i+1 \leq l \leq j} CS_l$
7:     query $L_2$ of the suffix $(t_1, t)$: $\hat{L}_2(t_1, t) = EH.\text{query}(b^1, t)$
8:     for each heavy hitter $(i, \hat{f}_i)$ in $CS$ do
9:         if $\hat{f}_i > \frac{3\varepsilon}{4}\hat{L}_2(t_1, t)$ then report $(i, \hat{f}_i)$
```

Theorem 3.5. *Algorithm 2 solves (ε, L_2)-heavy hitters problem in IQ model (Definition 2.1) using $O(\varepsilon^{-4} \log^3 n \log \delta^{-1})$ bits of space.*

Proof. Due to Lemma 3.4 for given parameters C_f and k:

$$\forall i : f_i(b^1, b^3) = f_i(t_1, t_2) \pm \frac{\varepsilon}{16} L_2(t_1, t).$$

Approximation guarantees of Count Sketch CS can be rewritten as:

$$\forall i : \hat{f}_i(b^1, b^3) = f_i(b^1, b^3) \pm \frac{\varepsilon}{16} L_2(b^1, t).$$

Therefore, $\forall i : \hat{f}_i(b^1, b^3) = f_i(t_1, t_2) \pm \frac{\varepsilon}{8} L_2(t^1, t)$. The same lemma applied to interval (t_1, t), given $\frac{\varepsilon}{16}$-approximation of L_2 on each bucket B_i, leads to $\hat{L}_2(t_1, t) = (1 \pm \frac{\varepsilon}{8}) L_2(t_1, t)$. Thus, any heavy item with $f_i(t_1, t_2) \geq \varepsilon L_2(t_1, t)$ will be reported in line 9:

$$\hat{f}_i(b^1, b^3) \geq f_i(t_1, t_2) - \frac{\varepsilon}{8} L_2(t_1, t) \geq \frac{7\varepsilon}{8} L_2(t_1, t) \geq \frac{3\varepsilon}{4} \hat{L}_2(t_1, t).$$

Similarly, any item with $f_i(t_1, t_2) \leq \frac{\varepsilon}{2} L_2(t_1, t)$ not be reported in line 9:

$$\hat{f}_i(b^1, b^3) \leq \frac{5\varepsilon}{8} L_2(t_1, t) \leq \frac{3\varepsilon}{4} \hat{L}_2(t_1, t).$$

According to Theorem 7 from [30] (see Appendix Theorem A.2), the EH approach requires $O\left(k \cdot g\left(\varepsilon, \frac{\delta\beta}{n}\right) \cdot \log n\right)$ bits of memory, where $g(\varepsilon, \delta)$ is the amount of memory needed for a sketch

Table 2. Summary of the proposed L_2 heavy hitter algorithms in the IQ model.

Algorithm	Space complexity	Update time	Query time
Section 3.2: Smooth Histogram based Algorithm	$O(\varepsilon^{-6}\log^3 n \log\delta^{-1})$	$O(\varepsilon^{-4}\log n)$	$O(\varepsilon^{-2}\log n)$
Section 3.3: Exponential Histogram based Algorithm	$O(\varepsilon^{-4}\log^3 n \log\delta^{-1})$	$O(\varepsilon^{-2}\log n)$	$O(\varepsilon^{-3}\log n)$
Section 3.4: Improved Smooth Histogram based Algorithm	$O(\varepsilon^{-3}\log^2 n \log\delta^{-1})$	$O(\log n)$	$O(\varepsilon^{-2}\log n)$

to get a $(1+\varepsilon)$-approximation of the target function with a failure probability of at most δ. Hence, the memory required to maintain SH for L_2^2 is $O\left(\varepsilon^{-2}\log^3 n \log\delta^{-1}\right)$. In addition we maintain k instances of Count Sketch that should succeed simultaneously, it requires $O\left(\varepsilon^{-4}\log^3 n\log\delta^{-1}\right)$ bits of memory. Thus, in total, the algorithm requires $O\left(\varepsilon^{-4}\log^3 n\log\delta^{-1}\right)$ bits. □

3.4 Improved Smooth Histogram based algorithm

Recently, a new algorithm for finding L_2-heavy hitters in the SW model was introduced in [21]. The authors show a significant improvement in space complexity and provide a matching lower bound. The memory footprint of the solution proposed in [21] is $O(\frac{1}{\varepsilon^2}\log^2 n\log\frac{1}{\varepsilon\delta})$ bits, while previous result [20] needed at least $O(\frac{1}{\varepsilon^4}\log^3 n\log\frac{1}{\varepsilon\delta})$ bits. Although the new approach uses the SH framework, it differs conceptually in the way of catching the heavy hitters. Recall that [20] requires a $(1+\varepsilon)$-approximation of the L_2 to make sure that no heavy hitters are lost between neighboring buckets. [21] in contrary maintains SH for $L_2(t-n,t)$ with only a constant approximation and as before runs Count Sketch on each bucket of the histogram. It takes advantage of the fact that streaming L_2-heavy hitter algorithms can report a heavy hitter after seeing only $\frac{\varepsilon}{16}L_2$ instances, when it happens the item gets into the set of heavy hitter candidates. The frequency of each candidate is maintained in the separate SH framework with a constant approximation. Note, that the algorithm is required to return all items with frequency larger than $\varepsilon L_2(t-n,t)$ and no items with frequency lower than $\frac{\varepsilon}{2}L_2(t-n,t)$. Therefore a constant approximation for the frequency of each candidate and $L_2(t-n,t)$ is enough to test if the candidate is indeed a heavy hitter.

Additionally, to reduce the memory footprint of the algorithm even further [21] suggests to replace the Count Sketch algorithm with BPTree [17], to use shared randomness and the strong tracking argument from [18] to avoid union bound for all bucket sketches to succeed.

We reuse this approach for the IQ model and show that:

(1) maintaining SH for the count of each candidate preserves the desired constant approximation on intervals,
(2) due to Lemma 3.2, SH with constant approximation for $L_2(t-n,t)$ will also approximate $L_2(t_1,t_2)$ with additive term $\pm O(1)L_2(t_1,t)$.

The query procedure is only slightly different from our first Smooth Histogram based algorithm, as now in addition to finding the interval (a_1,a_2) in SH for estimation $L_2(t_1,t_2)$, one also need to query SH for the count of corresponding heavy hitter candidates. Detailed pseudo-code is presented in Algorithm 3.

Lemma 3.6. *Let A be an SH construction for an (α,β)-smooth sum function S and consider a binary stream that consists of zeros and ones. Denote by $A_i=(a_i,t)$ the suffixes of A (see Fig. 2). Then for any interval (t_1,t_2):*

$$S(a^1,a^2)=S(t_1,t_2)\pm\alpha S(t_1,t),$$

where $a^1=\min a_i\geq t_1$ and $a^2=\min a_i\geq t_2$.

Proof. From the SH invariant [22] it follows that:

$$S(a^1,t)>(1-\alpha)S(t_1,t),$$

$$S(t_1,t) = S(a^1,t) + S(t_1,a^1) \Rightarrow S(t_1,a^1) \leq \alpha S(t_1,t).$$

Repeating the argument for (t_2,a^2): $S(t_2,a^2) \leq \alpha S(t_2,t)$.
Therefore the statement of the lemma follows:

$$S(t_1,t_2) = S(a^1,a^2) + S(t_1,a^1) - S(t_2,a^2) \Rightarrow S(a^1,a^2) = S(t_1,t_2) \pm \alpha S(t_1,t). \quad \square$$

Algorithm 3 L_2 heavy hitter algorithm based on [21]

```
1:  function Init
2:      init $SH_{L_2}$ with $(\alpha,\beta) = \left(\frac{1}{10}, \frac{1}{200}\right)$
3:      init $CS_i$ Count Sketch on each $SH_{L_2}$ bucket $A_i$
4:      init $HH_p$ an array for potential heavy items
5:                  // if $i \in HH_p$ then $HH_p[i].SH$ tracks $f_i$ in SW
6:  function Update(item)
7:      update $SH_{L_2}$ and all $CS_i$
8:      if item $\in HH_p$ then update $HH_p$[item]$.SH$
9:      for all $i$ for each item $(j,\hat{f}_j)$ in $CS_i$.heap
10:         if $\hat{f}_j > \frac{3\varepsilon}{4} SH_{L_2}$.query$(A_i)$ and $j \notin HH_p$:
11:             $HH_p$.add$(j)$
12:             init $HH_p[j].SH$ with $(\alpha,\beta) = \left(\frac{\varepsilon}{16}, \frac{\varepsilon}{16}\right)$
13: function Query$(t_1,t_2)$
14:     $a^1 = \min a_i \geq t_1$ and $\hat{L}_2(a^1,t) = SH$.query$(a^1,t)$
15:     for each item $i \in HH_p$ do
16:         $a^1 = \min a_i \geq t_1$ and $a^2 = \min a_i \geq t_2$
17:         $\hat{f}_i(t_1,t) = HH_p[i].SH$.query$(a_1,t)$
18:         $\hat{f}_i(t_1,t_2) = \hat{f}_i(t_1,t) - HH_p[i].SH$.query$(a_2,t)$
19:         if $\hat{f}_i(t_1,t_2) > \frac{3\varepsilon}{4}\hat{L}_2(a^1,t)$ then report $(i,\hat{f}_i)$
```

Theorem 3.7. *Algorithm 3 solves (ε,L_2)-heavy hitters problem in IQ model (Definition 2.1) using $O(\varepsilon^{-3}\log^3 n \log \delta^{-1})$ bits of space.*

Proof. First, let us show that all items with $f_i(t_1,t_2) \geq \varepsilon L_2(t_1,t)$ will appear in HH_p. Denote $a_0 = \max a_i \leq t_1$ then

$$f_i(a^0,t_2) \geq f_i(t_1,t_2) \geq \varepsilon L_2(t_1,t) \geq \varepsilon L_2(t_1,t_2).$$

Therefore, by moment t_2, Count Sketch of the bucket (a^0,t_2) should have reported it in line 10 of Algorithm 3.

Now we argue that all heavy items will be reported in line 19. Let (a_0,t) be the first bucket of $HH_p[i].SH$ then we should consider two cases: $t_1 \geq a_0$ and $t_1 \leq a_0$.

If $t_1 \geq a_0$ then it follows from Lemma 3.6, that:

$$\hat{f}_i(t_1,t_2) \geq f(t_1,t_2) - \frac{\varepsilon}{16} f(t_1,t).$$

At the same time SH framework guarantees $\hat{L}_2(t_1,t) \leq 1.1L_2(t_1,t)$. Therefore, if $f_i(t_1,t_2) \geq \varepsilon L_2(t_1,t)$, then:

$$\hat{f}_i(t_1,t_2) \geq \frac{15}{16}\varepsilon L_2(t_1,t) \geq \frac{3}{4}\varepsilon \hat{L}_2(t_1,t)$$

Recall, that Count Sketch reports an item after seeing $\frac{\varepsilon}{16}L_2$ of its instances. Therefore, for $t_1 \leq a_0$ using the same lemma we can conclude that $\hat{f}_i(t_1,t_2) \geq f(t_1,t_2) - \frac{\varepsilon}{8} f(t_1,t)$, while the rest of computation is the same. Thus, all items with $f_i(t_1,t_2) \geq \varepsilon L_2(t_1,t)$ will be reported by the line 19 of Algorithm 3 .

Then from Lemma 3.6 for every non-heavy on (t_1, t_2) item with $f_i(t_1, t_2) \leq \frac{\varepsilon}{2} L_2(t_1, t)$, if $i \in HH_p$:

$$\hat{f}_i(t_1, t_2) \leq f_i(t_1, t_2) + \frac{\varepsilon}{16} f_i(t_1, t) \leq \frac{\varepsilon}{2} L_2(t_1, t) + \frac{\varepsilon}{16} L_2(t_1, t) \leq \frac{3\varepsilon}{4} \hat{L}_2(t_1, t),$$

and item i will not be reported in line 19.

According to Theorem 3 from [21] (see Appendix Theorem A.1), the modified SH approach requires $O(\frac{1}{\beta} g(\varepsilon, \delta) \log n)$ bits of memory. Algorithm 3 uses $\beta = O(1)$ for SH_{L_2}, with $g(\varepsilon, \delta) = O(\frac{1}{\varepsilon^2} \log^2 n)$ due to L_2 sketch and Count Sketch instances. Thus, SH_{L_2} requires $O(\frac{1}{\varepsilon^2} \log^2 n)$ bits of space, and have $O(\log n)$ buckets and each can potentially generate up to $O(\frac{1}{\varepsilon^2})$ items in HH_p. Therefore, in total, to track ε-approximation to frequencies of all potential heavy hitters, data structure spend $O(\frac{1}{\varepsilon^3} \log^3 n)$ bits of space. Summing the two derived quantities, the space complexity of Algorithm 3 is $O(\varepsilon^{-3} \log^3 n \log \delta^{-1})$ bits. □

Note that replacing Count Sketch in Algorithm 3 with BPTree [17] improves the space complexity by another $\log n$ factor.

Corollary 3.8. *There exists an algorithm that solves (ε, L_2)-heavy hitters problem in IQ model using space $O(\varepsilon^{-3} \log^2 n \log \delta^{-1})$ bits of space.*

In this work, we mainly focus on the trade-off between memory and precision. For brevity, we omit the details of computation for the update and query time as it is straight forward. In Table 2, we compare space complexity, update and query time of our algorithms.

3.5 Extending to a wider class of functions

Many streaming algorithms and frameworks use a L_2-heavy hitter algorithm as a subroutine. One of them is UnivMon [51], the framework which promotes recent results on universal sketching [19] in the field of network traffic analysis. The main power of the framework is its ability to maintain only one sketch for many target flow functions rather than an ad-hoc sketch per each function. The class of functions that can be queried is comprehensive and covers the majority of those used in practice, among examples are the L_0, L_2 norms and entropy. Therefore, L_2-heavy hitters is an essential step towards UnivMon in IQ model. For more details on universal sketching refer to [19]; here we will cover the necessary basics.

Given a function $g : \mathbb{N} \to \mathbb{N}$, the goal of universal sketching is to approximate $G = \sum_{i=1}^{n} g(f_i)$ by making one or several passes over the stream. Theorem 2 in [19] states: if $g(x)$ grows slower than x^2, drops no faster than sub-polynomially, and has predictable local variability, then there is an algorithm that outputs an ε-approximation to G, using sub-polynomial space and only one pass over the data. The algorithm consists of two main subroutines: (g, ε)-heavy hitters and Recursive Sketch. The first one finds all items i such that $g(f_i) \geq \varepsilon G$ together with an ε-approximation to $g(f_i)$. In [19], the authors show that if an item is (g, ε)-heavy then it is also $(L_2, \frac{\varepsilon}{h})$-heavy for sub-polynomial h; therefore, Count Sketch can be used to find (g, ε)-heavy items. Recursive Sketch was initially introduced in [43] and further generalized in [23]. It finds ε-approximation of G using a (g, ε)-heavy hitter algorithm as the black box, by recursively subsampling the universe, and by estimating the sum G of the subsampled stream.

In [19] (g, ε)-heavy hitter algorithm requires a subroutine that finds all items i such that $f_i(t_1, t_2) \geq \varepsilon L_2(t_1, t_2)$. However, due to the limitations of the IQ model all three proposed algorithms only find all items i such that $f_i(t_1, t_2) \geq \varepsilon L_2(t_1, t)$. Further, we adjust the argument from [19] to argue that one can use algorithms from previous sections to find (g, ε)-heavy hitter with a guarantee defined as follows:

Definition 3.9 ((ε, g)-heavy hitters problem in IQ). For $0 < \varepsilon < 1$ output set of items $T \subset [N]$, such that T contains all items with $g(f_i(t_1, t_2)) \geq \varepsilon G(t_1, t) = \varepsilon \sum_j g(f_j(t_1, t))$ and no items with $g(f_i(t_1, t_2)) \leq \frac{\varepsilon}{2} G(t_1, t)$.

Propositions 15 and 16 in [19] show that if function g is slow-jumping and slow-dropping, then there exists a sub-polynomial function H:

$$\forall x \leq y : g(y) \geq \frac{g(x)}{H(y)}, \ g(y) \leq \left(\frac{y}{x}\right)^2 y^{\alpha} H(y) g(x). \tag{1}$$

We can adjust the argument of Lemmas 17 and 18 in [19] to handle the heavy hitters with additive error.

Lemma 3.10. *Let $HH(\varepsilon, \delta)$ be an algorithm that solves (ε, L_2)-heavy hitters problem in IQ model (Definition 2.1), and g is a slow-jumping and slow-dropping function. Then $HH\left(\frac{\varepsilon}{2H(n)}, \delta\right)$ solves (ε, g)-heavy hitters problem in IQ model (Definition 3.9).*

Proof. Note that for any (g, ε)-heavy i:

$$g(f_i(t_1, t_2)) \geq \varepsilon \textstyle\sum_j g(f_j(t_1, t)).$$

Therefore, applying the second statement of Equation 1:

$$g(f_i(t_1, t_2)) \geq \sum_{f_j(t_1,t) \leq f_i(t_1,t_2)} \frac{\varepsilon g(f_i(t_1, t_2)) f_j^2(t_1, t)}{H(n) f_i^2(t_1, t_2)}$$

$$f_i^2(t_1, t_2) \geq \frac{\varepsilon}{H(n)} \sum_{f_j(t_1,t) \leq f_i(t_1,t_2)} f_j^2(t_1, t)$$

Similarly, applying the first statement of Equation 1:

$$g(f_i(t_1, t_2)) \geq \sum_{f_j(t_1,t) \geq f_i(t_1,t_2)} \frac{\varepsilon g(f_i(t_1, t_2))}{H(n)}.$$

Hence, there are at most $\frac{H(n)}{\varepsilon}$ items with $f_j(t_1, t) \geq f_i(t_1, t_2)$, and $HH\left(\frac{\varepsilon}{2H(n)}, \delta\right)$ will detect it. □

Further we argue that Recursive Sketch with (g, ε)-heavy hitter algorithm which finds all i such that $g(f_i(t_1, t_2)) \geq \varepsilon G(t_1, t)$ will return $\hat{G}(t_1, t_2) = G(t_1, t_2) \pm \varepsilon G(t_1, t)$.

Algorithm 4 Recursive GSum(S_0, ε) [19]

```
1: $H_1, \ldots, H_{\phi = O(\log n)}$ are pairwise independent 0/1 vectors
2: $S_j$ is a subsampled stream $\{s \in S_{j-1} : H_j(s) = 1\}$
3: Compute, in parallel, $HH_j = HH(S_j)$
4: Compute $Y_\phi = G_\phi(t_1, t_2) \pm \varepsilon G_\phi(t_1, t)$
5: for $j = \phi - 1, \ldots, 0$ do
6:     compute $Y_j = 2Y_{j+1} + \sum_{i \in HH_j} (1 - 2H_j(i)) \hat{g}(f_i)$
7: return $Y_0$
```

Theorem 3.11. *Let HH be an algorithm that finds all i such that $g(f_i(t_1, t_2)) \geq \varepsilon G(t_1, t)$ together with ε-approximation of $g(f_i(t_1, t_2))$. Then Algorithm 4 computes $\hat{G}(t_1, t_2) = G(t_1, t_2) \pm \varepsilon G(t_1, t)$ and errs with probability at most 0.3. Its space overhead is $O(\log n)$.*

Proof. Note that G_j is a G-sum computed over the subsampled stream S_j. According to line 4 of Alg. 4 $Y_\phi = G_\phi(t_1,t_2) \pm \varepsilon G_\phi(t_1,t)$. Our goal is to evaluate the error propagation from the top level of subsampling ϕ to the bottom one and show that $Y_0 = G(t_1,t_2) \pm \varepsilon G(t_1,t)$.

Consider a random variable $X_j = \sum_{i \in HH_j} g(f_i) + 2\sum_{i \notin HH_j} H_j(i)g(f_i)$, an unbiased estimator of $G_j(t_1,t_2)$ with variance bounded as: $Var(X_j) = \sum\limits_{i \notin HH_j} g^2(f_i) \le \varepsilon G_j(t_1,t)\sum g(f_i) \le \varepsilon G_j^2(t_1,t)$ by the definition of HH_j and monotonicity of G. Therefore, by conditioning on HH_j success and Chebyshev inequality:

$$\Pr(|X_j - G_j(t_1,t_2)| \ge \varepsilon' G_j(t_1,t)) \le \frac{\varepsilon}{\varepsilon'^2} + \delta. \tag{2}$$

By definition of H_j, X_j can be rewritten as

$$X_j = 2G_{j+1} + \sum_{i \in HH_j}(1 - 2H_{j+1}(i))g(f_i).$$

Then, $|X_j - Y_j| \le 2|G_{j+1} - Y_{j+1}| + \sum_{i \in HH_j}|\hat{g}(f_i) - g(f_i)|$. To simplify further derivations, denote $E_j^1 = |X_j - G_j|$, $E_j^2 = |G_j - Y_j|$ and $E_j^3 = \sum_{i \in HH_j}|\hat{g}(f_i) - g(f_i)|$. $E_j^2 = |G_j - Y_j| \le |G_j - X_j| + |X_j - Y_j| \le E_j^1 + 2E_{j+1}^2 + E_j^3$. Therefore, the error will propagate to layer 0 as: $E_0^2 \le E_0^1 + 2E_1^2 + E_0^3 \le \ldots \le 2^\phi E_\phi^2 + \sum\limits_{j=0}^{\phi} 2^j E_j^1 + \sum\limits_{j=0}^{\phi} 2^j E_j^3$. Denote event $2^\phi E_\phi^2 \ge \varepsilon'' G(t_1,t)$ as A, event $\sum_{j=0}^{\phi} 2^j E_j^1 \ge \varepsilon'' G(t_1,t)$ as B, and event $\sum_{j=0}^{\phi} 2^j E_j^3 \ge \varepsilon'' G(t_1,t)$ as C.

Apply formula 2 for all j, then $\Pr(B)$ is upper bounded by:

$$\Pr\left(\varepsilon' \sum_{j=0}^{\phi} 2^j G_j(t_1,t) \ge \varepsilon'' G(t_1,t)\right) + (\phi+1)\left(\frac{\varepsilon}{\varepsilon'^2} + \delta\right).$$

Note that $E\left(\sum_{j=0}^{\phi} 2^j G_j(t_1,t)\right) = (\phi+1)G_j(t_1,t)$, therefore by Markov: $\Pr(B) \le (\phi+1)\frac{\varepsilon'}{\varepsilon''} + (\phi+1)\left(\frac{\varepsilon}{\varepsilon'^2} + \delta\right)$.

Recall that HH_j fails with probability at most δ, there are at most $1/\varepsilon'$ items such that $g(f_i) \ge \varepsilon G_j(t_1,t)$, and if HH_j succeeds then $\hat{g}(f_i) = (1 \pm \varepsilon)g(f_i)$ for all i such that $g(f_i) \ge \varepsilon G_j(t_1,t)$. Therefore, $\sum_{i \in HH_j}|\hat{g}(f_i) - g(f_i)| < \varepsilon G_j(t_1,t)$, and we can bound $\Pr(C)$ from above with:

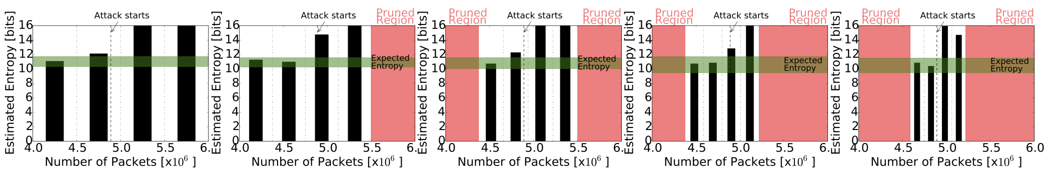
$$\Pr\left(\varepsilon \sum_{j=0}^{\phi} 2^j G_j(t_1,t) \ge \varepsilon'' G(t_1,t)\right) + (\phi+1)\delta.$$

Finally, applying Markov: $\Pr(C) \le (\phi+1)\frac{\varepsilon}{\varepsilon''} + (\phi+1)\delta$.

To bound $\Pr(A)$ recall that $Y_\phi = G_\phi(t_1,t_2) \pm \varepsilon G_\phi(t_1,t)$ with probability at least $1-\delta$ and $E(2^\phi G_\phi(t_1,t)) = G(t_1,t)$. Therefore, $P(A) \le \frac{\varepsilon}{\varepsilon''} + \delta$, and putting all together: $P(A \cup B \cup C) \le (\phi+2)\frac{\varepsilon}{\varepsilon''} + (\phi+1)\frac{\varepsilon'}{\varepsilon''} + (\phi+1)(\frac{\varepsilon}{\varepsilon'^2} + \delta)$. Choosing $\varepsilon \le \frac{0.1\varepsilon''^2}{(\phi+1)^3}$ and $\varepsilon' \le \frac{0.1\varepsilon''}{\phi+1}$ we get the statement of the theorem $P(|Y_0 - G(t_1,t_2)| \ge \varepsilon'' G(t_1,t)) \le 0.3$. □

3.6 Extending to a wider class of queries

Recall that the interval queries (t_1,t_2) introduced earlier were not measured in terms of time, but in the number of packets passed through. However, in practice, it is often more useful when one can query some statistic in time-based intervals, such as a one-hour interval that occurred half a day ago or from 5PM to 6PM yesterday. All presented algorithms are easily extendable to answer time-based interval queries. One only needs to create a timestamp for each bucket of SH or EH framework, and use that timestamp when searching for corresponding buckets approximating the



(a) Step 0 (b) Step 1 (c) Step 2 (d) Step 3 (e) Step 4

Fig. 6. Illustration of the attack pinpointing algorithm's first steps on the Chicago 16 trace with 63MB. In steps 1, 3 and 4, the first unexpected entropy appeared in the last two sub-intervals, allowing the algorithm to prune the first sub-interval. In steps 0 and 2, the algorithm encounters abnormally high entropy already in the second sub-interval and therefore prunes the latest sub-interval. For reference, we annotate the actual starting time of the attack (which is not known to the algorithm).

interval. Note that all presented algorithms support weighted packets, i.e., when each packet i arrives with its weight w_i, which corresponds to the update $f_i = f_i + w_i$.

3.7 Attack localization algorithm

Next, we describe an attack localization algorithm that assumes the capability to estimate entropy in the IQ model.

Our algorithm periodically estimates the entropy within a sliding window, e.g., of once per 1M packets, and maintains the average and standard deviation of the entropy over previous windows. If the current entropy measurement is not within $\alpha > 1$ standard deviations from the past entropy then we raise an *alarm* and begin with the attack localization process. α is a sensitivity parameter; a low α implies that we detect attacks quickly but raise some false alarms, while a larger value means that we are slower to detect attacks but raise fewer false alarms. In our evaluation, we select $\alpha = 5$, which provides very few false alarms and is still sensitive enough to quickly detect attacks.

We choose to base our system on the entropy signal which is known to detect a variety of attacks [31, 36, 48, 56, 58, 59].

Our attack localization process begins within a suspected interval of the two last windows (e.g., 4M-6M). We use these two window intervals as a *Baseline*, as any technique to measure entropy over sliding window (e.g., [6, 38]) can perform this kind of localization. Setting the Baseline interval as the last two windows makes sense as the attack did not necessarily start within the window that raised the alarm. It is possible that the attack began within the previous window but did not build enough impact to trigger the alarm.

Next, we attempt to recursively narrow the suspected interval. We partition the current interval (e.g., 4M-6M) into four equal parts (e.g., [4M,4.5M], [4.5M,5M], [5M,5.5M], and [5.5M,6M]) and estimate the entropy on each of them. Again, we seek intervals with entropy that is not within five standard deviations of the usual entropy. The delicate point is to understand that entropy is not linear, and we cannot compare the entropy of a short interval to that of a long interval.

Instead, we use the entropy of equal sized subintervals that happened before the Baseline intervals to define the 'normal' entropy. That is, we calculate the mean and standard deviation of such intervals. We call these *Safe* subintervals as they come before the attack. In this example, we estimate the entropy of [3.5M,4M], [3M,3.5M], etc. We favor the most recent safe intervals for practical concerns (as the normal behavior may gradually change), as well as for algorithmic consideration as the entropy estimation error for these intervals is smaller (see Figure 12).

We set the new suspected interval as a concatenation of all abnormal intervals, and recursively repeat the above process until we can no longer shorten the suspected interval. The operation of the

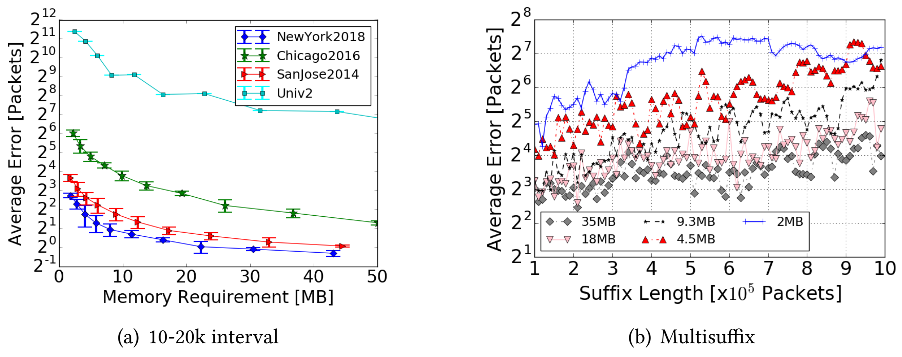


Fig. 7. Average frequency estimation error for flows (a) in 10-20k intervals, and (b) for various suffix lengths on the NY2018 dataset.

algorithm is illustrated in Figure 6. Specifically, if one of the first two subintervals had unexpectedly high entropy (as in Step 0 and 2), we prune the latest subinterval. Otherwise (as in steps 1,3, and 4), we can safely prune the first subinterval as we expect the entropy to rise after at least one complete attack subinterval. The algorithm proceeds until either no subinterval has too high entropy or until we get an estimated entropy of 0 from our algorithm (this happens if the queried interval is too small and consists of less than half a block, see Algorithm 4 and its description in Section 3.5).

Note that since network traffic has internal noise, and our algorithms have estimation error, we are unable to identify attacks that do not significantly change the entropy. This is usually acceptable for network operators as typically only significant attacks (e.g., that triple the bandwidth) are causing noticeable harms to the network services. Networks often have enough resources to cope with small attacks (e.g., only 10% increase in traffic) and these go unnoticed. Our limitations are inherently similar to other entropy-based detection algorithms [6, 38]. The novelty of our approach lies within the attack localization process that allows to accurately time the beginning of the attack.

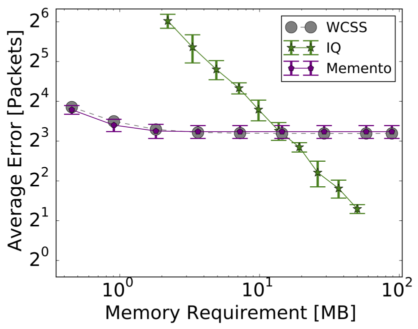
4 EVALUATION

Next, we evaluate Algorithm 3 for various network measurement tasks. We have implemented a prototype in C and evaluated the accuracy vs. memory using four Internet Traces: “Equinix-Sanjose” in 2014 (SanJose2014) [3], “Equinix-Chicago” in 2016 (Chicago2016) [1], “Equinix-NewYork” in 2018 (NewYork2018) [2]; and a data center trace from the University of Wisconsin (Univ2) [13].

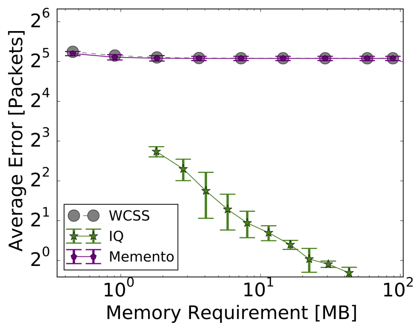
All experiments start by evaluating our algorithms on the first 10M packets from the specific trace. We evaluate multiple measurement metrics on two experiments. In the first experiment, we select a packet once every 30k packets and estimate the frequency of the corresponding flow on an interval between 10k-20k packets ago (suffix length of 20k). In the second, we estimate frequencies on varying suffixes and show how the suffix length affects the empirical error. To do so, we select a packet once per 200k packets and estimate the frequency of the corresponding flow for every possible suffix length from 100K to the window size of 1M. The depicted figures for the second experiment are for the NewYork2018 dataset.

4.1 Frequency estimation

We start with the frequency estimation problem. The results in Figure 7(a) show the trade-off between memory consumption and the empirical error for different network traces. As expected, having more memory increases the accuracy of frequency estimation. The difference between the

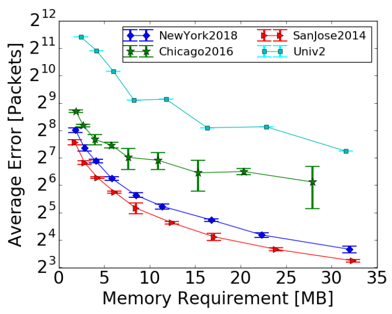


(a) Chicago2016

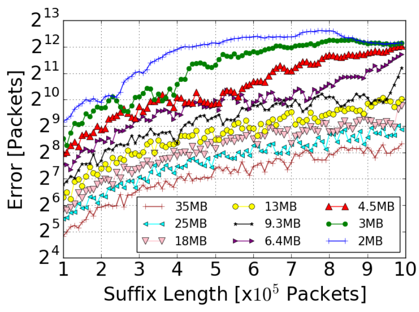


(b) NewYork2018

Fig. 8. Average frequency estimation error on the NY2018 dataset for flows in 10-20k intervals compared to sliding window algorithms.

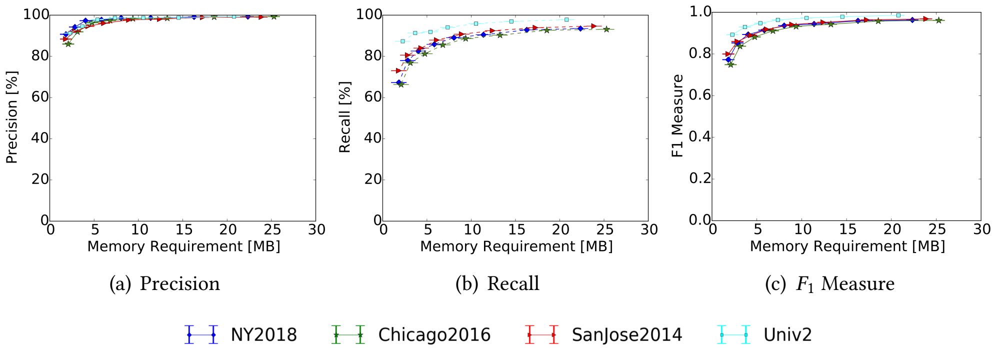


(a) 10-20k interval



(b) Multi suffix

Fig. 9. Average L_2 norm estimation error for flows (a) in 10-20k intervals, and (b) for various suffix lengths on the NY2018 dataset.



(a) Precision (b) Recall (c) F_1 Measure

Fig. 10. Quality of HH solution for 10k-20k interval (first experiment).

traces is mainly attributed to the workload characteristics and namely how the L_2 norm changes during each trace.

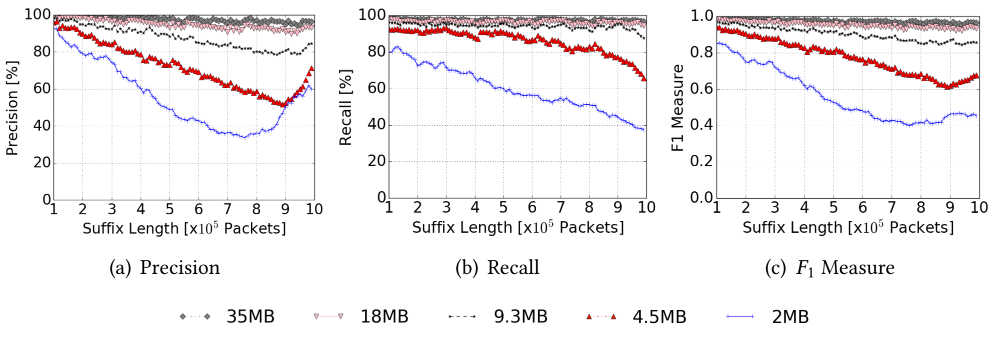


Fig. 11. Quality of HH solution for varying suffix lengths (second experiment).

Figure 7(b) shows the results for the second experiment on the NY2018 trace. Notice that (i) longer suffixes indeed have larger estimation error than shorter ones. This is expected as our analysis indicates that the error is proportional to the suffix length. (ii) Notice that more memory increases accuracy for every suffix length. This is also expected as the error is proportional to the accuracy parameter ε which decreases with the memory. (iii) The average error is small for moderate memory consumption, e.g., given 18MB, the average error is less than 64 packets for all suffix lengths.

Next, we compare Algorithm 3 (denoted IQ) to the deterministic WCSS [10] and the randomized Memento [8], which are state of the art sliding window algorithms. Since sliding window algorithms estimate the frequency over the entire window, we satisfy interval queries by normalizing the frequency in the window by the interval size. For example, we estimate the frequency in an interval whose size is 1% of the entire window as 0.01 times the estimated frequency over the whole window. The results, shown in Figure 8, indicate that the accuracy of WCSS and Memento is very similar for the entire evaluated memory range and that they do not significantly improve as we increase the memory. The reason is that both are able to provide highly accurate estimations for the frequency in the window and the depicted error is due to the difference between the distribution in the queried interval and that of the entire window. In contrast, our algorithm improves when given more space and is able to provide accurate interval estimates when given enough memory. We conclude that using a sliding window algorithm for the task is inadequate unless space is extremely tight and our solution cannot run.

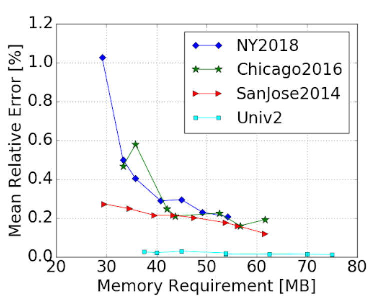
4.2 L_2 norm estimation

We repeat the above experiments for L_2 norm estimation. Figure 9(a) shows results for the first experiment. Notice that we get the same trend as before, more memory leads to better accuracy in estimating the L_2 norm. Figure 9(b) shows results for the second experiment. As can be seen, (i) we get better L_2 estimations for small suffixes, (ii) additional memory increases the accuracy.

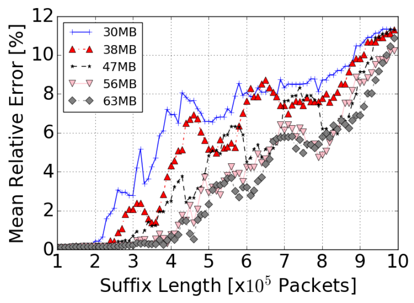
4.3 Heavy hitters estimation

We now evaluate L_2 heavy hitters, with threshold of: $\theta = 0.5\%$. That is, we define a heavy hitter as a flow whose frequency is at most 0.5% of the L_2 norm for the referenced interval.

We evaluate three metrics for heavy hitters estimation. (i) Precision: measure how many of the reported flows are indeed heavy hitters, (ii) Recall: measure how many of the real heavy hitters are reported, and (iii) the F_1 norm that factors both Precision and Recall into a single measure. The perfect solution has $F_1 = 1$, and the closer the value is to 1, the more accurate the solution.



(a) 10-20k interval



(b) Multi suffix

Fig. 12. Average entropy relative error for (a) 10-20k intervals, and (b) various suffix lengths on the NY2018 dataset.

Figure 10(a) shows precision and Figure 10(b) presents recall for various workloads for the first experiment. Notice that (i) increasing memory increases both precision and recall; (ii) we get to around 90% recall and 95+% precision within reasonable memory. Figure 10(c) shows the F_1 values in the same experiment. As can be observed, the value converges close to 1 as we increase the memory.

Figure 11 shows result for the second experiment. Figure 11(a) shows precision, Figure 11(b) shows recall and Figure 11(c) shows the F_1 metric. As can be observed, longer suffixes yield less accurate results. For precision, there is a slight anomaly for 2 and 4.5 MB sized algorithms as the precision improves for longer suffixes at the end of the range. While precision improves, the recall drops which implies that we detect fewer heavy hitters but the list contains a larger portion of heavy hitters. Despite this anomaly, we clearly see the trend that increasing the memory improves the accuracy in all metrics.

4.4 Entropy estimation

Finally, we evaluate the error in Entropy estimation. We used Algorithm 4, and extended Univ-Mon [51] to provide Entropy estimations. Figure 12 shows the results. As illustrated, we estimate the entropy for this interval very accurately throughout the range of memory. As expected, more memory means a better estimation.

4.5 Case Study: Attack Localization

In this section, we showcase our attack localization algorithm, which is made possible by the new infrastructure we developed for IQ queries with the L_2 norm. To do so, we inject a DDoS attack within a random offset of a real Internet trace. From that offset, 70% of the traffic belongs to the DDoS attack and the rest is the original trace. We then use our attack localization algorithm to identify the beginning of the attack.

DDoS attack generation: We simulate an attack containing numerous attackers from 50 randomly picked subnets and numerous victims that share a single 16-bit subnet. Our attack is similar to the one considered in [8]. The high number of attackers and the low per-attacker traffic makes attack localization impossible using existing IQ techniques.

Localization results: Figure 13 shows the results for our attack localization technique. The figure shows the true entropy and distinct count of the stream along with the best localization achieved by the baseline, as well as by our attack localization algorithm when the underlying IQ algorithm is given different amount of memory. Our results show that (i) the baseline solution

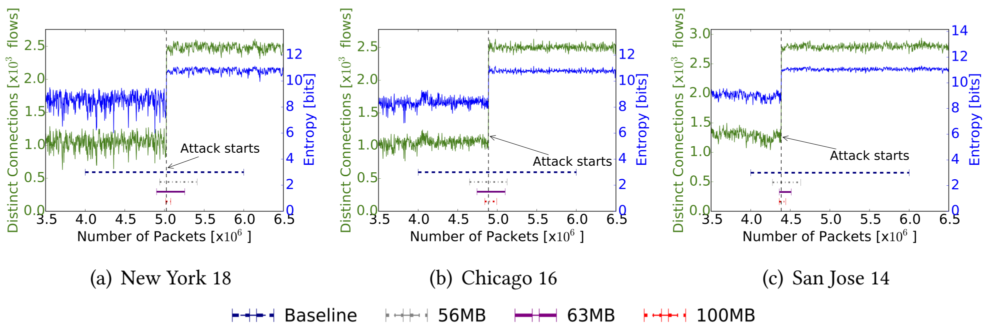


Fig. 13. Pinpointing the beginning of a DDoS attack with a sliding window algorithm (Baseline), as well as our own IQ model algorithm, varying memory consumption. Depicted are the exact entropy and distinct values; these are not known to the algorithm which approximates them.

Table 3. In-memory processing speed.

Performance	1 Core	2 Cores	4 Cores	8 Cores
Packet Rate	1.85Mpps	3.58Mpps	7.23Mpps	14.37Mpps
Throughput	13.7Gbps	26.51Gbps	53.54Gbps	106.42Gbps

using sliding windows is inaccurate for the task; That is, while existing methods may identify the attack they cannot localize it. (ii) Our attack localization successfully localizes the beginning of the attack; and (iii) there is a trade-off between the localization's accuracy and the allocated memory. More memory allows for better localization.

The results also confirm the effectiveness of our configuration, as the same setup works for all three traces. Our configuration also successfully detects stronger attacks.

4.6 Deployability

Finally, we evaluate the deployability of Algorithm 3. We measure the throughput of our algorithm when varying the number of threads. Table 3 shows the obtained results for the following configuration: Single-thread/multiple-thread with OpenMP in a commodity server with Intel Xeon E5-2640 v4 CPU. As shown, we attain 1.85 Million packets per second (Mpps) with a single thread using the smallest error we tested (ϵ=0.044) and can scale the throughput up to 14.37Mpps with eight threads. For reference, the average TCP packet size in the recent NewYork2018 trace is 994 Bytes, which implies that we must handle at least 1.25Mpps to cope with realistic traffic on a 10Gbps link. Clearly, such speed is within the capability of our approach, and we leave further improvements on the processing speed as future work.

5 DISCUSSION

This paper studies the IQ model that allows calculating metrics on any interval within a recent window. While databases often provide similar capabilities, they also maintain a full subset of the data. Our novelty lies in providing such capabilities in a space-efficient manner which is considerably smaller than any database. Our work utilizes sketches, which are extensively used in network measurement and is a step towards realizing efficient interval queries in network devices.

Our work studies how to extend L_2 heavy hitter algorithms to the IQ model. We justify the focus on L_2 as it is used as a building block in monitoring a large variety of network measurement tasks such as frequency estimation, L_2 norm, the number of distinct flows, and the entropy of traffic distribution. We suggest three different techniques to do so, each improving on the space requirement of the previous one. We also extend the analysis of the previously suggested Exponential Histogram [30], which enables it to be used as a building block in the IQ model. Finally, we use our best algorithm to extend UnivMon capable of all the network tasks mentioned above [51] to the IQ model, based on our best L_2 heavy hitter algorithm. Our work is the first to provide these measurement tasks in the IQ model, and through an extensive evaluation on real Internet traces, we show that these capabilities are practical within the available memory range of network devices.

We anticipate that IQ capabilities may enable a new generation of network applications. An example of such a potential application is automatic packet capture from attacks and traffic anomalies within network devices. Ideally, such captures would increase the visibility of attacks and network anomalies and may facilitate a new generation of security applications. Our work also makes an important practical step toward realizing such a capability by introducing an algorithm to localize the beginning of a traffic anomaly. We evaluate our localization algorithm under realistic conditions and demonstrate the ability to pinpoint the start of simulated attacks.

In the future, we plan to run the data plane of IQ algorithms on top of virtual switches such as OVS [61], VPP [34], or BESS [40], and implement the control plane within an SDN controller. Our evaluation shows that our approach has the potential to cope with the line speed of such measurement. Further, we believe that our speed can further be increased by replacing our CountSketch-based implementation within the faster AlwaysCorrect-NitroSketch [50] at the cost of a larger memory requirement. We envision that similar acceleration methods may enable IQ queries on 40G links.

ACKNOWLEDGMENTS

We thank our shepherd Ramakrishnan Durairajan and the anonymous reviewers for their valuable feedback. This work is supported in part by NSF grants 1813487, CNS-1700521, CNS-1565343, CAREER-1652257, ONR Award N00014-18-1-2364, Intel Labs University Research Office, Lifelong Learning Machines program from DARPA/MTO, Israeli Science Foundation grant #1505/16, the Zuckerman Foundation, the Technion Hiroshi Fujiwara Cyber Security Research Center, the Israel Cyber Directorate, the Cyber Security Research Center and the Lynne and William Frankel Center for Computing Science at Ben-Gurion University.

REFERENCES

[1] The CAIDA Anonymized Internet Trace, equinix-chicago 2016-06-21, Dir. A.

[2] The CAIDA Anonymized Internet Trace equinix-nyc 2018-03-15, Dir. A.

[3] The CAIDA Anonymized Internet Trace, equinix-sanjose 2014-03-20, Dir. B.

[4] Charu C Aggarwal. *Data Streams: Models and Algorithms*, volume 31. Springer Science & Business Media, 2007.

[5] Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comp. and sys. sciences*, 1999.

[6] Eran Assaf, Ran Ben-Basat, Gil Einziger, and Roy Friedman. Pay for a Sliding Bloom Filter and Get Counting, Distinct Elements, and Entropy for Free. In *IEEE INFOCOM*, 2018.

[7] Ziv Bar-Yossef, Thathachar S Jayram, Ravi Kumar, and D Sivakumar. An Information Statistics Approach to Data Stream and Communication Complexity. *Journal of Computer and System Sciences*, 2004.

[8] Ran Ben Basat, Gil Einziger, Isaac Keslassy, Ariel Orda, Shay Vargaftik, and Erez Waisbard. Memento: Making Sliding Windows Efficient for Heavy Hitters. In *ACM CoNEXT*, 2018.

[9] Ran Ben Basat, Roy Friedman, and Rana Shahout. Heavy Hitters over Interval Queries. In *PVLDB*, 2019. Also available on arXiv:1804.10740.

[10] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner. Heavy Hitters in Streams and Sliding Windows. In *IEEE INFOCOM*, 2016.

[11] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Randomized Admission Policy for Efficient Top-k and Frequency Estimation. In *IEEE INFOCOM*, 2017.

[12] Ran Ben-Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Constant Time Updates in Hierarchical Heavy Hitters. *ACM SIGCOMM*, 2017.

[13] Theophilus Benson, Aditya Akella, and David A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *ACM IMC*, 2010.

[14] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *ACM CoNEXT*, 2011.

[15] Arnab Bhattacharyya, Palash Dey, and David P Woodruff. An Optimal Algorithm for L_1-Heavy Hitters in Insertion Streams and Related Problems. In *ACM PODS*, 2016.

[16] V. Braverman. Sliding window algorithms. *Encyc. of Algorithms, 2004.*

[17] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. BPTree: an L_2 Heavy Hitters Algorithm using Constant Memory. *arXiv:1603.00759*, 2016.

[18] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, and David P Woodruff. Beating CountSketch for Heavy Hitters in Insertion Streams. In *ACM STOC*, 2016.

[19] Vladimir Braverman, Stephen R Chestnut, David P Woodruff, and Lin F Yang. Streaming Space Complexity of Nearly All Functions of One Variable on Frequency Vectors. In *ACM PODS*, 2016.

[20] Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to Catch L_2-heavy-hitters on Sliding Windows. *Theoretical Computer Science*, 2014.

[21] Vladimir Braverman, Elena Grigorescu, Harry Lang, David P Woodruff, and Samson Zhou. Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows. *arXiv preprint arXiv:1805.00212*, 2018.

[22] Vladimir Braverman and Rafail Ostrovsky. Smooth Histograms for Sliding Windows. In *IEEE FOCS*, 2007.

[23] Vladimir Braverman and Rafail Ostrovsky. Generalizing the Layering Method of Indyk and Woodruff: Recursive Sketches for Frequency-Based Vectors on Streams. In *APPROX/RANDOM*, 2013.

[24] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal Lower Bounds on the Multi-party Communication Complexity of Set Disjointness. In *IEEE CCC*, 2003.

[25] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding Frequent Items in Data Streams. In *ICALP*, 2002.

[26] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, and Ori Rottenstreich. Catching the Microburst Culprits with Snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*, SelfDN 2018, 2018.

[27] Edith Cohen. All-Distances Sketches, Revisited: HIP Estimators for Massive Graphs Analysis. *IEEE Trans. Knowl. Data Eng.*, 2015.

[28] Graham Cormode and Marios Hadjieleftheriou. Methods for Finding Frequent Items in Data Streams. *J. VLDB*, 2010.

[29] Graham Cormode and Shan Muthukrishnan. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms*, 2005.

[30] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM J. Comp., 2002.*

[31] Jisa David and Ciza Thomas. DDoS Attack Detection using Fast Entropy Approach on Flow-based Network Traffic. *Procedia Computer Science*, 2015.

[32] G. Einziger, B. Fellman, and Y. Kassner. Independent Counter Estimation Buckets. In *IEEE INFOCOM*, 2015.

[33] Cristian Estan, Stefan Savage, and George Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *ACM SIGCOMM*, 2003.

[34] FD.io. Vector packet processing, 2018.

[35] Shir Landau Feibish, Yehuda Afek, Anat Bremler-Barr, Edith Cohen, and Michal Shagam. Mitigating DNS Random Subdomain DDoS Attacks by Distinct Heavy Hitters Sketches. In *ACM/IEEE HotWeb 2017*.

[36] Laura Feinstein, Dan Schnackenberg, Ravindra Balupari, and Darrell Kindred. Statistical Approaches to DDoS Attack Detection and Response. In *Proceedings DARPA information survivability conference and exposition*, 2003.

[37] Éric Fusy and Frédéric Giroire. Estimating the Number of Active Flows in a Data Stream over a Sliding Window. In *ANALCO*, 2007.

[38] Moshe Gabel, Daniel Keren, and Assaf Schuster. Anarchists, Unite: Practical Entropy Approximation for Distributed Streams. In *ACM KDD*, 2017.

[39] Pedro Garcia-Teodoro, Jesus E. Diaz-Verdejo, Gabriel Macia-Fernandez, and E. Vazquez. Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers and Security*, 2009.

[40] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. SoftNIC: A Software NIC to Augment Hardware. Technical report, 2015.

[41] Hazar Harmouch and Felix Naumann. Cardinality Estimation: An Experimental Survey. *J. VLDB*, 2017.

[42] Stefan Heule, Marc Nunkesser, and Alexander Hall. HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm. In *ACM EDBT*, 2013.

[43] Piotr Indyk and David Woodruff. Optimal Approximations of the Frequency Moments of Data Streams. In *ACM STOC*, 2005.

[44] Nikita Ivkin, Edo Liberty, Kevin Lang, Zohar Karnin, and Vladimir Braverman. Streaming quantiles algorithms with small space and update time. *arXiv preprint arXiv:1907.00236*, 2019.

[45] Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. Qpipe: Quantiles sketch fully in the data plane. In *ACM CoNEXT*, 2019.

[46] Atul Kant Kaushik, Emmanuel S. Pilli, and R. C. Joshi. "Network Forensic Analysis by Correlation of Attacks with Network Attributes". In *Information and Communication Technologies*, 2010.

[47] Ilan Kremer, Noam Nisan, and Dana Ron. On Randomized One-round Communication Complexity. *Computational Complexity*, 1999.

[48] Krishan Kumar, RC Joshi, and Kuldip Singh. A Distributed Approach using Entropy to Detect DDoS Attacks in ISP Domain. In *IEEE ICDCS*, 2007.

[49] Xuemin Lin, Hongjun Lu, Jian Xu, and Jeffrey Xu Yu. Continuously Maintaining Quantile Summaries of the Most Recent N Elements over a Data Stream. ICDE, 2004.

[50] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. NitroSketch: Robust and General Sketch-based Monitoring in Software Switches. In *ACM SIGCOMM*, 2019.

[51] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *ACM SIGCOMM*, 2016.

[52] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *ICDT*, 2005.

[53] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM SIGCOMM*, 2017.

[54] Jayadev Misra and David Gries. Finding Repeated Elements. *Science of computer programming*, 1982.

[55] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. DREAM: Dynamic Resource Allocation for Software-defined Measurement. In *ACM SIGCOMM*, 2014.

[56] Michael Müter and Naim Asaj. Entropy-based Anomaly Detection for In-vehicle Networks. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011.

[57] Shanmugavelayutham Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in TCS*, 2005.

[58] AS Navaz, V Sangeetha, and C Prabhadevi. Entropy based anomaly detection system to prevent DDoS attacks in cloud. *arXiv preprint arXiv:1308.6745*, 2013.

[59] George Nychis, Vyas Sekar, David G. Andersen, Hyong Kim, and Hui Zhang. An Empirical Evaluation of Entropy-based Traffic Anomaly Detection. In *ACM IMC*, 2008.

[60] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketching Distributed Sliding-window Data Streams. *The VLDB Journal*, 2015.

[61] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The Design and Implementation of Open vSwitch. In *USENIX NSDI*, 2015.

[62] Vyas Sekar, Nick G Duffield, Oliver Spatscheck, Jacobus E van der Merwe, and Hui Zhang. LADS: Large-scale Automated DDoS Detection System. In *USENIX ATC*, 2006.

[63] Haya Shulman and Michael Waidner. Towards Forensic Analysis of Attacks with DNSSEC. In *IEEE SPW*, 2014.

[64] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 795–810. ACM, 2015.

[65] Li Yang, Wu Hao, Pan Tian, Dai Huichen, Lu Jianyuan, and Liu Bin. CASE: Cache-assisted Stretchable Estimator for High Speed Per-flow Measurement. In *IEEE INFOCOM*, 2016.

[66] Sen Yang, Bill Lin, and Jun Xu. Safe Randomized Load-Balanced Switching By Diffusing Extra Loads. *ACM Meas. Anal. Comput. Syst., 2007.*

[67] Ke Yi and Qin Zhang. Optimal Tracking of Distributed Heavy Hitters and Quantiles. *Algorithmica*, 2013.

A EXTERNAL THEOREMS REFERRED WITHIN MANUSCRIPT

Theorem A.1 ([22], Theorem 3). *Let f be a (α, β)-smooth function. If there exists an algorithm Λ that maintains an (ε, δ)-approximation of f on D, using space $g(\varepsilon, \delta)$ and performing $h(\varepsilon, \delta)$ operations per stream element, then there exists an algorithm Λ' that maintains a $(\alpha + \varepsilon)$-approximation of f on sliding windows and uses $O\left(\frac{1}{\beta}\left(g\left(\varepsilon, \frac{\delta\beta}{\log n}\right) + \log n\right)\log n\right)$ bits and $O\left(\frac{1}{\beta}h\left(\varepsilon, \frac{\delta\beta}{\log n}\right)\log n\right)$ operations per element.*

Theorem A.2 ([30], Theorem 7). *A function f with properties:*

(1) $f(B_i) \geq 0$.
(2) $f(B_i) \leq poly(|B_i|)$.
(3) *$f(B_1 + B_2) \geq f(B_1) + f(B_2)$, where $B_1 + B_2$ denotes the concatenation of adjacent buckets B_1 and B_2.*
(4) *$f(B_1 + B_2) \leq C_f(f(B_1) + f(B_2))$, where $Cf \geq 1$ is a constant.*
(5) *The function $f(B)$ admits a "sketch" which requires $g_f(|B|)$ space and is composable; i.e., the sketch for $f(B_1 + B_2)$ can be composed efficiently from the sketches for $f(B_1)$ and $f(B_2)$.*

can be estimated over sliding windows with relative error

$$0 \leq E_r \leq (1 + \varepsilon)^2 \frac{C_f^2}{k} + C_f - 1 + \varepsilon$$

using $O(k \log n(\log n + g_f(n)))$ bits of memory, where ε is the bound on relative error of the sketches.

B COMPARISON WITH "PERSISTENT DATA SKETCHING" [64]

Model suggested in [64] differs from the results presented in current manuscript in several dimensions:

(1) Our model assumes the limit to the size of the window which can be queried $(t_1, t_2) \subset (t-n, t)$ and all our sketches require space sublinear in the maximum size of the window n (see Table 2). [64], in contrary, have a space requirement which is linear in the total length of the stream (see paragraph "Space and query time" of Section 4.2 in [64]), therefore will not work with infinite streams. The attack localization use case, presented in Section 4.5, depicts the advantage of of the current paper.
(2) We prove the space bounds for all our sketches in the worst case (adversarial) scenario of the input stream. In contrary, [64] assumes the stream to be random, and provides space guarantees only in expectation. Clearly, such an assumption is a drawback for security applications.
(3) [64] introduces the data structures for L_1-heavy hitters and L_2-norm estimation. Our main result is L_2 heavy hitters and theoretical justification of our extension of UnivMon [51] framework to interval queries. In turn, this allows approximating a variety of functions that can be computed on interval queries (e.g., entropy). The closest comparable result in our manuscript and [64] is approximation of L_2-norm on the interval (t_1, t_2): Lemma 3.4 above and Theorem 4.2 in [64].

Though a direct comparison can not be done as [64] does not have worst-case guarantees, we compare the guarantees assuming random streams. According to Lemma 3.4, and Theorem A.2, our algorithm can estimate $L_2(t_1, t_2)$ with error $\pm O(\frac{1}{\sqrt{k}})L_2(t_1, t)$ using space of $O(k(\log n + g(\varepsilon)))$, where $g(\varepsilon)$ is space needed by AMS sketch[2], therefore approximation $\pm\varepsilon L_2(t_1, t)$ can be achieved with space $O(\frac{1}{\varepsilon^2}(\log n + \frac{1}{\varepsilon^2}))$. To achieve the same approximation we adopt results from Theorem 4.2 in [64]

[2]for simplicity we avoid terms $\log \frac{1}{\delta}$, $\log \frac{1}{\varepsilon}$, $\log \log n$ and count all complexities in words, rather than bits.

which provides approximation error of $\pm\varepsilon'(L_2^2(t_1,t_2)+\frac{\Delta^2}{\varepsilon'^2})$. We set $\varepsilon'=\varepsilon^2$ (this seems to be a natural choice as our approximations are $\pm\varepsilon L_2$ and [64] uses $\pm\varepsilon' L_2^2$).

$$\varepsilon'\left(L_2^2(t_1,t_2)+\frac{\Delta^2}{\varepsilon'^2}\right)=\varepsilon^2 L_2^2(t_1,t)\Rightarrow L_2^2(t_1,t_2)+\frac{\Delta^2}{\varepsilon^4}=L_2^2(t_1,t)$$

$$\Rightarrow\frac{\Delta^2}{\varepsilon^4}\le L_2^2(t_1,t)\Rightarrow\Delta\le\varepsilon^2 L_2(t_1,t)$$

The Space complexity of the algorithm presented in [64] is therefore

$$O\left(\frac{m}{\Delta}+\frac{1}{\varepsilon'^2}\right)=O\left(\frac{m}{\varepsilon^2 L_2(t_1,t)}+\frac{1}{\varepsilon^4}\right).$$

We assume, that $n=m$ (i.e., we adjust our solution to the model presented in [64]). Note, that $L_2(t_1,t)\le L_1(t_1,t)\le L_1(0,t)=m$ with the equality holding when only one unique item is present in the stream (i.e., unrealistically extreme case). In that case, on the the very outdated queries (i.e. when t_1 and t_2 both very close to beginning of the stream) we have both our algorithm and algorithm from [64] working in space $O(\frac{1}{\varepsilon^4})$; however, if we interested in queries more recent, for instance $L_2(t_1,t)=\sqrt{m}$, then our algorithm requires memory logarithmic in m: $O(\frac{1}{\varepsilon^2}(\log m+\frac{1}{\varepsilon^2}))$ while result from [64] requires space polynomial in m: $O(\frac{\sqrt{m}}{\varepsilon^2}+\frac{1}{\varepsilon^4})$.

Received August 2019; revised September 2019; accepted October 2019