Together is Better: Heavy Hitters Quantile Estimation

Stream monitoring is fundamental in many data stream applications, such as financial data trackers, security, anomaly detection, and load balancing. In that respect, quantiles are of particular interest, as they often capture the user's utility. For example, if a video connection has high tail (e.g., 99'th percentile) latency, the perceived quality will suffer, even if the average and median latencies are low.

In this work, we consider the problem of approximating the *per-item* quantiles. Elements in our stream are (ID, value) tuples, and we wish to track the quantiles for each ID. Existing quantile sketches are designed for a plain number stream (i.e., containing just a value). While one could allocate a separate sketch instance for each ID, this may require an infeasible amount of memory. Instead, we consider tracking the quantiles for the heavy hitters (most frequent items), which are often considered particularly important, without knowing them beforehand.

We first present a couple of simple and effective algorithms that serve as baselines, a sampling approach and a sketching approach. Then, we present SQUAD, an algorithm that combines sampling and sketching while improving the asymptotic space complexity. Intuitively, SQUAD uses a background sampling process to capture the behaviour of the quantiles of an item before it is allocated with a sketch, thereby allowing us to use fewer samples and sketches. The algorithms are rigorously analyzed, and we demonstrate SQUAD's superiority using extensive simulations on real-world traces.

Key Words: Data Stream Algorithms, Quantiles, Sketches

Rana Shahout, Roy Friedman, and Ran Ben Basat. 2024. Together is Better: Heavy Hitters Quantile Estimation.

1 INTRODUCTION

Quantiles enable to extract useful insights from non-uniformly distributed data. Popular examples include the wealth required to be recognized among the 1% richest persons in the world or in a given country, or the salary level required to be in the top 10% earners. Service Level Agreements (SLA) is another domain whose parameters are often expressed in terms of quantiles [37], most notably w.r.t. query execution time and overall service latency [15, 22, 42, 49]. Latency quantiles is also a vital metric in assessing a network's health and in debugging various networking middle-boxes and smart data-planes [46].

When handling large and fast incoming data streams, as common in network monitoring and large data-center and cloud applications, answering precise quantile queries is often prohibitively expensive. To that end, sketching [32] or sampling [25] are widely used to provide approximate answers with some bounded error guarantee.

Interestingly, in many real-world streaming applications, such as network monitoring, data-centers, and clouds, the streams are composed of elements whose header includes some identifier. In networking, the identifier is typically the source IP+port, destination IP+port and protocol 5-tuple, while in other applications, this can be a URI, a user's identifier, product identifier, etc. It is common to refer to elements with the same identifier as an *item* or a *flow* within the stream.

We argue that tracking the overall data stream's quantiles is not enough, and in many cases keeping track of the per item quantiles is highly beneficial. For example, it is possible that some global SLA parameter's quantile is within a given SLA bound, yet for some items (flows) it is not. This means that the experience of the respective users is unacceptable, despite the overall parameter

Authors' addresses: Rana Shahout, Technion, Israel, ranas@cs.technion.ac.il; Roy Friedman, Technion, Israel, roy@cs.technion.ac.il; Ran Ben Basat, University College London, UK, r.benbasat@cs.ucl.ac.uk.

distribution "looking OK". As another example, the reason the overall tail latency of a system is too high may be mainly due to a few items. Being able to estimate the quantile latency of individual items can help pinpoint the sources of the problem. Further, "typical" network routing paths for different items (flows) are often very different, depending on the source and destination and their topological locations within the Internet. Thus, considering a global path-length quantile is much less informative for network monitoring and debugging than the per item path-length quantile.

Existing works, however, cannot efficiently track fine-grained per-item quantiles for all traffic. Network monitoring techniques such as NetFlow [24] and In-band telemetry (INT) [52] suffer from significant bandwidth costs when performing full-packet monitoring. Accordingly, researchers proposed probabilistic approaches that reduce the overheads of collecting per-item quantiles [14]; however, these still incur high collection costs [34] and require modifying data packets, which is undesired. Sampling from the entire stream in order to obtain per-item quantiles also consumes a significant amount of memory, as we show later.

Several quantile *sketching* algorithms have been developed [4, 25, 27, 32, 38, 39, 51], with useful extensions explored such as calculating quantiles across sliding windows [7], over distributed data [4, 28, 30, 51], continuous monitoring of quantiles [16, 55], quantile computations using GPUs [26], and biased quantiles [20]. Such sketches return an approximation of a q-quantile's value up to an ε error guarantee, but they track the quantiles of an entire stream. It may be tempting to utilize any of these sketching techniques to address the per-item quantile estimation problem, by simply maintaining one sketch per item. However, in large scale applications, the number of items can be extremely high, thereby rendering this approach prohibitively costly.

Hence, we may instead focus on tracking the q-quantile's value for a subset of significant items in terms of their frequency in the stream. The motivation for targeting only significant items is that a quantile's value is often misrepresented when there are only a few elements associated with a given item. For example, when considering tail latency for an item that only appears in one or two transactions, it is enough that a single transaction suffers from a long delay for the tail-latency of that item to be very large. Such one-time events can be caused by, e.g., caching initialization, storage warm-up, route discovery overheads, and "bad luck" in terms of temporal overloads on intermediate components and devices. Therefore, identifying q-quantile's values for significant items is more important and computationally feasible.

A popular way to identify significant items is known as *heavy-hitters*, i.e., all items whose associated elements consume more than a threshold θ of the overall stream [44]. Since a heavy-hitter item accounts for a significant fraction of the overall system load, it is important to ensure good quality of service for it, and there is vast knowledge on identifying heavy-hitters in streams, e.g., [17, 33, 43].

In this work, we focus on reporting the q-quantile's value of heavy hitter items. Our goal is to figure out how to find all θ-heavy-hitters' quantiles with a maximum accuracy error of ε, for given parameters θ and ε. Let us note that efficiently tracking the quantiles of heavy-hitters is non-trivial, since we do not know ahead of time which items are heavy-hitters. Hence, we must simultaneously both detect the heavy-hitters and estimate their quantiles while keeping the memory consumption low and the processing speed high.

In contrast to heavy hitters, for small items (e.g., that appear 5 times) the notion of tail quantiles is not defined. In some cases, we may want to capture all flows that arrived enough (e.g., 100) times, independently of the stream length. To that end, we present an algorithmic variant that captures this definition as well.

Contributions

We start by introducing a formal definition of the heavy hitters (HH) quantiles problem, named $(\theta, \varepsilon, \delta)$-HH-quantiles, which tracks the quantiles of heavy hitters.

For solving the problem, we first examine what it would mean to solve $(\theta, \varepsilon, \delta)$-HH-quantiles with a pure *sampling* based solution. Then we formally analyze this solution and show that it takes $O(\theta^{-1}\varepsilon^{-2}\log\delta^{-1})$ space. This algorithm is memory-inefficient, but it serves as a baseline for our final design.

Next, we inspect a *sketching* approach to $(\theta, \varepsilon, \delta)$-HH-quantiles, named SSGK, that is based on a nested application of Space Saving (SS) heavy hitters detection algorithm [43] with a quantile sketch. Space Saving [43] is an asymptotically optimal HH algorithm that requires $O(\varepsilon^{-1})$ space for solving the ε heavy hitters problem, and is considered highly accurate in practice [17].

In SSGK, we use the SS data structure and add a deterministic quantile sketch (e.g., GK-sketch [27]) to each entry. The GK-sketch requires $O(\varepsilon^{-1}\log(N\varepsilon))$ space, which is optimal for comparison-based quantile approximation. We analyze this solution and show that its memory complexity is $O\left(\theta^{-1}\varepsilon^{-2}\cdot\log(N\varepsilon^2\theta)\right)$. This is a similar space complexity to the sampling approach but with a deterministic error guarantee rather than a probabilistic one.

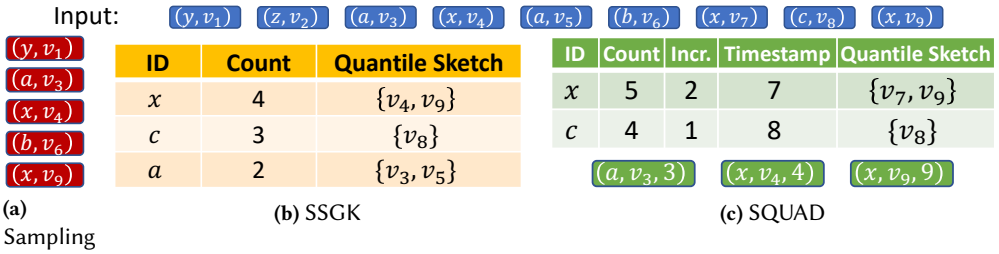
Our main contribution is the SQUAD algorithm, which uses a synergistic combination of the sketching and sampling approaches. Although there are a few works [36, 47] that combine sampling and sketching in some way, the worlds of sampling and sketching have stayed mostly distinct. Usually, this combination comes in the form of building the sketch over a sampled substream rather than the entire data stream in order to improve the time performance. In contrast, SQUAD uses the sketching approach of SSGK in tandem with sampling while slightly modifying both by adding a timestamp that allows merging. Intuitively, SQUAD employs an SS sketch with a quantile sketch per entry, together with background sampling to capture the behavior of a item's quantiles before assigning it a sketch. It adjusts the data structure size of each method and suggests how to merge the results of its components in order to benefit from each approach. At query time, the samples are incorporated into the sketch, yielding improved accuracy over both the baseline sketching and sampling approaches. As a result, SQUAD can use fewer samples and sketch entries for the same error guarantee. We formally analyze this solution and show that it takes $O\left(\theta^{-1}\varepsilon^{-1.5}\cdot\log\varepsilon^{-1}\right)$ space. We further present several enhancements that accelerate SQUAD and allow it to work at high processing rates. Figure 1 illustrates the algorithms presented in this paper, while the asymptotics are summarized in Table 1.

We perform a performance evaluation study of the above three solutions. To our knowledge, this is the first research to solve quantiles on a per-flow level. Thus, we compare our algorithms along with (*i*) GK-algorithm and Random [38], that serve as a best case reference point since it solves a more straightforward problem: quantiles over an entire stream (*ii*) the state-of-the-art Space Saving (SS) [43], which solves the heavy hitters problem which is a building block in SSGK and SQUAD. We evaluate our algorithms using large-scale datacenter FatTree topology using NS3 simulations [3]. The traffic is produced using the item size distribution in web search from Microsoft [5] and Hadoop from Facebook [48].

Conforming with the theory, the results show that given the same (ε, θ) error guarantees, SQUAD is the most space-efficient algorithm. SQUAD is faster than SSGK and while the sampling approach is faster, it also requires the most memory for solving the $(\theta, \varepsilon, \delta)$-HH-quantiles. Further, this advantage diminishes when we enable our runtime optimizations that enable SQUAD to process more than 100 million elements per second. Last, we generalize our method to support quantiles for traffic volume (weighted streams) as well as to items that appear at least Γ times (independently of the steam length) in the stream. All our code is open sourced [2].

Table 1. A comparison of the algorithms presented in this work in terms of their space complexity. The $\widetilde{O}$ notation hides polylogarithmic factors.

Algorithm	Space	Deterministic	Reference
Sampling	$\widetilde{O}(\theta^{-1}\varepsilon^{-2})$	✗	Section 4
SSGK	$\widetilde{O}(\theta^{-1}\varepsilon^{-2})$	✓	Section 5
SQUAD	$\widetilde{O}(\theta^{-1}\varepsilon^{-1.5})$	✗	Section 6



(a) Sampling **(b)** SSGK **(c)** SQUAD

Fig. 1. An illustration of our algorithms. Sampling simply selects a uniform random element subset from the input stream and uses the sample to infer frequencies and quantiles. SSGK uses a heavy hitters algorithm that has a quantile sketch embedded in each counter. SQUAD combines the two approaches to asymptotically reduce the space complexity. Crucially, SQUAD adds timestamps to both samples and sketches so that it can combine an item's sketch only with the samples that were not inserted into it. It also adds an increments counter, which is the increase in count since the item became monitored, thus reducing the frequency estimation error. SQUAD continues to sample elements of monitored items ($(x, v_9, 9)$ in this example) as the item may stop being monitored, i.e., the sampling process is independent of the sketching.

Paper roadmap: We survey related work in Section 2 and state the formal model and problem statement in Section 3. The sampling approach is described in Section 4 while SSGK is described and analyzed in Section 5. Our SQUAD algorithm is then described in Section 6, and its runtime optimizations in Section 7. The performance evaluation of our algorithms and their comparison to the GK, Random, and SS sketches is detailed in Section 8. Section 9 presents extensions of SQUAD and we conclude with a discussion in Section 10.

2 RELATED WORK

To the best of our knowledge, this is the first work that deals with the problem of per-element quantile estimation. Several earlier studies on streaming quantiles consider rank-queries, where the algorithm must associate an item y in the stream with a rank close to its true rank, defined as the number of stream elements smaller than or equal to y. In contrast, in our study, we focus on the quantile of individual elements in streams composed of identifiers and values, as described in Section 3. Below, we discuss prior work that has been done on solving streaming quantiles that guarantee an additive error ε with a constant failure probability δ.

Munro and Paterson included a p-pass algorithm for obtaining accurate quantiles in their classic study [45]. Although not explicitly studied, the method's initial run results in a streaming approach for producing approximate quantiles using $O(\varepsilon^{-1}\log^2(N\varepsilon))$ space. Manku, Rajagopalan, and Lindsay [39] extended this work with a deterministic solution that stores no more than $O(\varepsilon^{-1}\log N\varepsilon)$ objects, assuming previous knowledge of N. Though [40] has the same worst-case space bound, the algorithm is empirically better. In 2001, Greenwald and Khanna [27] developed a complex deterministic streaming algorithm, referred to as the GK-algorithm below, that stores $O(\varepsilon^{-1}\log(N\varepsilon))$ objects in the worst case. However, their experimental work used a simplified approach for which it is not clear if the $O(\varepsilon^{-1}\log(N\varepsilon))$ space limit still holds. Nonetheless, they

demonstrated that their method beats Manku et al. [39]'s approach in practice. Each of these methods is deterministic and relies on comparisons. The GK-algorithm is often considered the best in this area, both theoretically and experimentally. Indeed, Cormode and Veselý [21] recently showed that every deterministic comparison-based technique has a space constraint of $\Omega(\varepsilon^{-1}\log(N\varepsilon))$ items. We describe the GK algorithm in detail in Section 3.2.

Shrivastava et al [51] created q-digest in 2004, a deterministic, fixed universe method that consumes $O(\varepsilon^{-1}\log\mathcal{U})$ space, where $\mathcal{U}$ is the universe. This approach was developed to compute quantiles in sensor networks and is a mergeable summary [4], a more flexible model than streaming. However, no further efficient fixed-universe method exists in the streaming model. Note that the $\log\mathcal{U}$ and $\log N$ terms are not theoretically equivalent, and [51] omitted an experimental comparison with the GK-algorithm. The above works suggest a deterministic solution that tracks the quantiles of the entire stream, while our *deterministic* solution, SSGK, solves the quantiles problem for each significant item in the stream.

Randomized methods have also been considered in the past. The seminal results of [53] show that a random sample of size $O(\varepsilon^{-2}\log\varepsilon^{-1})$ contains all quantiles with at least a constant probability within the ε error. This fact was shown in [39] and was used to compute quantiles using a random sample fed to a deterministic algorithm. However, since this method needs knowledge of N in advance, it is not a true streaming algorithm.

Manku et al. [40] developed a randomized approach that does not require knowledge of N and demonstrated that the space required is $O(\varepsilon^{-1}\log^2\varepsilon^{-1})$ factor, which may be greater or smaller than GK's $\log(N\varepsilon)$ factor, although neither of these algorithms has been empirically tested.

Agarwal, Cormode, Huang, Phillips, Wei, and Yi [4] proposed a mergeable sketch whose size requirement is $O(\varepsilon^{-1}\log^{1.5}\varepsilon^{-1})$. For this new, simpler approach, called Random, Luo et al [38] were able to provide an improved $O(\varepsilon^{-1}\log^{1.5}\varepsilon^{-1})$ bound. We refer to this algorithm as "Random" and overview it in detail below. Felber and Ostrovsky [25] reduced the space complexity by using a combination of sampling and the GK-algorithm to $O(\varepsilon^{-1}\log\varepsilon^{-1})$.

Finally, Karnin, Lang, and Liberty [32] solved the problem by developing the KLL sketch, giving an optimal $O(\varepsilon^{-1})$-space solution. The KLL sketch achieves optimal accuracy in space. The algorithm's fundamental building component is a buffer called a compactor, which accepts an input stream of N items and generates a stream of no more than $\frac{N}{2}$ items that "approximates" the input stream. The overall KLL sketch is constructed as a series of at most $\log N$ compactors, with each compactor's output stream acting as the input stream for the next compactor. We note that our results are orthogonal to the above algorithms, which compute the quantiles of the entire input, in the sense that improved algorithms can replace the building blocks of our solution, further improving its speed and accuracy. All of the above papers provide a randomized solution that monitors the quantiles of the entire stream. In this paper, our compact algorithm, SQUAD, is a randomized algorithm that handles the fine-grained problem of approximating the quantiles for each significant item in the stream.

Several studies have attempted to provide more accurate quantile estimates for low and high rankings. Only a few provide answers to the relative error quantiles problem (also known as the biased quantiles problem). Gupta and Zane [29] presented an approach for computing relative error quantiles that saves $O(\varepsilon^{-3}\log^2(N\varepsilon))$ items and uses this to estimate the number of inversions in a list; their technique needs knowledge of the stream length, N. Zhang et al. [57] previously described an approach for storing $O(\varepsilon^{-2}\log(N\varepsilon^2))$ items. Cormode et al. [19] devised a deterministic sketch that stores $O(\varepsilon^{-1}\log(N\varepsilon\log(|\mathcal{U}|))$ elements and necessitates previous knowledge of the data universe $\mathcal{U}$. Shrivastava et al. [51]'s work on additive error has influenced their approach. Zhang and Wang [56] proposed a deterministic merge-and-prune method that stores $O(\varepsilon^{-1}\log^3(N\varepsilon))$

items and is capable of performing arbitrary merges with an upper constraint on n as well as streaming updates for unknown N. However, it does not address the most general case of merging without prior knowledge of N. Cormode et al. [18] presented a relative error variation of the KLL sketch. They achieve relative error ε in the randomized environment using $O(\varepsilon^{-1}\log^{1.5}(N\varepsilon))$ with constant failure probability by varying the sampling technique throughout the distribution and employing a hierarchy modeled after [32]. Yet, these works address the problem of quantiles over the entire stream and not per (significant) item.

3 PRELIMINARIES

3.1 Model

We consider a *stream* (sequence) of tuples $\mathcal{S} = \langle (x_1, v_1), (x_2, v_2) \ldots \rangle \in (\mathcal{U} \times \mathbb{R})^+$. Here, each element (x_i, v_i) has an *identifier* x_i from a universe $\mathcal{U}$, and a *value* $v_i \in \mathbb{R}$.

We denote by $f_x = |\{(x_i, v_i) \in \mathcal{S} : x_i = x\}|$ the frequency (size) of x and by $L_x = \{v_i : (x, v_i) \in \mathcal{S}\}$ its (multi-) set of values.

Given a *quantile* $q \in [0, 1]$, let $\mathcal{L}_{x,q}$ represent the q^{th} quantile (i.e., the $\lceil q \cdot f_x \rceil^{th}$ largest value) of L_x. The inverse operation is normalized rank, denoted by $\mathsf{rank}_x(v)$, which returns the quantile of v in L_x (that is, $\mathsf{rank}_x(\mathcal{L}_{x,q}) = q$). Any item x with frequency $f_x \geq N\theta$ is called a heavy hitter, where $N = |\mathcal{S}|$ is the overall number of elements, and $\theta \in [0, 1]$ is a given *threshold*.

We use $\varepsilon, \delta \in [0, 1)$ to denote the error parameters; given the parameters $\theta, \varepsilon, \delta$, we consider the $(\theta, \varepsilon, \delta)$-HH-quantiles that tracks the quantiles of heavy hitters. Specifically, we seek algorithms that support the following operations:

- Insert(x, v) — given an element $x \in \mathcal{U}$ and $v \in \mathbb{R}$, append (x, v) to $\mathcal{S}$.
- Query(x, q) — given $x \in \mathcal{U}$ and $q \in [0, 1]$, return a tuple $(\widehat{f_x}, \widehat{\mathcal{L}_{x,q}})$ where $\widehat{f_x}$ is the frequency estimation of x and $\widehat{\mathcal{L}_{x,q}}$ is an estimation of $\mathcal{L}_{x,q}$.

We now formalize the required guarantees.

Definition 1. *An algorithm solves $(\theta, \varepsilon, \delta)$-HH-quantiles if given any Query(x, q), the returned tuple $(\widehat{f_x}, \widehat{\mathcal{L}_{x,q}})$ satisfies:*

(1) $\Pr[|f_x - \widehat{f_x}| > N\varepsilon] \leq \delta$. *As standard in heavy hitter algorithms, we return an estimate of the item's frequency.*
(2) *If $f_x \geq \theta N$, $\Pr[|\mathsf{rank}(\widehat{\mathcal{L}_{x,q}}) - q|) > \varepsilon] \leq \delta$. That is, if x is a heavy hitter the algorithm returns an estimate whose quantile is likely (with probability $1 - \delta$) to be off by no more than ε.*

We note that part (1) of our query response is designed to help the user understand whether the quantile estimate is reliable and a similar guarantee can be obtained by running a separate heavy hitters algorithm. Specifically, if $\widehat{f_x} > N(\theta + \varepsilon)$, then x is likely to be a heavy hitter and therefore $\widehat{\mathcal{L}_{x,q}}$ is a credible approximation of $\mathcal{L}_{x,q}$. Table 2 has a summary of basic notations used in this work.

3.2 Useful Streaming Algorithms

In this work, we utilize the Reservoir Sampling (RS) algorithm [54] in Section 4 as well as the Space Saving (SS) algorithm [43] and the GK-algorithm [27] in Section 5. We overview them here.

Reservoir sampling (RS) [54]: is a randomized algorithm for selecting a uniform random sample of a given size from an input stream of an unknown size without replacement in a single pass through the objects. The algorithm keeps a k-sized reservoir, which initially holds the first k items of the input. On the arrival of the n'th item, RS selects a uniform random integer $i \in \{0, \ldots, n-1\}$; the item overrides slot i of the reservoir if $i < k$ and otherwise discarded.

Table 2. List of Symbols

Symbol	Meaning
$\mathcal{S}$	The data stream
$\mathcal{U}$	The universe of elements
$\mathcal{R}$	The universe of values
N	The number of elements in the stream
q	The quantile q i.e. the q^{th} largest value
L_x	The set of values of $\mathcal{S}$ with identifier x
$\mathcal{L}_{x,q}$	The q^{th} quantile of L_x
$\widetilde{\mathcal{L}_{x,q}}$	an estimation of $\mathcal{L}_{x,q}$
$\text{rank}_x(v)$	The quantile of v in $\mathcal{S}_x$
f_x	The frequency of an element x in $\mathcal{S}$
$\widehat{f_x}$	An estimate of f_x
ε	An estimate accuracy parameter
δ	A bound on the failure probability
θ	The heavy hitters threshold

Space Saving (SS) [43]: is a counter-base algorithm for (approximately) finding the most frequent items in a data stream, a.k.a. the heavy hitters. SS processes a stream of identifiers with the goal of estimating the size (frequency) of each. SS maintains a set of $1/\varepsilon$ integer counters, each with an associated ID. When an item arrives, SS increments its counter if one exists. Otherwise, SS allocates the item with a minimal-valued counter before incrementing it (disassociating the previous ID). For example, assume that the smallest counter was associated with ID x and had a value of 4; if y arrives and has no counter, it will take over x's counter and increment its value to 5 (leaving x without a counter). When queried for the frequency of an item, we return the value of its counter if it has one, or the minimal counter's value otherwise.

If we denote the overall number of insertions processed by the algorithm by Z, then we have that the sum of counters equals Z, and thus the minimal counter is at most $Z\varepsilon$. This ensures that the error in the SS estimate is at most $Z\varepsilon$.

The GK-algorithm (GK) [27]: is a deterministic algorithm for supporting single-pass quantile summaries of a number stream. A quantile summary is a subset of the input data sequence that uses quantile estimations to provide approximate answers to any arbitrary quantile query.

The GK technique is based on the idea that if a sorted subset of the input stream of size N can be kept so that the ranks of v_i and v_{i+1} are within $2N\varepsilon$ of each other, then any quantile query can be answered with an error no larger than $N\varepsilon$. That is, given a quantile q, GK returns an element whose quantile falls within $[q-\varepsilon, q+\varepsilon]$.

GK allows maintaining such a subset using $O(\frac{1}{\varepsilon}\log N\varepsilon)$ elements, which has recently been shown to be optimal for comparison-based deterministic algorithms [21].

The Random algorithm [38]: a randomized algorithm that reports all quantiles within the specified error with constant success probability, e.g., 2/3, that can be amplified with repetition.

Random separates the stream into fixed-size buffers, each of which is assigned a level. Whenever there are two buffers at the same level, Random merges them into a buffer at one level higher, such that at any time, there is at most one buffer at any level. Random aggregates the ranks of x in all buffers to report the rank of an element x. Overall, it requires keeping $O(\frac{1}{\varepsilon}\log^{1.5}\frac{1}{\varepsilon})$ elements to guarantee that for any q, it returns an element whose quantile falls within $[q-\varepsilon, q+\varepsilon]$ with constant probability. If we aim for a specific quantile, then the space reduces to $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ per repetition.

4 SAMPLING APPROACH

Sampling was introduced by Vapnik and Chervonenkis [53]. It is a popular method that is useful for many applications, including quantile estimation, and will serve as a baseline for our more complex algorithms. An online sampling method, such as Reservoir Sampling (RS) of Vitter [54], can yield a random sample of the data stream almost linearly in time and space with respect to the sample size. We analyze the number of samples needed for our problem.

THEOREM 1. *Sampling solves the $(\theta, \varepsilon, \delta)$-HH-quantiles while requiring $\Theta(\theta^{-1}\varepsilon^{-2}\log\delta^{-1})$ space.*

PROOF SKETCH. For any QUERY(x, q), a random sample of size $\Theta\left(\varepsilon^{-2}\log\delta^{-1}\right)$ elements from a stream is an ε-approximation of frequencies and quantiles with probability at least $1-\delta$ [40].

As a result, any heavy hitter x (an ID that appears at least $N\theta$ times) must be sampled $\Omega(\varepsilon^{-2}\log\delta^{-1})$ times to solve the $(\theta, \varepsilon, \delta)$-HH-quantiles using sampling.

Notice that there can be $\Theta(\theta^{-1})$ heavy hitters. As a result, one can use RS to sample $M = \Theta(\theta^{-1}\varepsilon^{-2}\log\delta^{-1})$ elements from $\mathcal{S}$. This way, a given heavy hitter is sampled $\Omega(\varepsilon^{-2}\log\delta^{-1})$ times with probability $1-\delta/2$, and its samples produce an appropriate quantile estimate with probability $1-\delta/2$; using the union bound, we get that the overall estimate is accurate with probability $1-\delta$.

Further, by selecting $\widehat{f_x}$ to be N/M times its frequency in the sample (e.g., see the analysis of [8]), we can estimate the frequency of an item to within an $N\cdot\varepsilon\sqrt{\theta}$ factor with probability $1-\delta$. □

For example, in Figure 1, $M = 5, N = 9$, and for a Query(x, q), we estimate x frequency estimation using the samples (x, v_4), (x, v_9) (i.e., $\widehat{f_x} = 2\cdot 9/5$), and we estimate the q quantile on v_4, v_9.

5 SKETCHING APPROACH: SSGK ALGORITHM

Next, we offer a sketching algorithm called Nested Space-Saving GK algorithm (SSGK). It is a **deterministic** algorithm, which is based on a nested application of Space-Saving and GK algorithms. Intuitively, SSGK allocates a separate GK sketch [27] to track the quantiles of each potential heavy hitter. Because we do not know the IDs of the heavy hitters ahead of time, SSGK uses a space-saving instance with $k = 2\varepsilon^{-1}\theta^{-1}$ entries, where each entry has a GK sketch instance configured for error $\varepsilon_{GK} = \varepsilon/2$ in addition to its counter and ID fields. This way, SSGK can use the SS counter value to estimate the frequency and use the GK sketch to approximate the quantile.

Whenever an item (x, v) arrives, if x has an allocated counter, SSGK increments x's counter and inserts v to the associated GK instance. Otherwise, we replace the item with the minimal counter value with x and *reset* its corresponding GK instance. Then, we insert v to this GK instance. In Figure 1's example, we keep the first three elements exactly as we have space for them in the SS. When (x, v_4) arrives, we replace y with x, increasing its counter to 2, and inserting v_4 into the (reset) quantile sketch. Later, the arrivals of (x, v_7) and (x, v_9) increase x's counter and add v_7 and v_9 to the sketch. Only $\{v_4, v_9\}$ are stored in the quantile sketch because it keeps only a ($\widetilde{O}(\varepsilon^{-1})$-sized) subset of the input values.

Using the SS variant mentioned above, we can compute any QUERY(x, q) as follows. If x has an allocated SS entry, we estimate its frequency using its counter value. Its GK instance then estimates the q^{th} quantile of L_x. Otherwise, if x has no allocated entry, we estimate the minimal SS counter value as (an upper bound on) its frequency and do not report the quantile. Since SS deterministically guarantees that every element with a frequency larger than $N/k \le N\theta$ (i.e., in particular, every heavy hitter) will have an entry, we can satisfy the accuracy guarantees.

Algorithm 1 provides a high-level pseudo code of SSGK using the pseudocode of Space Saving as shown in [43] without the implementation details. The additions to manipulate the GK sketch instances are highlighted in blue. Table 3 contains a list of the used variables. We summarize the analysis in the following theorem.

Algorithm 1 SSGK

```
1:  function INSERT(x, v)
2:      if x is monitored then
3:          Increment count_x, the counter of x
4:          Insert v to GK_x, the GK sketch of x
5:      else
6:          if Less than k items are monitored then
7:              count_x ← 1
8:              Initialize a GK sketch for x
9:          else
10:             Let x' be the element with smallest count_x'
11:             Start monitoring x instead of x';
12:             count_x ← count_x' + 1
13:             Reset the GK sketch for x
14:         Insert v to the GK sketch of x
15: function QUERY(x, q)
16:     if x is monitored then
17:         return (count_x, GK_x.Quantile(q))
18:     else
19:         return (count_min, undefined)
```

Table 3. Variables used by SSGK (Algorithm 1)

k	number of entries in the SS
$count_x$	counter of x in the SS
$count_{\min}$	minimal counter value in the SS
GK_x	the GK sketch instance of x

THEOREM 2. *SSGK solves $(\theta, \varepsilon, 0)$-HH-quantiles (i.e.,* deterministically*) while requiring $O\left(\theta^{-1}\varepsilon^{-2} \cdot \left(1 + \log(N\varepsilon^2\theta)\right)\right)$ space.*

PROOF SKETCH. Using the standard analysis for an SS instance with k entries, we have every heavy hitter receive a space-saving instance no later than its N/k arrival (because the counters' sum is at most N, the minimal one cannot be greater than N/k).

Therefore, if the queried element has no counter, it cannot be a heavy hitter. Otherwise, the GK sketch of the queried heavy hitter x processes all but at most $N/k = N\theta\varepsilon/2$ values from L_x. Let $L'_x \subseteq L_x$ denote the subset of values processed by GK_x. Due to GK_x's guarantees, our output $\widehat{q} = GK_x.Quantile(q)$ is within ε_{GK} from the true quantile of L'_x. That is, $\widehat{q}$ deviates from the true quantile by at most $|L'_x| \cdot \varepsilon_{GK} = |L'_x| \cdot \varepsilon/2$ values. Together with the missing values of $L_x \setminus L'_x$, we have that $\widehat{q}$ deviates by at most $|L'_x| \cdot \varepsilon/2 + N\theta\varepsilon/2$. In terms of quantiles, this means that our error is

$$\frac{|L'_x| \cdot \varepsilon/2 + N\theta\varepsilon/2}{|L_x|} \leq \varepsilon/2 \cdot \left(1 + \frac{N\theta}{|L_x|}\right) \leq \varepsilon.$$

Let us analyze the space next. Let a_i be the number of times the i'th GK instance was incremented; therefore, $\sum_{i=1}^{k} a_i \leq N$. By the space complexity of the GK algorithm we have that SSGK's overall space requirement is

$$\sum_{i=1}^{k} O(\varepsilon^{-1}(1 + \log(a_i\varepsilon))) = O\left(k \cdot \varepsilon^{-1} \cdot \left(1 + \log\frac{N \cdot \varepsilon}{k}\right)\right)$$
$$= O\left(\theta^{-1}\varepsilon^{-2} \cdot \left(1 + \log(N\varepsilon^2\theta)\right)\right).$$

Here, we used Jansen's inequality and the concaveness of the logarithm function. □

6 THE SQUAD ALGORITHM

The quadratic dependency on $1/\varepsilon$ of the previous algorithms is sometimes prohibitively costly. Interestingly, while both the sampling (Sampling) and sketching (SSGK) approaches require $\widetilde{\Omega}(1/\varepsilon^2)$ space[1], applying them in tandem we can significantly improve the space complexity. Specifically, we present Sketching-Sampling QUAntiles Duo (SQUAD), a hybrid algorithm that requires only $\widetilde{O}(\varepsilon^{-1.5})$ space[1] and combines *sampling* and *sketching* approaches. Intuitively, sampling helps us capture the behavior of the values of an item before it was allocated with an SS entry and a quantile sketch.

SQUAD keeps $z = O(\varepsilon^{-1.5}\theta^{-1}\log\delta^{-1})$ samples chosen by RS (see section 3.2). Each sample is now a *triplet* $(ID, value, timestamp)$. That is, when sampling an element (x_i, v_i) we store the triplet (x_i, v_i, i). We will later use the timestamp for merging the sample with the sketch. Additionally, SQUAD employs a nested Space Saving [43] (SS) as described in SSGK (Section 5), but instead of using instances of the GK algorithm, it employs instances of the Random algorithm (configured for $\varepsilon/2$ error similarly to SSGK).[2]

In contrast to SSGK, which requires $k = \Theta(\varepsilon^{-1}\theta^{-1})$ entries, SQUAD only uses $m = 4\varepsilon^{-0.5}\theta^{-1}$. Intuitively, we can avoid sketching additional values from L_x because the ones processed before x is allocated to a sketch, are well approximated by the sample.

In addition to the counter and the Random instance, an entry for ID x in the SS structure has a timestamp (t_x) that indicates when x was (last) allocated with an entry and an additional value (I_x) that represents the number of times x arrived since it received the entry.

If RS decides to consider the i'th element (x_i, v_i), we store (x_i, v_i, i) in the samples array. After that, we update the augmented SS as follows: If x_i has an allocated counter, SQUAD increments it and inserts v_i into the associated random instance. Otherwise, we reallocate the counter with the lowest value for x, flush its Random instance, and set its t_{x_i} to i. After that, we add v to this Random instance. Notice that x continues to participate in the RS process *regardless of whether it has a counter in SS or not.* Intuitively, x's entry could become minimal and it could be evicted from the SS, so we keep tracking it by sampling. In Figure 1, SQUAD employs two entries in the SS and a sample size of three. (a, v_3) and (x, v_4) arrive at $t = 3$ and $t = 4$, respectively, replace the entries of y and z, update their entries, and are (by chance) selected for the sample as the triples $(a, v_3, 3)$, $(x, v_4, 4)$. Since a already has an entry at $t = 5$, (a, v_5) simply updates its entry. The arrival of (b, v_6) replaces x entry because it has the smallest counter (2), then (x, v_7) replaces b entry and increases its counter to 4, indicates that x has an allocated entry at $t = 7$, and inserts v_9 into the sketch. (c, v_8) replaces a entry at $t = 8$, and at $t = 9$, (x, v_9) updates x entry in the SS, and SQUAD includes it in the sample as $(x, v_9, 9)$.

For answering Query(x, q), we search for x in the augmented SS. If x does not have an entry, our algorithms cannot promise anything about its q^{th} quantile (similarly to SSGK, this means that x is not a heavy hitter). To estimate the frequency of an item x, we use *both the sample and the SS counter.* Specifically, let t_x denote the timestamp of x in the SS, and let S_x represent the number of samples that belong to x with a timestamp smaller than t_x. We estimate the number of times that x arrived *before* t_x as $N/z \cdot S_x$ because the probability that RS samples a specific element is z/N. As a result, we estimate the frequency as $\widehat{f_x} = N/z \cdot S_x + I_x$.

[1]The $\widetilde{\Omega}$, $\widetilde{\Theta}$ and $\widetilde{O}$ notations assume that the heavy hitters parameter θ is constant and hide polylogarithmic factors.
[2]We chose Random as it is the fastest algorithm we are aware of, and our guarantee is probabilistic anyhow due to the sampling. One can replace it with a state-of-the-art sketch such as KLL [32], slightly improving the accuracy at a potential loss of speed, or with GK. In any case, the complexity remains $\widetilde{\Theta}(\varepsilon^{-1.5})$.

Algorithm 2 SQUAD

```
1: function Insert(x, v)
2:     n ← n + 1                                   ▷ The current timestamp
3:     RS.Add(x, v, n)
4:     if x is monitored then
5:         Increment count_x, the counter of x
6:         Increment I_x, the count since x became monitored
7:         Insert v to Rnd_x, the Random sketch of x
8:     else
9:         if Less than m items are monitored then
10:            Initialize a Random sketch for x
11:            count_x ← 1
12:        else
13:            Let x' be the element with smallest count_x'
14:            Start monitoring x instead of x';
15:            count_x ← count_x' + 1
16:            Reset the Random sketch for x
17:        I_x ← 1
18:        t_x ← n
19:        Insert v to the Random sketch of x
20: function Query(x, q)
21:     S_x ← 0
22:     SList_x ← empty list
23:     for j ∈ 0, 1, ..., z do
24:         if RS[j].ID = x and RS[j].ts < t_x then
25:             S_x ← S_x + 1
26:             Insert RS[j].v to SList_x
27:     sf_x = (n/z) · S_x
28:     if x is monitored then
29:         RndNew_x = Rnd_x
30:         Insert samples from SList_x with weight n/z to RndNew_x
31:         return (sf_x + I_x, RndNew_x.Quantile(q))
32:     else
33:         return (sf_x, undefined)
```

Quantiles are estimated similarly: we take the samples collected before t_x as representing the values before t_x and merge them with the in $Random_x$ (that represent entries between t_x and N).

To merge the samples and sketch, we duplicate $Random_x$ and then insert the S_x samples, each with a weight of N/z. Our q'th quantile approximation is then the quantile of the combined array.

An alternative approach is to merge the samples of x, $Samples_x$, with the buffers of the Random algorithm inside buffers in the function Query(x, q), and then report the rank of x using the merged buffers. This approach requires an additional modification in the implementation of the Random's Quantile(q) function.

The variables of the SQUAD algorithm are described in Table 4 and its pseudocode appears in Algorithm 2. We summarize the analysis in the following theorem.

Theorem 3. *SQUAD solves $(\theta, \varepsilon, \delta)$-HH-quantiles while requiring $O\left(\theta^{-1}\varepsilon^{-1.5} \cdot \log \varepsilon^{-1}\right)$ space.*

Proof. Our analysis relies on the observation that if the sample approximates x's frequency before t_x to within an α additive error, and the space saving approximates its frequency since t_x to

Table 4. Variables used by SQUAD (Algorithm 2)

n	number of arrived elements
z	samples size used by RS
RS	a Reservoir Sampling instance with a most z samples.
m	number of entries in the SS
$count_x$	counter of x in the SS
I_x	the count since x became monitored
t_x	timestamp of x in the SS
S_x	number of samples that belongs to x
sf_x	estimation of x's frequency before t_x
$Rand_x$	the Random instance of x

within β elements, then the error of the merging process cannot exceed $\alpha + \beta$; a similar logic also applies to the quantiles (e.g., see [28]).

Let us start by analyzing the sample. Let $f_{x,1}$ denote x's frequency before (not including) t_x and let $f_{x,2}$ denote its frequency starting with t_x (i.e., $f_x = f_{x,1} + f_{x,2}$). As before, we denote by S_x the number of x's samples collected before t_x (observe that $S_x \sim \text{Hypergeometric}(N, f_{x,1}, z)$). Thus, we use the approximation $\widehat{f_{x,1}} = S_x \cdot N/z$. Denoting $p = f_{x,1}/N$, we can use standard concentration bounds (e.g., [50]) on the hypergeometric distribution to bound the sampling error as, for any $\Delta \in (0, z \cdot p]$

$$\Pr\left[|S_x - \mathbb{E}[S_x]| \geq \Delta\right] < 2e^{-\frac{\Delta^2}{3z \cdot p}}. \tag{1}$$

Notice that once an item that reaches a frequency of N/m it cannot have the minimum SS entry. Therefore, we have that $f_{x,1} \leq N/m$. As the sampling error is monotonically increasing in $f_{x,1}$ (for $f_{x,1} < N/2$), we bound the error by analyzing the error of an item with $f_{x,1} = N/m$. In our context, we sample $z = O(\varepsilon^{-1.5}\theta^{-1}\log\delta^{-1})$ elements from the stream; that is, the probability for each of the z samples to belong to the first $f_{x,1}$ insertions of x is $p = \frac{f_{x,1}}{N} = 1/m$.

Next, let $\Delta = \sqrt{\frac{3z\log(2/\delta)}{m}} = \Theta\left(\sqrt{\frac{z\log\delta^{-1}}{m}}\right)$. [3] Our goal in what follows is to show that the error in estimating $f_{x,1}$ is likely to be lower than $N \cdot \sqrt{\frac{3\log(2/\delta)}{z \cdot m}} = \Theta(N \cdot \varepsilon \cdot \theta)$. Using (1), we have that:

$$\begin{aligned}
&\Pr\left[|\widehat{f_{x,1}} - f_{x,1}| \geq N \cdot \sqrt{\frac{3\log(2/\delta)}{z \cdot m}}\right] \\
&\qquad = \Pr\left[|S_x \cdot N/z - \mathbb{E}[S_x] \cdot N/z| \geq \frac{N}{z} \cdot \Delta\right] \\
&\qquad = \Pr\left[|S_x - \mathbb{E}[S_x]| \geq \Delta\right] \leq 2e^{-\frac{\Delta^2}{3z \cdot p}} = \delta.
\end{aligned}$$

Next, recall that $f_{2,x}$ is calculated accurately using I_x, and thereby $\left|\widehat{f_x} - f_x\right| = \left|\widehat{f_{x,1}} - f_{x,1}\right|$. Therefore, we established that the frequency estimation error is bounded by $N \cdot \varepsilon \cdot \theta$, with probability $1 - \delta$, using $z = c \cdot \varepsilon^{-1.5}\theta^{-1}\log\delta^{-1}$ samples, for an appropriate constant $c > 0$.

To analyze the quantile estimation error, we consider the error of the sampling phase (before t_x) separately from the error once x is allocated with a sketch (starting with t_x). An analysis similar to

[3] Observe that $z \cdot p = z/m = \Theta(\varepsilon^{-1}\log\delta^{-1})$, and therefore:

$$\Delta = \Theta\left(\sqrt{\frac{z\log\delta^{-1}}{m}}\right) = \Theta\left(\sqrt{\varepsilon^{-1}} \cdot \log\delta^{-1}\right) = o\left(z \cdot p\right).$$

the above (with different constants) yields that the error in the sampling phase is bounded by $\varepsilon/2$ except with probability $\delta/2$. Specifically, we can get $S_x = \Omega(z \cdot f_{x,1}/N) = \Omega(\varepsilon^{-1} \log \delta^{-1})$ samples except with probability $\delta/4$, and have these approximate the quantile within an additive $\Theta(\sqrt{\varepsilon})$ error except with further $\delta/4$ error probability. This means that the rank of the value is off by at most $\Theta(\sqrt{\varepsilon} \cdot f_{x,1}) = \Theta(\sqrt{\varepsilon} \cdot N/m) = \Theta(N\varepsilon\theta)$ from the true quantile. Therefore, by configuring the quantile sketch to have an $\varepsilon/2$ error with probability $1 - \delta/2$, we can get that the overall estimate error, which results from the combination of the sample and the sketch, is bounded by $f_x \cdot \varepsilon + \Theta(N\varepsilon\theta) = O(f_x\varepsilon)$, as $f_x \geq N\theta$ per our problem definition. □

7 OPTIMIZING THE PROCESSING SPEED

We now detail several optimizations that enable our algorithms to process elements faster. First, we use the Algorithm L [35], which provides a fast simulation of RS. Intuitively, instead of drawing a random integer per item, it generates geometric random variables that represent how many items to skip before the next one is admitted into the reservoir. Once an item is chosen, it replaces a uniform slot in $\{0, \ldots, k-1\}$. As a result, the total number of updates falls to $O(k(1 + \log(N/k))$, implying that it takes $o(1)$ computation per element because the majority are skipped.

While we can use the above to optimize the RS process, SSGK and SQUAD are bottlenecked as arrivals of elements not tracked by the SS require initializing a new sketch. To speed up the processing of both, we propose using an initial probabilistic filtering stage. Intuitively, as both quantiles and frequencies can be accurately estimated from sampled streams for heavy hitters, we can process a small (e.g., 10%) of the input and obtain rather precise results. Namely, consider a wrapper that with probability $\mathfrak{p}$ calls the INSERT function of SQUAD (or SSGK, although this makes the algorithm randomized.) and otherwise ignores the packet. This means that the algorithms look at a *sampled stream* $\mathcal{S}' \subset \mathcal{S}$ such that each element in $\mathcal{S}$ i.i.d. appears with probability $\mathfrak{p}$ in $\mathcal{S}'$.

Intuitively, the sampling error would be of size $\Theta(\sqrt{N\theta/\mathfrak{p}})$; if this is comparable or smaller than the $\Theta(N\varepsilon\theta)$ error of SQUAD, we can compensate for the error resulting from analyzing $\mathcal{S}'$ (rather than $\mathcal{S}$) without asymptotically increasing the space requirements.

There are several approaches to selecting $\mathfrak{p}$. One option is to dynamically change $\mathfrak{p}$ as N grows, inserting elements with a weight of $1/\mathfrak{p}$, e.g., as suggested by [10, 36]. For simplicity, here we consider using a fixed probability, which means that the accuracy guarantees of the algorithms only hold after a short *convergence time* (as common in some sampling algorithms [9, 13, 41]). Namely, consider setting SQUAD to solve the $(\theta, \alpha \cdot \varepsilon, \alpha \cdot \delta)$-HH-quantiles for some $\alpha \in (0, 1)$ (e.g., $\alpha = 0.9$). Then, if the frequencies and the quantiles are maintained in $\mathcal{S}'$ (the frequency, after scaling by $1/\mathfrak{p}$) to within error $(1-\alpha)\varepsilon$, except with probability $(1-\alpha)\delta$, then the overall scheme solves $(\theta, \varepsilon, \delta)$-HH-quantiles. Here, α is a tradeoff parameter: the larger α is, the less space the algorithm requires, but also the higher the sampling probability needs to be.

As analyzed above, a sample of size $|\mathcal{S}'| = \Omega(\theta^{-1}\varepsilon_s^{-2} \log \delta_s^{-1})$ is enough for $\mathcal{S}'$ to be an ε_s approximation of the quantiles and frequency (the $_s$ subscript represents sampling) of an element except with probability δ_s. In our case, we have $|\mathcal{S}'| = N\mathfrak{p}$, i.e., $N\mathfrak{p} = \Omega(\theta^{-1}\varepsilon_s^{-2} \log \delta_s^{-1})$, and thus we need a convergence time of at least $N = \Omega(\theta^{-1}((1-\alpha)\varepsilon)^{-2}\mathfrak{p}^{-1} \log((1-\alpha)\delta)^{-1})$ elements before the algorithm solves the $(\theta, \varepsilon, \delta)$-HH-quantiles.

Intuitively, since in practical applications we often have $N \gg \theta^{-1}\varepsilon^{-2} \log \delta^{-1}$ [1, 36], we can set a large α value (e.g., $\alpha = 0.9$). We can then use an intermediate value for $\mathfrak{p}$ (e.g., $\mathfrak{p} \in [0.01, 0.1]$) as this gives a large speed boost and lowering the sampling probability further is not as beneficial. This way, we do not require significantly more space (e.g., about 20% increase for $\alpha = 0.9$) nor compromise the accuracy guarantees (following the short convergence time) while significantly accelerating the solution.

8 EVALUATION

8.1 Setup

We developed a C++ prototype for each of the algorithms mentioned in this paper: Sampling, SSGK and SQUAD. The SSGK and SQUAD are implemented here using the Random sketch [38] as a building block. Additionally, we compared our results to the GK [27] and Random [38] algorithms as a general baseline, since these are the state of the art for the more basic problem of quantiles across whole data streams, rather than per-item quantiles. To to the best of our knowledge, this is the first study that solves quantiles on a per-item level. Furthermore, we compared to Space Saving (SS) [43], since this is a building block in SSGK and SQUAD.

8.1.1 Dataset: We evaluate our algorithms using NS3 simulations [3] for a FatTree topology comprised of 16 Core switches, 20 Agg switches, 20 ToRs, and 320 servers (16 in each rack). Each server has a single 100Gbps NIC and the default load is 60%. Each connection between Core and Agg switches, as well as between Agg switches and ToRs, has a capacity of 400Gbps. The switch buffer size is 32MB. The traffic follows the flow size distribution in web search from Microsoft [5] or Hadoop from Facebook [48].

The evaluation was performed on an Intel 3.20GHz Xeon(R) CPU E5-2667 v4 with Linux kernel 4.4.0-71. Each data point in all runtime measurements is shown as a 95% confidence interval of 10 runs.

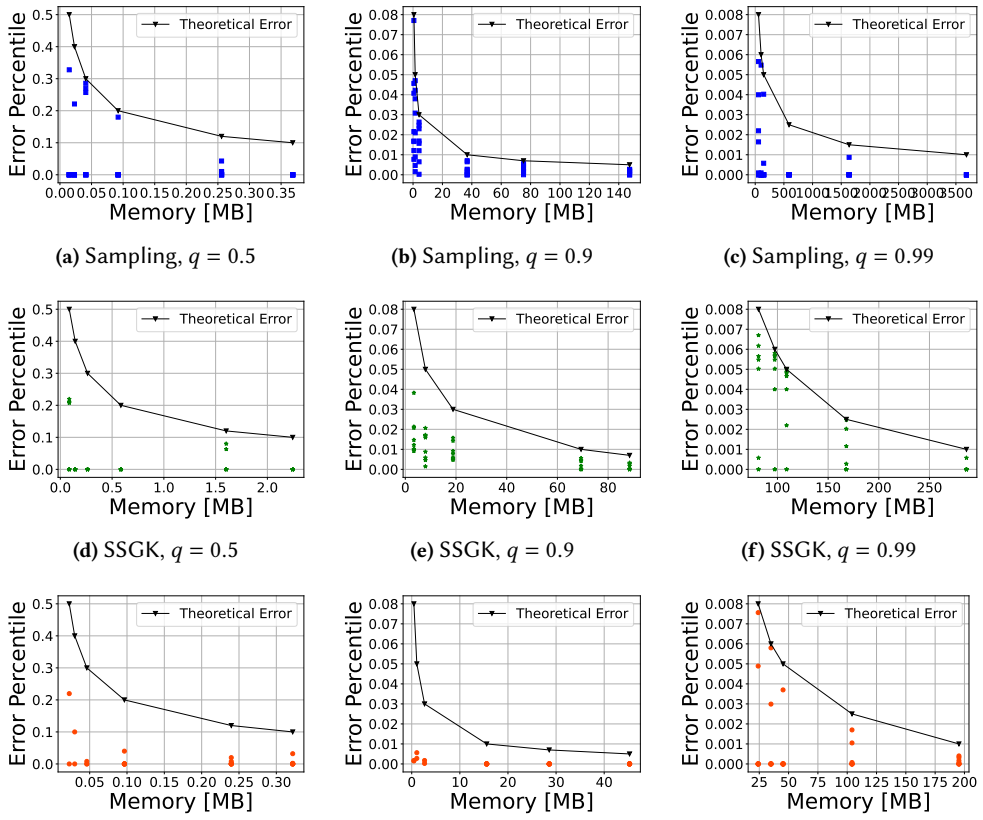
8.2 Accuracy Comparison

We measure accuracy in this experiment as a function of used memory. Specifically, given quantile q, we measure $|\mathsf{rank}(\widehat{\mathcal{L}_{x,q}}) - q|$), a.k.a *percentage error*, as a function of consumed memory for each x that satisfies $f_x \geq N\theta$. Additionally, we present the theoretical error (ε) which demonstrates that the empirical error is constrained by the theoretical error.

Figure 2 illustrates the percentage error in terms of quantiles: $q = 0.5, 0.9, 0.99$ for each algorithm: Sampling, SSGK, and SQUAD as a function of memory use with a constant value of $\theta = 0.01$ using NS3-simulated online search trace. Each point in the graphs represents the percentage error for a heavy-hitter. Note that all graphs have the same amount of points, but some of them overlap in several graphs. Additionally, Figure 3 illustrates the percentage error for SQUAD in terms of quantiles: $q = 0.5, 0.9, 0.99$ using an NS3-simulated trace following the Hadoop flow size distribution.

As shown in Table 1, the memory consumption of the three algorithms relies on ε and θ. Since we chose a fixed $\theta = 0.01$, higher memory consumption results in a lower ε, which leads to lower empirical error in all algorithms. Thus, as memory use increases, our algorithms get more precise, resulting in a decrease in empirical error that is lower than the theoretical value. As can be seen, SQUAD is the most compact algorithm among Sampling and SSGK, whereas Sampling is the most resource-intensive. As previously stated, Sampling stores $\Theta(\theta^{-1}\varepsilon^{-2}\log\delta^{-1})$ elements from the stream. To ensure a small error of ε, Sampling should keep a high number of samples, which results in saving the whole stream size in small values of ε and θ, as seen in Figure 2c.

For the SSGK algorithms, keeping the heavy hitters in the SS instance together with their GK-algorithm sketch results in a smaller footprint than Sampling. SQUAD, on the other hand, is the most efficient algorithm for solving the $(\theta, \varepsilon, \delta)$-HH-quantiles due to its compact data structure, as seen in Table 1. In general, a lower space consumption required for a specific ε and θ values translates into better empirical error. For example, SSGK consumes more memory than SQUAD for the same ε and θ. Thus, for a given memory budget, SSGK is more accurate than Sampling and SQUAD is more accurate than both, resulting in smaller error percentiles.



(a) Sampling, $q = 0.5$ **(b)** Sampling, $q = 0.9$ **(c)** Sampling, $q = 0.99$

(d) SSGK, $q = 0.5$ **(e)** SSGK, $q = 0.9$ **(f)** SSGK, $q = 0.99$

(g) SQUAD, $q = 0.5$ **(h)** SQUAD, $q = 0.9$ **(i)** SQUAD, $q = 0.99$

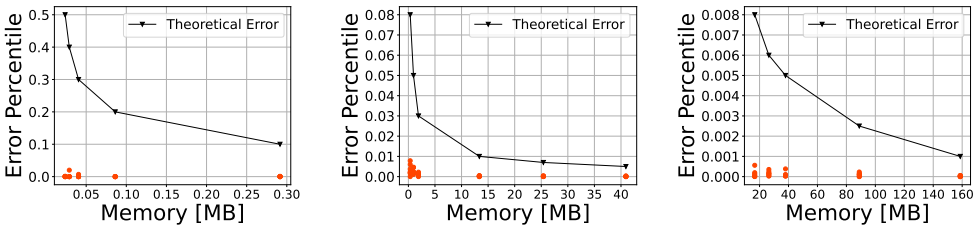
Fig. 2. Accuracy as a function of used memory using NS3-simulated online search trace. Each marker corresponds with one heavy hitter; i.e., we show the error ($|\mathsf{rank}(\widehat{\mathcal{L}_{x,q}}) - q|$)) for each x that satisfies $f_x \geq N\theta$ for a fixed value of $\theta = 0.01$, as a function of memory consumed. We examine the quantiles $q = 0.5, 0.9, 0.99$ of each algorithm: Sampling, SSGK, and SQUAD. Notice the different x/y-axis ranges.

Figure 4a illustrates SQUAD percentage error in correlation with θ using a Hadoop dataset with fixed $\varepsilon = 0.025$ and $q = 0.9$. Higher θ values, as expected, reduce the number of tracked heavy hitters because there are fewer items with frequencies greater than θN. SQUAD requires a smaller sample size and fewer entries in its Space Saving component in this case, which decreases the space consumption. Yet, the observed errors are lower than the user-selected value (ε) for all θ values.

8.3 Performance Comparison

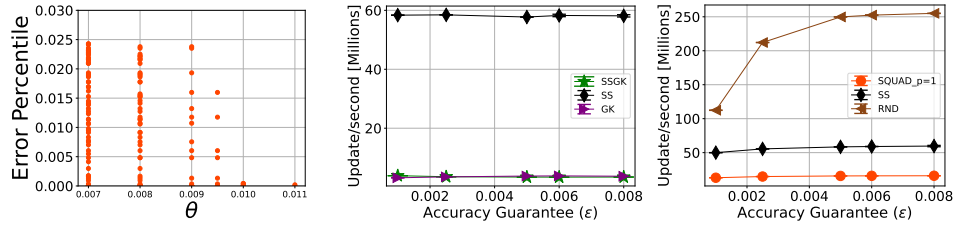
Figures 4b, 4c and 5 compare the update speed via an NS3-simulated online search trace. The Hadoop trace evaluation yields very similar results. We explore the trade-off of ε with a fixed $\theta = 0.01$.

8.3.1 SSGK Update Time: Figure 4b illustrates the performance of SSGK in terms of update time when compared to its building blocks: Space Saving (SS) [43] and the GK-algorithm [27]. Recall that although SS is the quickest, neither it nor the GK-algorithm solve the $(\theta, \varepsilon, \delta)$-HH-quantiles and rather serve as a best-case reference point. As can be observed, SSGK's update performance is



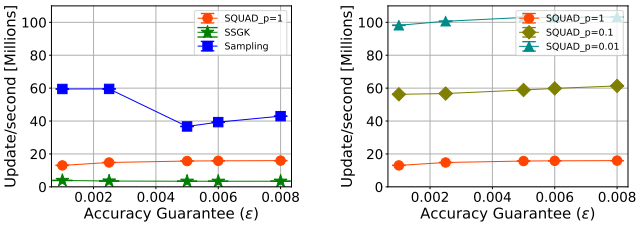
(a) SQUAD, $q = 0.5$ **(b)** SQUAD, $q = 0.9$ **(c)** SQUAD, $q = 0.99$

Fig. 3. Accuracy as a function of memory using NS3-simulated trace following the flow size distribution in the Hadoop dataset. Each marker corresponds with one heavy hitter; we show the percentage error ($|\text{rank}(\widehat{\mathcal{L}_{x,q}}) - q|$)) for each x that satisfies $f_x \geq N\theta$ with a fixed value of $\theta = 0.01$, as a function of memory consumed. We examine the quantiles $q \in \{0.5, 0.9, 0.99\}$ of SQUAD.



(a) SQUAD Percentage Err vs. θ **(b)** SSGK **(c)** SQUAD

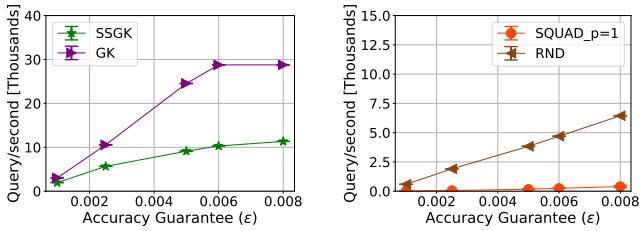
Fig. 4. (a) SQUAD Percentage error as a function of θ with fixed $\varepsilon = 0.025$ and $q = q = 0.9$ using NS3-simulated trace following the flow size distribution in the Hadoop dataset. Each marker represents one heavy hitter (b) SSGK update runtime as a function of ε with fixed $\theta == 0.01$ compared to its building blocks: SS and GK using NS3-simulated online search trace (c) SQUAD update runtime as a function of ε with fixed $\theta = 0.01$ compared to its building blocks: SS and Random (RND in the graph) using NS3-simulated online search trace. $SQUAD\_\mathbf{p} = 1$ indicates that the implementation excludes the optimizations of Section 7.



(a) All algorithms **(b)** Filtering

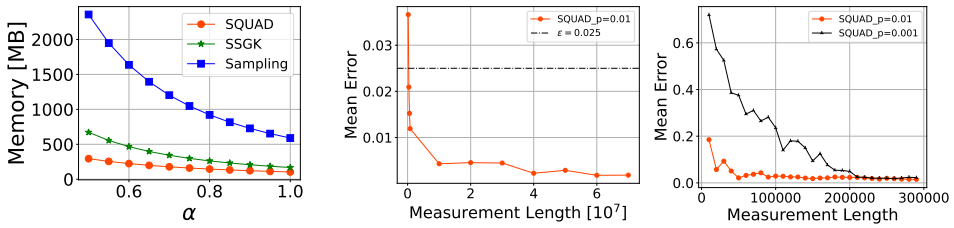
Fig. 5. Update runtime as function of the accuracy guarantee (ε) with fixed $\theta = 0.01$ using NS3-simulated online search trace (a) Comparing all three of our algorithms: Sampling, SSGK and SQUAD (b) The effect of the optimization on the performance of SQUAD update.

insensitive to changing ε values since its building blocks are insensitive to ε, and it is comparable to that of the GK algorithm. Recall that the SSGK update operation is equivalent to an update operation in SS and an update operation on the corresponding GK instance. Due to the high effectiveness of SS updates, the run time of SSGK updates is limited by the run time of GK. That is, while GK-algorithm



(a) SSGK **(b)** SQUAD

Fig. 6. Query runtime vs. accuracy guarantee (ε) with fixed $\theta = 0.01$ when asked for the $q = 0.9$ quantile using NS3-simulated online search trace (a) SSGK query performance compared to GK (b) SQUAD query performance compared to Random (RND).



(a) Memory **(b)** Error **(c)** Zoomed Error

Fig. 7. (a) Memory consumption as function of α with $\varepsilon = 0.025$ and $\theta = 0.01$ (b) Accuracy as a function of the measurement length till the stream's end when sampling with $\mathfrak{p} = 0.01$ in SQUAD with $q = 0.9$, $\theta = 0.01$, and $\varepsilon = 0.025$ using an NS3-simulated Hadoop trace. (c) Zoomed Mean Accuracy as a function of the measurement length size (N) using an NS3-simulated trace that follows the Hadoop flow size distribution. Each marker represents the mean of the heavy hitters' percentage error, with fixed values of $q = 0.9$, $\theta = 0.01$ and $\varepsilon = 0.025$. We examine the sampling probability $\mathfrak{p} = 0.01, 0.001$ in SQUAD.

solves the quantile problem for the full stream, the SSGK algorithm solves per-element quantiles without adding any extra update time cost.

Specifically, we may replace the GK instances in SSGK with any sketch that solves quantiles, such as Random [38], which has a higher update speed, as seen in Figure 4c. However, SSGK will no longer be a deterministic solution in this case. As a result, there is a trade-off between update speed and determinism. Additionally, it was shown that the GK-algorithm is an optimal **deterministic** comparison based algorithm.

8.3.2 SQUAD Update Time: Figure 4c compares SQUAD's update speed to that of its building blocks: Space Saving (SS) [43] and the Random algorithms [38].

Recall that the Random algorithm reports quantiles over the entire stream, thus it does not solve the $(\theta, \varepsilon, \delta)$-HH-quantiles and only serves as a best case reference point. SQUAD update operation is translated to sampling operation, SS update and Random update. As a result, the run time of its update is impacted by the run time of all of them. $SQUAD\_\mathfrak{p} = 1$ in the graph indicates that the implementation excludes the optimizations detailed in Section 7.

8.3.3 Comparing Sampling, SSGK and SQUAD Update Time: As seen in Figure 5a, Sampling is the fastest algorithm since each update is converted to a sampling update. Recall that every update operation in SSGK means SS and GK updates, while every update operation in SQUAD means sampling, SS and Random updates. Thus, SQUAD is better than SSGK in terms of update

performance since it is based on the Random algorithm, which has a faster update time than the GK-algorithm, the building block of SSGK, and the sampling update is extremely efficient. While Sampling is the quickest, it solves the $(\theta, \varepsilon, \delta)$-HH-quantiles with a high memory cost, as seen in Figure 2 and Table 1. When ε is small, Sampling keeps a significant number of samples more than the stream size. In this scenario, Sampling just saves all streams, which results in superior performance than larger values of ε, which allows items to override earlier samples.

Figure 5b shows how the filtering optimization discussed in Section 7 improves the update performance of SQUAD. As $\mathfrak{p}$ decreases, more arrivals from the stream are disregarded in an update operation, thus the update operations get faster. In this scenario, when the filter sampling probability $\mathfrak{p}$ is 0.1, its performance comparable or better than Sampling, and with $\mathfrak{p} = 0.01$, SQUAD becomes the clear winner. We explore these optimizations further below.

8.3.4 Query Speed Comparison: We used the quantiles $q = 0.5, 0.9, 0.99$ for comparing the query speed. We investigated the effect of the ε parameter using a fixed θ of 0.01 using NS3-simulated online search trace, and the experiment includes quantile queries for items x that satisfy the condition $f_x \geq N\theta$. For decreasing ε values, more values access the quantile sketches. Consequently, we got slower query operations in all algorithm. As seen in Figure 6, SSGK performs better than SQUAD because SQUAD relies on Random queries, which perform worse than the GK. Additionally, SQUAD checks its samples part to figure out the samples that were taken before the time the given identifier enters the SS. This becomes extremely expensive when the sample size is large, i.e. with small ε.

Particularly, as seen in Figure 4c, we may replace the Random instances in SQUAD with a GK sketch that has a faster query performance. However, as seen in Figure 4b, GK is slower in update performance. Indeed, there is a trade-off between update and query performance. However, since in most streaming applications updates occur more often than query operations, Random would usually be the preferred choice.

8.4 Optimizations Comparison

In this section we evaluate the optimizations described in Section 7. We examine the influence of the optimizations on the update runtime, memory usage, and empirical error.

8.4.1 Effect of Optimization on SQUAD Update Time: We implement the optimizations in SQUAD since it is the most space-efficient algorithm. The update runtime is shown in Figure 5b as a function of ε with a fixed $\theta = 0.01$. The three SQUAD implementations differ in the probability of the wrapper calling the Insert function of SQUAD. We consider three probabilities: $\mathfrak{p} = 1$ (indicating that the implementation does not include optimizations), $\mathfrak{p} = 0.1$, and $\mathfrak{p} = 0.01$. That is, each element in $\mathcal{S}$ occurs with probability $\mathfrak{p}$ in the sampled stream i.i.d. As expected, decreasing the value of $\mathfrak{p}$ results in improved update speed, as the algorithm invokes the SQUAD Insert function infrequently. As can be observed, the optimizations considerably improve the speed of the update.

8.4.2 Effect of α on Memory Consumption: Figure 7a shows the space consumed by our algorithms as function of α with fixed values $\varepsilon = 0.025$ and $\theta = 0.01$. As seen in Figure 7a, the greater α is, the less space is required for the algorithm, but the sampling probability must be increased. With $\alpha = 0.9$, $\varepsilon = 0.025$, and $\theta = 0.01$, the space needed for SQUAD increases by 18% compared to $\alpha = 1$. Since a greater α value necessitates a higher sampling probability ($\mathfrak{p}$), there is a trade-off between update speed and required space. That is, when the value of α decreases, so does the sampling probability, resulting in faster update operations but higher space consumption. The parameter α has a smaller effect on the memory of SQUAD than it does on Sampling and SSGK, since SQUAD has a better space complexity than the others for the same values of ε and θ.

8.4.3 The Effect of the Length of the Measurement on the Error: We study the trade-off between geo-sampling rate $\mathfrak{p}$ and the convergence time (in terms of the number of packets) and report the results in Figures 7b and 7c. We use an NS3-simulated trace that follows the Hadoop flow size distribution with fixed values of $q = 0.9$, $\theta = 0.01$, and $\varepsilon = 0.025$. Each marker represents the mean of the heavy hitters' percentage error. Since SQUAD with optimizations uses sampling to select packets, it requires a convergence time to produce a guaranteed accurate result (analyzed in Section 7). We examine the mean error of SQUAD during the whole trace and show it in Figure 7b. As the trace gets longer, the mean error drops and converges to an error less than the theoretical error. In Figure 7c, we examine the effect of two sampling probabilities $\mathfrak{p} = 0.01, 0.001$ on convergence time. As expected, larger $\mathfrak{p}$ value leads to faster convergence time as we sample elements in higher probability.

9 EXTENSIONS

9.1 Supporting Quantiles for Traffic Volume Heavy-Hitters

It is often desirable to find the quantiles for heavy hitters in terms of traffic volume. That is, consider a stream in which each element has a *size* and our goal is to find the quantile for items that use the majority of the bandwidth. Formally, we look at a *weighted stream* $\mathcal{S} = \langle (w_1, x_1, v_1), (w_2, x_2, v_2) \ldots \rangle \in (\{1, 2, \ldots, Z\} \times \mathcal{U} \times \mathbb{R})^+$ and define item's volume as the sum of sizes for elements that belong to it. The total weight of all elements is denoted $\mathrm{W} \triangleq \sum_{(x_i, v_i, w_i) \in \mathcal{S}} w_i$.

We say that an algorithm $\mathcal{A}$ solves $(\theta, \varepsilon, \delta)$-*Weighted*-HH-quantiles if every Query(x, q) returns a tuple $(\widehat{f_x}, \widehat{\mathcal{L}_{x,q}})$ such that $\Pr[|f_x - \widehat{f_x}| > \mathrm{W} \cdot \varepsilon] \leq \delta$ and if $f_x \geq \mathrm{W} \cdot \theta$, then $\Pr[|\mathsf{rank}(\widehat{\mathcal{L}_{x,q}}) - q|) > \varepsilon] \leq \delta$. Notice that for an unweighted stream (all weights are set to 1), the problem degenerates to the $(\theta, \varepsilon, \delta)$-HH-quantiles problem. Here, the weight w_i refers to the heavy-hitter definition but quantiles are unweighted (below we also consider a variant with weighted quantiles).

We augment SSGK and SQUAD to solve the $(\theta, \varepsilon, \delta)$-*Weighted*-HH-quantiles. These algorithms are termed w-SSGK and w-SQUAD.

To address $(\theta, \varepsilon, \delta)$-*Weighted*-HH-quantiles, we must estimate each element's total traffic volume, identify the weighted heavy hitters, and track the quantiles for those items. For SSGK, we simply replace the Space Saving of SSGK with a Space Saving for weighted elements (that updates in $O(\log \varepsilon^{-1})$ time), or one of its constant-time variants [6, 11, 12]. This ensures that a weighted heavy hitter has an entry and a quantile sketch that tracks its values.

Augmenting SQUAD is more complex. As sampling is indifferent to elements' weights, large-weight elements may be missed, resulting in a large error. Further, the weight applies to the item frequencies, so the sample, which is used to capture the behavior of an item's values before it was assigned an SS entry, should be kept as in SQUAD. Thus, we need to modify SQUAD frequency estimation since it is based on both the sample and the Space Saving. To that end, we propose two solutions. The first is to allocate additional weighted Space Saving (without quantile sketches) whose space complexity is $O(\frac{1}{\varepsilon})$, in order to estimate the frequency of weighted elements, while the original Space Saving remains unchanged. In this scenario, a w-SQUAD Insert(x, v, w) operation is executed as SQUAD Insert(x, v) with an additional update of the weighted Space Saving by w. The second option is to use the existing Space Saving in SQUAD and extend it to max $\left\{\varepsilon^{-1}, 4\varepsilon^{-0.5}\theta^{-1}\right\}$ entries, but we only maintain the quantile sketch for the top $m = 4\varepsilon^{-0.5}\theta^{-1}$ entries. In this case, w-SQUAD Insert(x, v, w) updates the extended weighted Space Saving with weight w: if x has an entry from the first m entries, we update the quantile sketch similarly to SQUAD; otherwise, if x has an entry j from the added entries ($j > m$), we only update the weighted Space Saving counter because this type of entry is only allocated with counters (with no quantile sketch).

Theorem 4. *w-SSGK and w-SQUAD solve $(\theta, \varepsilon, \delta)$-Weighted-HH-quantiles for weighted streams while requiring same space complexity as SSGK and SQUAD respectively.*

Proof Sketch. The correctness follows from the fact that Space Saving (and its faster variants) for weighted items guarantee an error of $W\varepsilon$ with the same space complexity as the unweighted version of Space Saving where W is the total weights of all elements in the stream. For w-SQUAD, the additional space complexity is bounded by SQUAD space complexity in both cases of additional weighted Space Saving or extension of the existing Space Saving. Thus, our algorithms are capable of solving the $(\theta, \varepsilon, \delta)$-*Weighted*-HH-quantiles problem with the same space complexity as the unweighted version, since their promised error is at most $W\varepsilon$. □

Another interesting variant is when the weight applies to both the identifier and the value quantiles. We call this problem $(\theta, \varepsilon, \delta)$-*Weighted*^{2}-HH-quantiles. Such a scenario may arise if the observed values of an identifier have different importance. The correctness definition is similar to $(\theta, \varepsilon, \delta)$-*Weighted*-HH-quantiles, but now the `rank` is weighted (i.e., between $q - \varepsilon$ and $q + \varepsilon$ of the weight should be smaller than the output value, except with probability δ).

We now expand w-SSGK and w-SQUAD to handle weighted values and denote them by w^2-SSGK and w^2-SQUAD respectively. In w-SSGK, we replace the quantile sketches of the weighted Space Saving with quantile sketches that support weighted items. KLL sketch [32] is a sketching algorithm that returns an approximation of a q-quantile over the entire stream. KLL can also handle weighted items due to its design, which includes a hierarchy of compactors and weighted items. Its size remains $O(\varepsilon^{-1}\sqrt{\log \varepsilon^{-1}})$, but the update time becomes $O(\log \varepsilon^{-1})$, as demonstrated by [31]. That is, each GK sketch in w-SSGK is substituted with the weighted version of KLL sketch [31]. Note that this makes w^2-SSGK randomized.

The building blocks of w-SQUAD are sampling, weighted Space Saving, and the quantile sketch. Since the weight now applies to quantiles as well, we need to modify both the sampling and the quantile sketch. As in the case of w^2-SSGK, the quantile sketch is substituted by the weighted version of KLL. We also replace the reservoir sampling with a weighted sampling procedure with replacement (e.g., see [23]). Intuitively, one can view this as breaking down each update (x, v_i, w_i) into w_i updates of (x_i, v_i) and feeding them into the reservoir sampling, but optimized for runtime.

Theorem 5. *w^2-SSGK and w^2-SQUAD solve $(\theta, \varepsilon, \delta)$-Weighted^2-HH-quantiles for weighted streams while requiring the same space complexity as SSGK and SQUAD respectively.*

Proof Sketch. The correctness comes from Theorem 4 and the fact that the proposed weighted version of KLL (Algorithm 5 and Theorem 4.3 in [31]) ensures ε-approximate quantiles using $O(\varepsilon^{-1}\sqrt{\log \varepsilon^{-1}})$ space, which is constrained by GK and Random space complexity. Furthermore, the reduction we employ in the sampling procedure preserves the error guarantee and the space complexity. It is worth mentioning that both the weighted KLL and the sampling impact the update performance. □

9.2 Tracking all Γ-sized items

In this section, we present a variant of SQUAD that allows it to provide reliable estimates to all items that appear at least Γ (e.g., for $\Gamma = 1000$). Intuitively, if the stream length N is not known in advance (otherwise, we could simply set $\theta = \Gamma/N$), we need to change the algorithm to support this new definition. For heavy hitters, as an example, while one could use SS with N/Γ counters if N is known apriori, to the best to our knowledge no standard solution is applicable for the case that N unknown beforehand (thus the proposed method may be of independent interest).

Instead, we modify the algorithm as follows: First, the Reservoir Sampling component is replaced with a simple sampling procedure that adds each incoming packet with probability $\Theta\left(\frac{\varepsilon^{-1.5}\log\delta^{-1}}{\Gamma}\right)$. Next, every item that is sampled at least $\Theta\left(\varepsilon^{-1}\log\delta^{-1}\right)$ times is allocated (independently of other items) with an exact counter and a quantile sketch (e.g., Random) configured for error $\varepsilon/2$ except with probability $\delta/2$. We now summarize the theoretical guarantees:

Theorem 6. *The above provides an ε-approximation for each ID whose frequency is at least Γ while requiring $\widetilde{O}\left(\frac{N}{\Gamma}\cdot\varepsilon^{-1.5}\right)$ space.*

Proof sketch. Intuitively, we ensure that any item of size $\Gamma\sqrt{\varepsilon}$ is sampled at least $\Omega(\varepsilon^{-1}\log\delta^{-1})$ times except with probability $\delta/4$. A similar analysis to the previous sections show that the samples yield $\Theta(\sqrt{\varepsilon})$-error quantile and frequency estimates except with further probability $\delta/4$ and thus the rank estimate is off by at most $(\Gamma\sqrt{\varepsilon})\cdot\Theta(\sqrt{\varepsilon})=\Theta(\Gamma\varepsilon)$. This ensures that the quantile error for items larger than Γ is at most ε. For the memory analysis, notice that with high probability at most $O(\frac{N}{\Gamma\sqrt{\varepsilon}})$ items will be allocated with sketches, and thus the overall space is at most $\widetilde{O}\left(\frac{N}{\Gamma}\cdot\varepsilon^{-1.5}\right)$. □

We further note that our filtering optimization (Section 7) is applicable to this variant as well and that any algorithm for this problem variant must use at least $\widetilde{\Omega}(\frac{N}{\Gamma}\cdot\varepsilon^{-1})$ space as there can be $\frac{N}{\Gamma}$ items of size Γ and any quantile sketch must use $\widetilde{\Omega}(\varepsilon^{-1})$ space.

10 DISCUSSION

In this paper, we studied the problem of reporting the quantiles of heavy hitter items. To our knowledge, this is the first research to solve quantiles on a per-item level rather than reporting quantiles of an entire stream. Such capabilities can be useful when one wishes to assess a network's health and to debug various networking middle-boxes and smart data-planes as well as multi-tenant clouds.

We presented a formal definition of the generalized problem and explored different solutions: a sampling approach, a sketching approach (SSGK), and a sampling-sketching combined approach (called SQUAD). SSGK is a deterministic solution that assigns a unique quantile-sketch (GK) to each potential heavy hitter that is obtained from a Space Saving instance. SQUAD combines sampling with SSGK, resulting in superior memory reduction.

SQUAD is the most memory-efficient algorithm. Both SQUAD and the sampling algorithm use about the same amount of memory. On the other hand, SSGK is deterministic, but the sampling has an error probability. This is true both asymptotically and empirically in a large-scale NS3 simulation, where we observed orders of magnitude memory reductions for similar estimation errors in SQUAD.

While the sampling algorithm's update rate is higher than SSGK and SQUAD, it consumes a lot of memory. To reach high speeds with SQUAD, we suggested several efficiency enhancements for our algorithms' update operation in Section 7. In fact, the update performance of SSGK is comparable to that of the state-of-the-art method that can only handle quantiles throughout the whole stream, not per-item quantiles. Our approach can be applied to both volume traffic, where each element in the stream has a size, and items that appear at least Γ times in the stream.

Code Availability: All code is available online [2].

Acknowledgements: We thank Sivaram Ramanathan for his advice on designing the NS3 simulations. This research was partially funded by ISF grant #3119/21 and Technion HPI.

REFERENCES

[1] Handling elephant flow on a dpdk-based load balancer - https://dpdksummitapac2021.sched.com/event/hdlm/handling-elephant-flow-on-a-dpdk-based-load-balancer.

[2] Open source code. https://anonymous.4open.science/r/squad-3BB9.

[3] The Network Simulator ns-3. https://www.nsnam.org/research/wns3/wns3-2015/.

[4] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):1–28, 2013.

[5] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[6] Daniel Anderson, Pryce Bevan, Kevin Lang, Edo Liberty, Lee Rhodes, and Justin Thaler. A high-performance algorithm for identifying frequent items in data streams. In *Proceedings of the 2017 Internet Measurement Conference*, pages 268–282. ACM, 2017.

[7] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 286–296, 2004.

[8] Ran Ben Basat, Gil Einziger, Shir Landau Feibish, Jalil Moraney, and Danny Raz. Network-wide routing-oblivious heavy hitters. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, pages 66–73, 2018.

[9] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Volumetric hierarchical heavy hitters. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 381–392. IEEE, 2018.

[10] Ran Ben Basat, Gil Einziger, Marcelo Caggiani Luizelli, and Erez Waisbard. A black-box method for accelerating measurement algorithms with accuracy guarantees. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.

[11] Ran Ben Basat, Gil Einziger, and Roy Friedman. Space efficient elephant flow detection. In *Proceedings of the 11th ACM International Systems and Storage Conference*, pages 115–115, 2018.

[12] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Optimal elephant flow detection. In *IEEE INFOCOM*, 2017.

[13] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo C Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 127–140, 2017.

[14] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. Pint: Probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 662–680, 2020.

[15] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems*, 31(3), aug 2013.

[16] Graham Cormode, Minos Garofalakis, Shanmugavelayutham Muthukrishnan, and Rajeev Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 25–36, 2005.

[17] Graham Cormode and Marios Hadjieleftheriou. Methods for Finding Frequent Items in Data Streams. *J. VLDB*, 19(1):3–20, 2010.

[18] Graham Cormode, Zohar Karnin, Edo Liberty, Justin Thaler, and Pavel Veselỳ. Relative error streaming quantiles. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 96–108, 2021.

[19] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 155–166, 2004.

[20] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 263–272, 2006.

[21] Graham Cormode and Pavel Veselỳ. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 81–93, 2020.

[22] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-Value

Store. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*, SOSP, page 205–220, 2007.
[23] Pavlos S Efraimidis. Weighted random sampling over data streams. In *Algorithms, Probability, Networks, and Games*, pages 183–195. Springer, 2015.
[24] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a Better NetFlow. In *ACM SIGCOMM*, 2004.
[25] David Felber and Rafail Ostrovsky. A randomized online quantile summary in $o(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$ words. *Theory of Computing*, 13(1):1–17, 2017.
[26] Naga K Govindaraju, Nikunj Raghuvanshi, and Dinesh Manocha. Fast and approximate stream mining of quantiles and frequencies using graphics processors. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 611–622, 2005.
[27] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.
[28] Michael B Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 275–285, 2004.
[29] Anupam Gupta and Francis Zane. Counting inversions in lists. In *SODA*, volume 3, pages 253–254, 2003.
[30] Zengfeng Huang, Lu Wang, Ke Yi, and Yunhao Liu. Sampling based algorithms for quantile computation in sensor networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 745–756, 2011.
[31] 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.
[32] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*, pages 71–78. IEEE, 2016.
[33] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Transactions Database Systems*, 2003.
[34] Jonatan Langlet, Ran Ben-Basat, Sivaramakrishnan Ramanathan, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. Zero-cpu collection with direct telemetry access. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, pages 108–115, 2021.
[35] Kim-Hung Li. Reservoir-sampling algorithms of time complexity o (n (1+ log (n/n))). *ACM Transactions on Mathematical Software (TOMS)*, 20(4):481–493, 1994.
[36] 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 *Proceedings of the ACM Special Interest Group on Data Communication*, pages 334–350. 2019.
[37] Kristina Bennett Liz Fong-Jones and Stephen Thorne. Developing Effective Service Level Indicators and Service Level Objectives. In *SRECon*, 2018.
[38] Ge Luo, Lu Wang, Ke Yi, and Graham Cormode. Quantiles over data streams: experimental comparisons, new analyses, and further improvements. *The VLDB Journal*, 25(4):449–472, 2016.
[39] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record*, 27(2):426–435, 1998.
[40] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *ACM SIGMOD Record*, 28(2):251–262, 1999.
[41] Andrew McGregor, A Pavan, Srikanta Tirthapura, and David P Woodruff. Space-efficient estimation of statistics over sub-sampled streams. *Algorithmica*, 74(2):787–811, 2016.
[42] Kathryn McKinley. Measuring and optimizing tail latency. In *Strange Loop Conference*, 2017. https://www.youtube.com/watch?v=_Zoa3xkzgFk.
[43] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, pages 398–412. Springer, 2005.
[44] Jayadev Misra and David Gries. Finding repeated elements. Technical report, 1982.
[45] J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.
[46] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 85–98, 2017.
[47] Bruno Ribeiro, Tao Ye, and Don Towsley. A resource-minimalist flow size histogram estimator. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 285–290, 2008.
[48] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137,

2015.

[49] Eric Schurman and Jake Brutlag. Performance related changes and their user impact. In *Velocity Web Performance and Operations Conference*, 2009. https://www.youtube.com/watch?v=bQSE51-gr2s.

[50] Robert J Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, pages 39–48, 1974.

[51] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.

[52] The P4.org Applications Working Group. In-band Network Telemetry (INT) Dataplane Specification. https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report.pdf.

[53] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015. First published in 1971.

[54] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[55] Ke Yi and Qin Zhang. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 65(1):206–223, 2013.

[56] Qi Zhang and Wei Wang. An efficient algorithm for approximate biased quantile computation in data streams. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 1023–1026, 2007.

[57] Ying Zhang, Xuemin Lin, Jian Xu, Flip Korn, and Wei Wang. Space-efficient relative error order sketch over data streams. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 51–51. IEEE, 2006.

Received July 2022; revised October 2022; accepted November 2022