Weighted Reservoir Sampling from Distributed Streams

Rajesh Jayaram
Carnegie Mellon University
rkjayara@cs.cmu.edu

Gokarna Sharma
Kent State University
gsharma2@kent.edu

Srikanta Tirthapura
Iowa State University
snt@iastate.edu

David P. Woodruff
Carnegie Mellon University
dwoodruf@cs.cmu.edu

ABSTRACT

We consider message-efficient continuous random sampling from a distributed stream, where the probability of inclusion of an item in the sample is proportional to a weight associated with the item. The unweighted version, where all weights are equal, is well studied, and admits tight upper and lower bounds on message complexity. For weighted sampling with replacement, there is a simple reduction to unweighted sampling with replacement. However, in many applications the stream may have only a few heavy items which may dominate a random sample when chosen with replacement. Weighted sampling *without replacement* (weighted SWOR) eludes this issue, since such heavy items can be sampled at most once.

In this work, we present the first message-optimal algorithm for weighted SWOR from a distributed stream. Our algorithm also has optimal space and time complexity. As an application of our algorithm for weighted SWOR, we derive the first distributed streaming algorithms for tracking *heavy hitters with residual error*. Here the goal is to identify stream items that contribute significantly to the residual stream, once the heaviest items are removed. Residual heavy hitters generalize the notion of ℓ_1 heavy hitters and are important in streams that have a skewed distribution of weights. In addition to the upper bound, we also provide a lower bound on the message complexity that is nearly tight up to a $\log(1/\epsilon)$ factor. Finally, we use our weighted sampling algorithm to improve the message complexity of distributed L_1 tracking, also known as count tracking, which is a widely studied problem in distributed streaming. We also derive a tight message lower bound, which closes the message complexity of this fundamental problem.

KEYWORDS

Random Sampling, Continuous Streams, Weighted Sampling, Heavy Hitters, L_1 Tracking

ACM Reference Format:
Rajesh Jayaram, Gokarna Sharma, Srikanta Tirthapura, and David P. Woodruff. 2019. Weighted Reservoir Sampling from Distributed Streams. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3294052.3319696

1 INTRODUCTION

We consider the fundamental problem of maintaining a random sample of a data stream that is partitioned into multiple physically distributed streams. In the streaming setting, it is often observed that many distributed monitoring tasks can be reduced to sampling. For instance, a search engine that uses multiple distributed servers can maintain the set of "typical" queries posed to it through continuously maintaining a random sample of all the queries seen thus far. Another application is network monitoring, one of the driving applications for data stream processing, which deploys multiple monitoring devices within a network. Each device receives extremely high rate data streams, and one of the most commonly desired aggregates is a random sample over all data received so far [15, 16]. Oftentimes, the predominant bottleneck in distributed data processing is the network bandwidth, and so it is highly desirable to have algorithms that communicate as few messages as possible. In particular, as the volume of the data and the number of distributed sites observing the streams scale, it becomes infeasible to communicate all of the data points observed to a central coordinator. Thus, it is necessary to develop algorithms which send significantly fewer messages than the total number of data points received.

PODS'19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6227-6/19/06…$15.00
https://doi.org/10.1145/3294052.3319696

The above applications have motivated the study of the *continuous, distributed, streaming model [14]*, where there are k physically distributed sites, numbered 1 through k. Each site i receives a local stream of data $\mathcal{S}_i$. The sites are connected to a central coordinator, to which they can send and recieve messages. Queries are posed to the coordinator, asking for an aggregate over $\mathcal{S} = \cup_{i=1}^{k} \mathcal{S}_i$, the union of all streams observed so far. The goal is to minimize the *message complexity*, i.e., number of messages sent over the network, over the entire observation.

We now define the **distributed weighted random sampling** problem. Each local stream $\mathcal{S}_i$ consists of a sequence of items of the form (e, w) where e is an item identifier and w is a positive weight. The streams $\mathcal{S}_i$ are assumed to be disjoint. Let n denote the size of $\mathcal{S} = \cup_{i=1}^{k} \mathcal{S}_i$. The task is for the coordinator to continuously maintain a weighted random sample of size $s = \min\{n, s\}$ from $\mathcal{S}$. Note that the same identifier e can occur multiple times, perhaps in different streams with different weights, and each such occurrence is to be sampled as if it were a different item. We consider two variations of weighted random sampling – sampling with replacement and sampling without replacement.

A single weighted random sample from $\mathcal{S}$ is defined to be an item chosen from $\mathcal{S}$ where *the probability of choosing item* (e, w) *is proportional to* w*, i.e., equal to* $\frac{w}{\sum_{(e',w')\in\mathcal{S}} w'}$.

Definition 1.1. A *weighted random sample without replacement* (weighted SWOR) from $\mathcal{S}$ is a set S generated according to the following process. Initially S is empty. For i from 1 to s, a single weighted random sample is chosen from $(\mathcal{S} \setminus S)$ and added to S.

Definition 1.2. A *weighted random sample with replacement* (weighted SWR) from $\mathcal{S}$ is a set S generated according to the following process. Initially S is empty. For i from 1 to s, a single weighted random sample is chosen from $\mathcal{S}$ and added to S.

Definition 1.3. A distributed streaming algorithm $\mathcal{P}$ is a weighted sampler without (with) replacement if for each $t > 0$, the coordinator maintains a set S of size $\min\{t, s\}$ such that S is a weighted random sample chosen without (with) replacement from all items seen so far, $\{(e_1, w_1), \ldots, (e_t, w_t)\}$.

Distributed random sampling generalizes the classic *reservoir sampling* problem [28, 33] from a centralized to a distributed setting. Random sampling serves as a building block in other aggregation tasks, such as estimation of the number of distinct elements [12, 13] and identifying heavy hitters [5, 26, 29, 36]. Distributed random sampling has also been used in approximate query processing in big data systems such as BlinkDB [1]. Unweighted distributed sampling, where all weights are equal, is well studied and admits tight upper and lower bounds on message complexity [10, 14, 31]. However, to the best of the authors' knowledge, weighted distributed sampling has not been studied so far, though there is prior work in the centralized setting [7, 18].

Designing a message-efficient distributed sampler is nontrivial. One challenge is that the system state is distributed across multiple sites. It is not possible to keep the distributed state tightly synchronized, since this requires a significant message overhead. Since the states of the sites are not synchronized with the coordinator, sites may send updates to the coordinator even though the sample at the coordinator does not change. A second challenge is that the traditional metrics for a centralized sampling algorithm, such as space complexity and time per item, do not affect the message complexity of a distributed algorithm. For instance, a centralized algorithm that uses $O(1)$ update time and optimal $O(s)$ space need not lead to a distributed algorithm with optimal messages, since the number of messages sent depends on how many times the random sample changes according to the views of the sites and the coordinator. Finally, we emphasize the fact that, in Definition 1.3, the protocol must maintain a weighted SWOR *at all times* in the stream. There is no notion of failure in the definition of a weighted sampler – the protocol can never fail to maintain the sample S. These two features make the problem substantially more challenging.

1.1 Contributions

Let k denote the number of sites, s the desired sample size, and W the total weight received across all sites.

– We present an algorithm for weighted SWOR that achieves an optimal expected message complexity of $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$. The algorithm uses an optimal $\Theta(1)$ space at each site, and an optimal $\Theta(s)$ space at the coordinator. The update time of our algorithm is also optimal: $\Theta(1)$ for a site to process an update, and $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ total runtime at the coordinator. Our algorithm is the first message-optimal algorithm for distributed weighted SWOR. We note that a message-efficient algorithm for weighted SWR follows via a reduction to the unweighted case, and obtaining an algorithm for SWOR is significantly harder, as described below.

– As an application of our weighted SWOR, we provide the first distributed streaming algorithm with small message complexity for continuously monitoring heavy hitters with a *residual* error guarantee. This allows us to identify heavy hitters in the residual stream after extremely heavy elements are removed. A residual error guarantee is stronger than the guarantee provided by ℓ_1 heavy hitters, and is especially useful for streams where the weight of items is highly skewed [6]. The expected message complexity of our algorithm is $O(k\log(W)/\log(k)+\log(1/\epsilon)\log(W)/\epsilon)$. We

prove that our algorithm is nearly optimal, by also giving a $\Omega(k \log(W)/\log(k) + \log(W)/\epsilon)$ lower bound, which is tight up to a $\log(1/\epsilon)$ factor in the second term.

– We demonstrate another application of our sampling algorithms to the well-studied problem of L_1 tracking (or count tracking) in distributed streams, which requires the coordinator to maintain a $(1 \pm \epsilon)$ relative error approximation of the total weight seen so that at any given time, with constant probability, the estimate is correct.

For the case $k \geq 1/\epsilon^2$, the best known upper bound was $O(k \log W)$ messages in expectation [23]. Our algorithm for L_1 tracking uses $O(k \log(W)/\log(k) + \log(\epsilon W)/\epsilon^2)$ messages in expectation, which improves on the best known upper bound when $k \geq 1/\epsilon^2$. In this setting, we also improve the lower bound from $\Omega(k)$ to $\Omega(k \frac{\log(W)}{\log(k)})$.

For the case $k \leq 1/\epsilon^2$, matching upper and lower bounds of $\Theta((\sqrt{k}/\epsilon) \log W)$ were known [23]. When $k \geq 1/\epsilon^2$, the lower bound of [23] becomes $\Omega(\log(W)/\epsilon^2)$. So if $k \geq 1/\epsilon^2$ and $k \frac{\log(W)}{\log(k)} < \log(W)/\epsilon^2$, our upper bound is $O(\log(W)/\epsilon^2)$, which is tight, and otherwise our upper bound is $O(k \log(W)/\log(k))$, which is also tight. Thus, combined with the upper and lower bounds of [23], our results close the complexity of the distributed L_1 tracking problem.

1.2 Our Techniques

For the problem of sampling *with* replacement, there is a relatively straightforward reduction from the weighted to the unweighted case, which we elaborate on in Section 2.2. The reduction involves duplicating a item (e, w) a total of w times into *unweighted* updates, and does not require an increase in message complexity. On the other hand, there are inherent difficulties in attempting to carry out a similar reduction for sampling *without* replacement, which we will now discuss.

On the Difficulty of a Reduction from Weighted SWOR to Unweighted SWOR: We now examine the difficulties which arise when attempting to reduce the problem of weighted SWOR to unweighted SWOR. We consider the following natural candidate reduction. Given a weighted stream of items $\mathcal{S} = \{(e_i, w_i) | i = 1 \ldots n\}$, consider an unweighted stream $\mathcal{S}'$ where for each $(e_i, w_i) \in \mathcal{S}$, there are w_i copies of e_i in $\mathcal{S}'$. Note that $\mathcal{S}'$ can be constructed in a streaming manner as items of $\mathcal{S}$ are seen.

Let S' be an unweighted SWOR of s items from $\mathcal{S}'$. We remark that *if S' consists of s distinct identifiers*, then those distinct identifiers are in fact a weighted SWOR of $\mathcal{S}$. The difficulty of using this reduction is that of ensuring s distinct items within S'. One could consider a method that maintains an unweighted SWOR of size greater than s in S', expecting to get at least s distinct identifiers among them. However, this is not straightforward either, due to the presence of heavy-hitters (items with very large weight) which may contribute to a large fraction of the total weight in $\mathcal{S}$. For instance, if there are $s/2$ items that contribute to a more than $1 - 1/(100s)$ fraction of the total weight within $\mathcal{S}$, then S' is likely to contain only identifiers corresponding to these items. This makes it very unlikely that S' has s distinct identifiers, even if the size of S' is much larger than s. If the number of distinct items in S' falls below s, then one could invoke a "recovery" procedure that samples further items from the stream to bring the sample size back to s, but this itself will be a non-trivial distributed algorithm. Moreover, re-initializing or recovering the algorithm would be costly in terms of message complexity, and introduce unintended conditional dependencies into the distribution of the output. Note that one cannot apply distinct sampling [20, 21] to maintain s distinct items from $\mathcal{S}'$, since distinct sampling completely ignores the frequency of identifiers in $\mathcal{S}'$ (which correspond to weights in $\mathcal{S}$), while we do not want this behavior – items with a greater weight should be chosen with a higher probability.

Thus, one of the primary difficulties with an algorithm for weighted SWOR is to handle heavy hitters. One could next consider a method that explicitly maintains heavy hitters within the stream (using a streaming algorithm). On the remainder of the stream excluding the heavy hitters, one could attempt to apply the reduction to unweighted SWOR as described above. However, this does not quite work either, since after removing the heavy hitters from the stream, the remaining weight of the stream may still be dominated by just a few items, and the same difficulty persists. One has only swapped one set of heavy hitters for another. Such "residual heavy hitters" may not be heavy hitters in the original stream, and may have evaded capture when the first heavy hitters were identified. This may proceed recursively, where the weight of the stream is still dominated by only a few items after removing the heavy hitters and the residual heavy hitters. Therefore, heavy hitter identification does not solve our problem and further ideas are needed.

Algorithmic Ideas for Weighted SWOR: Our main algorithm for weighted SWOR combines two key ideas. Our first key technical contribution is to divide the items into multiple "level sets" according to the magnitude of their weights. All items within a single "level set" have weights that are close to each other. Our algorithm withholds an item from being considered by the sampler until the level set that the item belongs to has a (pre-specified) minimum number of items. This property ensures that when an item is considered for sampling, there are at least a minimum number of items of the same (or similar) weight, so that the difficulty that we faced with extreme heavy hitters does not surface. When a

level set reaches a certain size, all items within the level set are released for further sampling. By choosing the weights of different levels to increase geometrically, we ensure that the number of level sets remains small, and the overhead of additional messages for filling up the level sets is small. While an item is withheld, we also run a procedure so that, at every time step, it will still be output in the sample at that time with the correct probability. Thus, the notion of withholding an item applies means only that it is withheld from an internal sampling algorithm (a subroutine), and not the overall sampler. Note that, for a distributed sampler to be correct, it cannot entirely withhold an item from being sampled, even for a single time step.

The second idea is that of "precision sampling". Originally used in [2] for sampling in non-distributed data streams, and then extended by [24, 25], the idea of precision sampling is to scale each weight w_i by an i.i.d. random variable x_i, which generates a "key" $v_i = w_i x_i$ for each item (e_i, w_i). One then looks at the resulting vector $v = (v_1, v_2, \ldots, v_n)$ of keys, and returns the largest coordinates in v. In particular, [24] used the random variables $x_i = 1/t_i^{1/p}$ where t_i is *exponentially distributed*, to develop perfect L_p samplers. A similar idea is also used in "priority sampling" [17] which was developed in the context of network monitoring to estimate subset sums. We use precision sampling to generate these keys for each item, such that the items with the s largest keys form the weighted SWOR of the stream.

It is not difficult to see that if each site independently ran such a sampler on its input–storing the items with the s largest keys–and sent each new sample to the coordinator, who then stores the items with the overall s largest keys, one would have a correct protocol with $O(ks \log(W))$ expected communication. This could be carried out by generating the keys w_i/t_i with exponential variables t_i, or also by using the keys u_i^{1/w_i} with u_i uniform on $(0, 1)$, as used for non-distributed weighted sampling without replacement in [18]. Thus, a key challenge addressed by our work is to improve the naïve multiplicative bound of $\tilde{O}(ks)$ to an additive bound of $\tilde{O}(k + s)$.

We also remark that by duplicating weighted items *in combination* with our new level set technique, it may be possible to adapt the message-optimal *unweighted* sampling algorithms of [14, 31] (see also [32]) to yield a weighted SWOR. The approach would nonetheless require the use of level sets, along with a similar analysis as provided in this work, to remove extremely heavy items from the stream which would cause issues with these samplers. We believe that our approach by scaling an entire weight by a random variable, rather than manually duplicating weighted items (e, w) into w unweighted items, is more natural and simpler to understand. Moreover, it is likely that we obtain improved runtime bounds at the sites by using the algorithm presented in this paper (which are runtime optimal).

Residual Heavy Hitters: For the problem of monitoring heavy hitters, the technique of sampling *with replacement* has been frequently applied. By standard coupon collector arguments, taking $O(\log(1/\epsilon)/\epsilon)$ samples with replacement is enough to find all items which have weight within an ϵ fraction of the total. On the other hand, it is possible and frequently the case that there are very few items which contain nearly all the weight of the stream. This is precisely the domain where SWOR achieves remarkably better results, since a with replacement sampler would only ever see the heavy items. Using this same number of samples *without replacement*, we demonstrate that we can recover all items which have weight within an ϵ fraction of the total *after* the top $1/\epsilon$ largest items are removed. This is *residual error* guarantee is much stronger in the case of skewed distributions of weights, and is a important application of SWOR.

L_1 **Tracking:** Finally, we observe that the following desirable property of our weighted SWOR: namely that, once the heavy hitters are withheld, the values of the keys stored by the algorithm at any time provide good estimates of the total L_1 (the sum of all the weights seen so far). By taking enough samples, we can show that the s-th order statistic, that is the s-largest key overall, concentrates around the value W^t/s, up to a $(1 \pm \epsilon)$ factor, where W^t is the sum of all the weights up to a given time t. Unfortunately, withholding heavy items alone is not sufficient for good message complexity, as one must naively withhold an extra $\Theta(1/\epsilon)$ factor more heavy items than the size of s to obtain good concentration. To avoid a $\Theta(1/\epsilon)$ blow-up in the complexity, we must remove heavy hitters in another way. To do this, we duplicate updates instead of withholding them, a trick used in [24] for a similar sampler. This observation yields an optimal algorithm for distributed L_1 tracking for $k \geq 1/\epsilon^2$, by using our weighted SWOR as a subroutine. Previously an optimal algorithm for L_1 tracking was known only for $k < 1/\epsilon^2$.

1.3 Related Work

Random sampling from a stream is a fundamental problem and there has been substantial prior work on it. The reservoir sampling algorithm (attributed to Waterman [28]) has been known since the 1960s. There has been much follow-up work on reservoir sampling including methods for speeding up reservoir sampling [33], sampling over a sliding window [4, 8, 19, 22, 35], and sampling from distinct elements in data [20, 21].

The sequential version of weighted reservoir sampling was considered by Efraimidis and Spirakis [18], who presented a one-pass $O(s)$ algorithm for weighted SWOR. Braverman et

al. [7] presented another sequential algorithm for weighted SWOR, using a reduction to sampling with replacement through a "cascade sampling" algorithm.

Unweighted random sampling from distributed streams has been considered in prior works [10, 14, 31], which have yielded matching upper and lower bounds on sampling without replacement. Continuous random sampling for distinct elements from a data stream in a distributed setting has been considered in [11].

There has been a significant body of research on algorithms and lower bounds in the continuous distributed streaming model. This includes algorithms for frequency moments, [12, 13], entropy estimation [3, 9], heavy hitters and quantiles [36], distributed counts [23], and lower bounds on various statistical and graph aggregates [34].

Roadmap: We present preliminaries and basic results in Section 2, followed by an optimal algorithm for weighted SWOR in Section 3, applications to residual heavy hitters and lower bounds in Section 4, applications to L_1 tracking and lower bounds in Section 5, and concluding remarks in Section 6. Some proofs are deferred to Appendix.

2 PRELIMINARIES AND BASIC RESULTS

2.1 Preliminaries

As in prior work in the continuous distributed streaming model, we assume a *synchronous* communication model, where the system operates in *rounds*, and in each round, each site can observe (at most) one item, and send a message to the coordinator, and receive a response from the coordinator. We assume that the messages are delivered in FIFO order (no overtaking of messages), there is no message loss, and the sites and the coordinator do not crash. We make no assumption on the sizes of the different local streams received by different sites, the order of arrival, and the interleaving of the streams at different sites. The only assumption we have is a global ordering of the stream items by their time of arrival onto one of the sites. If n is the total number of items observed in the system, then for $j = 1 \dots n$, $o^j = (e_j, w_j)$ is the jth item observed in the total order. The partitioning of the items across different processors is carried out by an adversary.

Our space bounds are in terms of machine words, which we assume are of size $\Theta(\log(nW))$ bits, and that arithmetic operations on machine words can be performed in $O(1)$ time. We also assume that an element identifier and weight fits in a constant number of words. In our algorithms, the length of a message is a constant number of words, so that the number of messages is of the same order as the number of words communicated over the network. For simplicity (but without loss of generality), in the following sections we assume each weight w_j satisfies $w_j \geq 1$. Since each weight w_j can be written in a constant number of machine words by assumption, it follows that we could always scale the weights by a polynomial factor to ensure $w_j \geq 1$, which blows up the complexity of our algorithms only by a constant, since we only have logarithmic dependency on the total weight.

2.2 Basic Results

We first present some basic results for weighted SWR and weighted SWOR. Let n denote the number of items received in the stream.

Algorithm for Weighted SWR: We first derive a message-efficient algorithm for weighted SWR using a reduction to unweighted SWR. For this reduction, we assume that the weights w_i are integers. We first note the following result, from [14].

Theorem 2.1 ([14]). *There is a distributed algorithm for unweighted SWR with message complexity $O((k + s\log s)\frac{\log n}{\log(2+k/s)})$. The coordinator needs $O(s)$ space and $O((k + s\log s)\frac{\log n}{\log(2+k/s)})$ total time; each site needs $O(1)$ space and $O(1 + \frac{s}{n}\log s\frac{\log n}{\log(2+k/s)})$ time per item amortized.*

Corollary 2.2. *There is a distributed algorithm for weighted SWR with message complexity $O((k + s\log s)\frac{\log W}{\log(2+k/s)})$ where W is the total weight received so far. The coordinator needs $O(s)$ space and $O((k + s\log s)\frac{\log W}{\log(2+k/s)})$ total time. Each site needs $O(1)$ space and the amortized processing time per item at a site is $O(1 + \frac{1}{n}(k + s\log s)\frac{\log W}{\log(2+k/s)})$.*

Proof. We reduce weighted SWR to unweighted SWR as follows. Given stream of items with weights $\mathcal{S} = (e_i, w_i), i = 1 \dots n$, consider an unweighted stream $\mathcal{S}'$ where for each $(e_i, w_i) \in \mathcal{S}$, there are w_i copies of e_i in $\mathcal{S}'$. Note that $\mathcal{S}'$ can be constructed in a streaming manner as items of $\mathcal{S}$ are seen.

Note that an unweighted SWR of s items chosen from $\mathcal{S}'$ is a weighted SWR of s items chosen from $\mathcal{S}$. To prove this, let S' denote an unweighted SWR of size s from $\mathcal{S}'$. Consider an arbitrary item in S'. Since there are w_i copies of e_i in $\mathcal{S}'$, this item is likely to be e_i with probability $w_i/\sum_j w_j$, and hence obeys the distribution we desire of a single item in a weighted SWR of $\mathcal{S}$. Since the items of S' are chosen independent of each other, the entire sample S' has the same distribution as a weighted SWR of s items from $\mathcal{S}$. The number of items in $\mathcal{S}'$ is equal to W, the total weight of $\mathcal{S}$. The message complexity, as well as the space complexity at the sites and the coordinator follow from Theorem 2.1.

The processing time per item at the site needs additional work. An unweighted SWR sampler simulates the execution

of s independent copies of a single element sampler. Each element (e, w) in the weighted input stream leads to w elements into each of the s samplers. Naively done, this reduction leads to a total runtime of $O(sw)$, which can be very large. To improve on this, we speed up the sampling as follows. We note that in the algorithm in Section 3.2 of [14] for a single element unweighted SWR, in round j, an element is sent to the coordinator with probability 2^{-j} [1] When w elements are inserted, as in our reduction, the probability that at least one of them is sent to the coordinator is $\alpha(w, j) = 1 - (1 - 2^{-j})^w$. The site can simply send the element (e, w) to the coordinator with probability $\alpha(w, j)$. Repeating this s times leads to a runtime of $O(s)$ per element. To further improve on this, note that across all the s single element samplers, the number of samplers that send a message to the coordinator is a binomial random variable $B(s, \alpha(w, j))$. In the improved algorithm, the site samples a number X from this distribution. If $X > 0$, it chooses a random size X subset of the s samplers, and send (e, w) to the coordinator for each chosen sampler – as also noted in [14], this leads to the same distribution as making an independent decision for each sampler. The total time taken at the sites is now of the same order as the number of messages sent to the coordinator, which is $O\left((k + s \log s)\frac{\log W}{\log(2+k/s)}\right)$. The amortized time per element at the site is thus $O\left(1 + \frac{1}{n}(k + s \log s)\frac{\log W}{\log(2+k/s)}\right)$. □

Lower Bound for Weighted SWOR: We next present a lower bound on the message complexity of weighted SWOR, which follows from prior work on the message complexity of unweighted sampling without replacement. Let s denote the desired sample size and k the number of sites.

Theorem 2.3 ([31]). *For a constant $q, 0 < q < 1$, any correct algorithm that continuously maintains an unweighted random sample without replacement from a distributed stream must send $\Omega\left(\frac{k \log(n/s)}{\log(1+(k/s))}\right)$ messages with probability at least $1 - q$ and where the probability is taken over the algorithm's internal randomness, and where n is the number of items.*

Corollary 2.4. *For a constant $q, 0 < q < 1$, any correct algorithm that continuously maintains a weighted random sample without replacement from $\mathcal{S}$ must send $\Omega\left(\frac{k \log(W/s)}{\log(1+(k/s))}\right)$ messages with probability at least $1 - q$, where the probability is taken over the algorithm's internal randomness, and W is the total weight of all items so far.*

Proof. This lower bound follows since unweighted SWOR is a special case of weighted SWOR. We consider an input stream of W items, each with a weight of 1, and apply Theorem 2.3. □

[1] The algorithm in [14] divides execution into rounds; the exact definition of a round does not impact our analysis here, and is hence not provided.

3 WEIGHTED SWOR VIA PRECISION SAMPLING

We now present an algorithm for weighted SWOR on distributed streams. Our algorithm utilizes the general algorithmic framework of *precision sampling*, introduced by Andoni, Krauthgamer, and Onak [2]. The framework of the algorithm is relatively straightforward. When an update (e_i, w_i) is received at a site, the site generates a "key" $v_i = w_i/t_i$, where t_i is a generated exponential random variable. The coordinator then keeps the stream items (e_i, w_i) with the top s largest keys v_i. We first state a result on exponential scaling that allows for the basic correctness of our sampling algorithm.

Proposition 3.1 ([30], Equation 11.7 and Remark 1). *Let $\mathcal{S} = \{(e_1, w_1), (e_2, w_2), \ldots, (e_n, w_n)\}$ be a set of items, where each item has an identifier e_i and a weight w_i. Suppose for each $i \in [n]$ we generate a key $v_i = w_i/t_i$, where the t_is are i.i.d. exponential random variables with rate 1 (i.e., with pdf $p(x) = e^{-x}$ for $x \geq 0$). For $k \in [n]$, let the anti-ranks $D(k)$ be defined as the random variable indices such that $v_{D(1)} \geq v_{D(2)} \geq \cdots \geq v_{D(n)}$. For $s \leq n$, let $S(s) \subset [n]$ be the set of items (e_i, w_i) such that $i = D(k)$ for some $k \in [s]$ (i.e. v_i is among the top s largest keys). Then,*

- *$S(s)$ is a weighted SWOR from the set of items $\mathcal{S}$.*
- *We have the distributional equality:*

$$v_{D(k)} = \left(\sum_{j=1}^{k} \frac{E_j}{\sum_{q=j}^{n} w_{D(q)}}\right)^{-1}$$

where the random variables $E_1, \ldots, E_n$ are i.i.d. exponential random variables with rate 1, and are independent of the anti-rank vector $(D(1), D(2), \ldots, D(n))$.

The above remark demonstrates that taking the items with the top s largest keys indeed gives a weighted sample without replacement. The distributional equality given in the second part of Proposition 3.1 will be used soon in our analysis. Note that while the above results require continuous random variables to be used, we show that our results are not effected by limiting ourselves to a machine word of precision when generating the exponentials (Proposition 3.11). Thus our expected message complexity and runtime take into account the complexity required to generate the exponentials to the precision needed by our algorithm.

To reduce the number of messages sent from the sites to the coordinator, each site locally filters out items whose keys it is sure will not belong to the globally s largest keys. In order to achieve this, our algorithm divides the stream into *epochs*. The coordinator continuously maintains a threshold u, equal to the value of the smallest key v of an item (e, w, v) held in its sample set S, where S always holds the items (e_i, w_i, v_i) with the s largest values of v_i. For $r = \max\{2, k/s\}$, whenever $u \in [r^j, r^{j+1})$, the coordinator declares to all sites that the

algorithm is in epoch i (at the beginning, the epoch is 0 until u first becomes equal to or larger than r). Note that this announcement requires k messages to be sent from the coordinator, and must be done once per epoch.

If the algorithm is in epoch j, and a site receives an update (e_i, w_i), it generates key v_i and sends (e_i, w_i, v_i) to the coordinator if and only if $v_i > r^j$. The coordinator will add the update (e_i, w_i, v_i) to the set S, and if this causes $|S| > s$, it removes the item (e, w, v) from S which has the smallest key v of all items in S. This way, at any time step t, the coordinator holds the items with the s-largest key values $v_i = w_i/t_i$, and outputs the set S as its weighted sample.

A complication in analyzing this algorithm is that it is not possible to directly connect the total weight received so far to the number of epochs that have progressed. For instance, it is not always true that the larger the total weight, the more epochs have elapsed (in expectation). For instance, if a few (fewer than s) items with a very large weight are received, then the total weight received can be large, but the sth largest key is still 0, so that we are still in the zeroth epoch.

To handle this situation, we introduce a *level set* procedure. The idea is to *withhold* heavy items seen in the stream from being sent to the sampler until they are no longer heavy. Here, by releasing an update (e_i, w_i) to the sampler we mean generating a key v_i for the item (e_i, w_i) and deciding whether to accept (e_i, w_i, v_i) into the sample set S. Specifically, we only release a heavy item to the sampler when its weight is no more than a $1/4s$ fraction of the total weight released to the sampler so far. We now defined the *level* of an update (e, w) below (Definition 3.2).

Definition 3.2. The level of an item (e, w) is the integer $j \geq 0$ such that $w \in \left[r^j, r^{j+1}\right)$. If $w \in [0, r)$, we set $j = 0$.

For $j > 0$, the *level set* D_j will consist of the first $4rs$ items in the stream that are in level j. We do this for all $j \geq 0$, but note that we can clearly stop at $j = \log(W)/\log(r)$, and we do not need to explicitly initialize a level set until at least one item is sent to the set. The coordinator stores the set D_j. As long as $|D_j| < 4rs$, if a site received an item (e, w) that is in level j, the item is sent directly to the coordinator without any filtering at the site, and then placed into D_j. We call such a message an "early" message, as the item (e, w) is withheld from the sampling procedure. We call all other messages which send an item (e, w, v) from the site to the coordinator "regular" messages. Similarly, we call an update that resulted in an early message as an "early update", and all other updates as "regular updates". Note that a regular update may not result in a regular message, if the key of the regular update is smaller than the threshold for the current epoch.

We call the level set D_j *unsaturated* if $|D_j| < 4rs$ at a given time, and *saturated* otherwise. Each site stores a binary value $\texttt{saturated}_j$, which is initialized to `false`, indicating that $|D_j| < 4rs$. Once $|D_j| = 4rs$, the level set is saturated, and the coordinator generates keys $v_i = w_i/t_i$ for all items $(e_i, w_i) \in D_j$. For each new item-key pair, it then adds (e_i, w_i, v_i) to S if v_i is in the top s largest keys in $S \cup \{(e_i, w_i, v_i)\}$. At this point, the coordinator announces to all the sites that the level set D_j has been saturated (who then set $\texttt{saturated}_j$ = `true`), and thereafter no early updates will be sent for this level set.

Note that in this procedure, the set S will only be a weighted sample over the updates in level sets that are saturated. Since our algorithm must always maintain a weighted sample over *all* stream items which have been received so far, we can simply simulate generating the keys for all items in the unsaturated level sets D_j, and take the items with the top s largest keys in $S \cup (\cup_{j \geq 0} D_j)$ as the weighted sample (see the description of this in the proof of Theorem 3.12). The algorithm for weighted SWOR is described in Algorithm 1 (algorithm at the site), and Algorithms 2, 3 (algorithm at the coordinator).

We now observe the following result of this level-set procedure.

Lemma 3.3. *At any time step, let $(e_1, w_1), \ldots, (e_t, w_t)$ be all items so far that are in saturated level sets. For any $i \in [t]$, $w_i \leq \frac{1}{4s} \sum_{p \in [t]} w_p$.*

Proof. The reason is that for each item (e_i, w_i) in a saturated level set, there are at least $4rs$ items in the same level set, whose weights are within a factor of r of w_i. Thus w_i can be no more than $\frac{1}{4s}$ of the total weight of this level set, and hence no more than $\frac{1}{4s}$ of the total weight. □

3.1 Analysis

We now analyze the message complexity of the algorithm. Let $r = \max\{2, k/s\}$, and let u be the s-th largest value of the keys v_j's given to the coordinator. As described, we define an epoch i as the sequence of updates such that $u \in [r^i, r^{i+1})$. Set $W = \sum_{i=1}^{n} w_i$, where n is the total number of updates at the end of the stream.

Let $(e_1, w_1), \ldots, (e_n, w_n)$ be the set of all stream items. For the sake of analysis, we consider a new ordering of the stream items, which corresponds to the order in which the keys v_i are generated for the items e_i in our algorithm. In other words, we assume the items are ordered $(e_1, w_1), \ldots, (e_{n'}, w_{n'})$, such that the key v_i for e_i was generated before the key for e_{i+1}, and so on. This reordering is simply accomplished by adding each regular item to the ordering as they arrive, but withholding each early item, and only adding an early item to the ordering once the level set D_j that it was sent to is saturated. Note that at the end of the

Algorithm 1: Weighted SWOR: Algorithm at Site i

```
// Initialization
for each level j ≥ 0 do
    saturated_j ← false
// we will have that u_i = r^j whenever the
   algorithm is in epoch j
u_i ← 0, r ← max{2, k/s}
// End Initialization
while true do
    if receive item (e, w) then
        // e an item id and w a positive weight
        Let j be the level set of (e, w), i.e. the integer j
          such that w ∈ [r^j, r^{j+1}). If w ∈ (0, 1), we set
          j = 0
        if saturated_j = false then
            Send (early, e, w) to the coordinator
        else
            // saturated_j = true
            Let t be an exponential random variable
            v ← w/t
            if v > u_i then
                Send (regular, e, w, v) to the
                  coordinator
    if receive ("level saturated", j) from coordinator then
        Set saturated_j ← true
    if receive ("update epoch", r^j) from coordinator then
        u ← r^j
```

Algorithm 2: Weighted SWOR: Algorithm at Coordinator

```
// Initialization
for each level j ≥ 0 do
    saturated_j ← false
    D_j ← ∅ // set of early messages in level j
u ← 0 // sth max of keys of regular items
S ← ∅ // the current sample at the coordinator
r ← max{2, k/s}
// End Initialization
while true do
    if receive (early, e, w) from site i then
        Let j be the level set of (e, w), i.e. the integer j
          such that w ∈ [r^j, r^{j+1}). If w ∈ (0, 1), we set
          j = 0
        Generate exponential t with rate 1
        Generate key v = w/t
        Add (e, w, v) to D_j
        if |D_j| ≥ 4sr then
            for each (e', w', v') ∈ D_j: do
                Add-to-Sample(S, (e', w', v'))
            D_j ← ∅, saturated_j ← true
            Broadcast ("level saturated", j) to all sites
    if receive (regular, e, w, v) from site i then
        if v > u then
            Add-to-Sample(S, (e, w, v))
    if receive query for sample of s items then
        Return items with the s largest keys in
          S ∪ (∪_j D_j)
```

Algorithm 3: Weighted SWOR: Add-to-Sample$(S, (e_i, w_i, v_i))$

Input: S is the current sample at the coordinator, and (e_i, w_i, v_i) the item to be inserted

```
S ← S ∪ {(e_i, w_i, v_i)}
if (|S| > s) then
    Let v_{j_min} = arg min_{v_t} {v_t | (e_t, w_t, v_t) ∈ S}
    S ← S \ {(e_{j_min}, w_{j_min}, v_{j_min})}
u_old ← u
u ← min_{(e,w,v)∈S} v
if u ∈ [r^j, r^{j+1}) and u_old ∉ [r^j, r^{j+1}) for some j then
    Broadcast ("update epoch", r^j) to all sites
```

stream, some level sets may not be saturated, thus $n' \leq n$. The remaining $n - n'$ items will have already been sent to the coordinator as early messages, and since no exponentials t_i will have been generated for them when the stream ends, we can ignore them when analyzing the message complexity of the regular stream items. Note that this ordering is a deterministic function of the original ordering and weights of the stream, and has no dependency on the randomness of our algorithm.

Now let $(e_{p_1}, w_{p_1}), (e_{p_2}, w_{p_2}), \ldots, (e_{p_\tau}, w_{p_\tau})$ be the subsequence of the regular items in $(e_1, w_1), \ldots, (e_{n'}, w_{n'})$, so that $p_1 \leq p_2 \leq \cdots \leq p_\tau$. We group the regular items $e_{p_1}, e_{p_2}, \ldots, e_{p_\tau}$ into sets Ω_i^j as follows. For any $t \in [n']$, let W_t be the total weight seen in the stream up to time t under the new ordering. Thus $W_t = \sum_{i=1}^{t} w_i$. We know by construction of the level sets, at any point in time that if e_{p_i} is a regular item then $w_{p_i} \leq \epsilon_0 W_{p_i - 1}$, where $\epsilon_0 = \frac{1}{4s}$. For any $i \geq 1$, let Ω_j be the set of all regular items (e_{p_i}, w_{p_i}) such that $W_{p_i} \in (sr^{j-1}, sr^j)$. Note that by definition, $\Omega_j = \emptyset$ for all $j > z$,

where $z = \lceil \log(W/s)/\log(r) \rceil$. We now further break Ω_i into blocks $\Omega_i^1, \Omega_i^2, \ldots, \Omega_i^{q_i}$, such that Ω_i^t consists of all regular items (e_{p_i}, w_{p_i}) such that $W_{p_i} \in (sr^{i-1}(1+\epsilon)^{t-1}, sr^{i-1}(1+\epsilon)^t)$, for some value $\epsilon = \Theta(1)$ which we will later fix. Note that $q_i < \epsilon^{-1}\log(r)$.

Let Y_i indicate the event that the i-th regular item caused a message to be sent, where we order the items via the new ordering as above. If $\mathfrak{p}$ is the total number of regular items, then we would like to bound $\mathbb{E}\left[\sum_{i=1}^{\mathfrak{p}} Y_i\right]$. Note that the property of an item being regular is independent of the randomness in the algorithm, and is defined deterministically as a function of the ordering and weights of the stream. Thus $\mathfrak{p}$ is a fixed value depending on the stream itself. Then $\mathbb{E}[Y_i] = \sum_{j=1}^{\infty} \mathbb{E}\left[Y_i|\mathcal{E}_{i,j}\right] \Pr\left[\mathcal{E}_{i,j}\right]$ where $\mathcal{E}_{i,j}$ is the event that we are in epoch j when the i-th update is seen. Note that since $\mathcal{E}_{i,j}$ only depends on the values $t_1, \ldots, t_{i-1}$, it follows that $\mathbb{E}\left[Y_i|\mathcal{E}_{i,j}\right] = \Pr\left[t_i < w_i/r^j\right] \leq w_i/r^j$. Thus

$$\mathbb{E}[Y_i] \leq w_i \sum_{j=1}^{\infty} r^{-j} \Pr\left[\mathcal{E}_{i,j}\right]$$

Our goal will now be to bound the above quantity. We will only concern ourselves with $\mathcal{E}_{i,j}$ when e_i is a regular message, since the early messages will be sent to the coordinator deterministically.

To bound $\Pr\left[\mathcal{E}_{i,j}\right]$, we must bound the probability that we are in an epoch j much longer than expected. To do this, we develop a tail bound on the probability that the s-th largest key $v_i = w_i/t_i$ is much smaller than expected. To do so, we will first need the following standard tail bound on sums of exponential random variables.

Proposition 3.4. *Let $E_1, \ldots, E_k$ be i.i.d. mean 1 exponential random variables, and let $E = \sum_{i=1}^k E_i$. Then for any $c \geq 1/2$,*

$$\Pr[E > ck] < \lambda e^{-Cc}$$

where $\lambda, C > 0$ are fixed, absolute constants.

Proof. The moment generating function of an exponential E_i with mean 1 is given by $\mathbb{E}\left[e^{tE_i}\right] = \frac{1}{1-t}$ for $t < 1$. Thus $\mathbb{E}\left[e^{tE}\right] = \mathbb{E}\left[e^{t\sum_{i=1}^k E_i}\right] = \prod_{i=1}^k \mathbb{E}\left[e^{tE_i}\right] = (\frac{1}{1-t})^k$. Setting $t = 1/2$, then by Markov's inequality, $\Pr[E > ck] < \Pr\left[e^{tE} > e^{tck}\right] \leq \frac{2^k}{e^{1/2ck}} \leq e^{-1/2ck+k} = \lambda e^{-\Omega(c)}$ for any $c \geq 1/2$ and a fixed constant $\lambda > 0$ as needed. □

We now introduce our main tail bound on the behaviour of the s-th largest key v_i.

Proposition 3.5. *Let $w_1, \ldots, w_t$ be any fixed set of weights, and let $W = \sum_{i=1}^t w_i$. Fix $\ell \in [t]$, and suppose that $w_i \leq \frac{1}{2\ell} W$ for all $i \in [t]$. Let $v_i = w_i/t_i$ where the t_i's are i.i.d. exponential random variables. Define the anti-ranks $D(k)$ for $k \in [t]$ as the random variables such that $v_{D(1)} \geq v_{D(2)} \geq \cdots \geq v_{D(t)}$. Then for any $c \geq 1/2$, we have*

$$\Pr\left[v_{D(\ell)} \leq \frac{W}{c\ell}\right] \leq O(e^{-Cc})$$

where $C > 0$ is a fixed constant.

Proof. We note that $1/v_i$ is distributed as an exponential with rate w_i. The exponential order statistics of independent non-identical exponential random variables were studied by Nagaraja [30], who demonstrates that the ℓ-th largest value of v_k in $\{v_1, v_2, \ldots, v_t\}$ is distributed as (see Proposition 3.1, or equation 11.7 in [30]):

$$v_{D(\ell)} = \left(\sum_{j=1}^{\ell} \frac{E_j}{W - \sum_{q=1}^{j-1} w_{D(q)}}\right)^{-1} \geq \frac{W}{2}\left(\sum_{j=1}^{\ell} E_j\right)^{-1}$$

where the E_i's are i.i.d. exponential random variables with mean 1 that are independent of the anti-ranks $D(1), \ldots, D(t)$. Here, for the inequality, we used the fact that each regular item is at most a $1/(2\ell)$ heavy hitter at every intermediary step in the stream. It follows that if $v_{D(\ell)} \leq W/(c\ell)$, then $\sum_{j=1}^{\ell} E_j \geq \ell c/2$ Which occurs with probability $O(e^{-Cc})$ for some fixed constant $C > 0$ by Proposition 3.4. □

Proposition 3.6. *Let (e_i, w_i) be any regular stream item, and let a_i, b_i be such that $e_i \in \Omega_{a_i}^{b_i}$. Then we have:*

$$\sum_{j=1}^{\infty} r^{-j} \Pr\left[\mathcal{E}_{i,j}\right] \leq O\left(r^{-a_i+2} e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}} + r^{-a_i+1}\right)$$

where $C > 0$ is some absolute constant.

Proof. First note for $j \geq a_i - 1$, we have $\sum_{j \geq k_i}^{\infty} r^{-j} \Pr\left[\mathcal{E}_{i,j}\right] \leq 2r^{-a_i+1}$. So we now bound $\Pr\left[\mathcal{E}_{i,j}\right]$ for $j < a_i$. Let $v_q = w_q/t_q$ for $q \in [i-1]$, and let $D(p)$ be the anti-ranks of the set $\{v_1, \ldots, v_{i-1}\}$. We note then that if $\mathcal{E}_{i,j}$ holds, then $v_{D(s)} < r^{j+1}$ by definition. Thus $\Pr\left[\mathcal{E}_{i,j}\right] \leq \Pr\left[v_{D(s)} \leq r^{j+1}\right]$. Note that by definition of the level sets $\Omega_{a_i}^{b_i}$, the total weight of all items seen up to time $i-1$ is $W_{i-1} > W_i \frac{1}{(1+\epsilon_0)} > sr^{a_i-1}\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}$. Here we have used the fact that, by construction of the ordering, no item w_i has weight greater than $\epsilon_0 W_i$ where $\epsilon_0 = \frac{1}{4s}$. This fact will also be needed to apply Proposition 3.5.

It then suffices to bound $\Pr\left[v_{D(s)} < r^{j+1}\right] = \Pr\left[v_{D(s)} < \frac{1}{c}W_{i-1}/s\right]$, where $c = W_{i-1}/(r^{j+1}s) \geq \frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)} r^{a_i-j-2}$. Note that since $j \leq i-2, b_i \geq 1$ and $\epsilon_0 = \frac{1}{4s} < 1/2$, we have $c > 1/2$. So by Proposition 3.5, $\Pr\left[\mathcal{E}_{i,j}\right] \leq \Pr\left[v_{D(s)} < \frac{1}{c}W_{i-1}/s\right] = O(e^{-Cc}) = O(e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)} r^{a_i-j-2}})$, where $C > 0$ is some absolute constant. Thus

$$r^{-j} \Pr\left[\mathcal{E}_{i,j}\right] \leq O\left(e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)} r^{a_i-j-2}} r^{-j}\right).$$

If $j = a_i - 2$, this is $O(r^{-a_i+2}e^{-\frac{C(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}})$. In general, if $j = a_i - 2 - \ell$ for any $\ell \geq 1$, then this bound becomes:

$$O\left(r^{-a_i+2+\ell}e^{-\frac{C(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}}r^{\ell}\right) = O\left(r^{-a_i+2-\ell}e^{-\frac{C(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}}\right)$$

where here we used the fact that $(e^{-x})^y = O(e^{-y}x^{-2})$ for any $x \geq 2$ and $y \geq \tau$ where $\tau > 0$ is some constant. Thus $\sum_{\ell=0}^{a_i-1} r^{-a_i+2-\ell}\Pr\left[\mathcal{E}_{i,j}\right] = O(r^{-a_i+2}e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}})$ since $r \geq 2$, which completes the proof. □

Lemma 3.7. *Let $\mathfrak{p}$ be the number of regular items in the stream. If $Y_1, Y_2, \ldots$ are such that Y_i indicates the event that the i-th regular stream item (in the ordering defined above) causes a message to be sent, then:*

$$\mathbb{E}\left[\sum_{i=1}^{\mathfrak{p}} Y_i\right] = O\left(sr\frac{\log(W/s)}{\log(r)}\right).$$

Proof. Let $\mathcal{W}_i^j$ be the weight in set Ω_i^j. In other words, $\mathcal{W}_i^j = \sum_{w_{p_t} \in \Omega_i^j} w_{p_t}$. Recall we set $z = \lceil \log(W/s)/\log(r) \rceil$, and note that $\Omega_i = \emptyset$ for $i > z$. Also note that by construction we have $W_i^j < sr^{i-1}(1+\epsilon)^j$. Recall q_i is the number of sets of the form Ω_i^j for some j, and that $q_i \leq O(\epsilon^{-1}\log(r))$ by construction. Using Proposition 3.6, we have:

$$\begin{aligned}
\mathbb{E}\left[\sum_i Y_i\right] &\leq \sum_{i=1}^{z}\sum_{j=1}^{q_i}\sum_{e_{p_i} \in \Omega_i^j} w_{p_i} O\left(r^{-i+2}e^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}} + r^{-i+1}\right) \\
&\leq \sum_{i=1}^{z}\sum_{j=1}^{q_i} \mathcal{W}_i^j O\left(r^{-i+2}e^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}} + r^{-i+1}\right) \\
&\leq O(s)\sum_{i=1}^{z}\sum_{j=1}^{q_i}(1+\epsilon)^j\left(re^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}} + 1\right) \\
&= O(rs)\sum_{i=1}^{z}\sum_{j=1}^{q_i}(1+\epsilon)^j e^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}} \\
&\quad + O(s)\left(\sum_{i=1}^{z}\sum_{j=1}^{q_i}(1+\epsilon)^j\right)
\end{aligned}$$

Setting $\epsilon = .5$, we have $(1+\epsilon_0) < (1+\epsilon)$, this is

$$\leq O(sr)\sum_{i=1}^{z}\sum_{j=1}^{q_i}\frac{1.5^j}{\exp\left(C(1.5)^{j-2}\right)} + s\left(\sum_{i=1}^{z}O(r)\right)$$

Now by the ratio test $\sum_{j=1}^{\infty}\frac{(1.5)^j}{\exp(C(1.5)^{j-2})}$ converges absolutely to a constant (recalling $C > 0$ is just some absolute constant), so the whole sum is:

$$\leq sr\sum_{i=1}^{z}O(1) + sr\left(\sum_{i=1}^{z}O(1)\right) = O\left(sr\frac{\log(W/s)}{\log(r)}\right)$$

where we used $z = \lceil \log(W/s)/\log(r) \rceil$. □

We now bound the number of epochs used in the algorithm. We let the random variable ζ denote the total number of epochs in the algorithm.

Proposition 3.8. *If ζ is the number of epochs in the algorithm, then if $z = \lceil \frac{\log(W/s)}{\log(r)} \rceil$, then*

$$\mathbb{E}[\zeta] \leq 3\left(\frac{\log(W/s)}{\log(r)} + 1\right).$$

Proof. After epoch $z + \ell$ for any $\ell \geq 0$, we have that $u > r^{\ell}W$, where u is the value of the s-th largest key at the coordinator. Let $Y = |\{i \in [n] \mid v_i \geq r^{\ell}W\}|$. Since the pdf of an exponential random variable is upper bounded by 1, it follows that $\Pr\left[v_i \geq r^{\ell}W/s\right] = \Pr\left[t_j \leq sr^{-\ell}\frac{w_j}{W}\right] \leq sr^{-\ell}\frac{w_j}{W}$, thus $\mathbb{E}[Y] \leq sr^{-\ell}$. Thus $\Pr[\zeta \geq z + \ell] \leq \Pr[Y \geq s] \leq r^{-\ell}$, where the last inequality follows by a Markov bound. Thus

$$\begin{aligned}
\mathbb{E}[\zeta] &\leq z + \sum_{\ell \geq 1}(z+\ell)\Pr[\zeta \geq z + \ell] \\
&\leq z + \sum_{\ell \geq 1}(z+\ell)r^{-\ell} \leq 3z.
\end{aligned}$$

□

Lemma 3.9. *The total expected number of messages sent by the algorithm is $O\left(sr\frac{\log(W/s)}{\log(r)}\right)$, where $r = \max\{2, k/s\}$. Thus this can be rewritten as*

$$O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right).$$

Proof. For each level set B_t for $t < \log(W/s)/\log(r)$, at most $4rs + k$ messages are sent, corresponding to the $4rs$ messages sent by sites to saturate the set, and then k messages coming from the reply from the coordinator to all k sites announcing that the set B_t is full. This gives a total of $(4rs + k)\log(W/s)/\log(r)$ messages. Finally, there are at most s items with weight greater than W/s, and thus we pay one message for each of these when they arrive at a site. The level sets corresponding to values greater than W/s will never be saturated, so the total message complexity to handle the early messages is $O(sr\log(W/s)/\log(r))$ as needed.

Next, we pay k messages at the end of every epoch. Thus if ζ is the number of epochs in the algorithm, the expected number of messages sent due to the end of epochs is

$$\mathbb{E}\left[\sum_{i=1}^{\zeta}k\right] = k\mathbb{E}[\zeta] < k\frac{\log(W/s)}{\log(r)} = O(sr\frac{\log(W/s)}{\log(r)})$$

by Proposition 3.8. Finally, the expected number of messages sent due to regular stream items is $O(sr\log(W/s)/\log(r))$ due to Lemma 3.7, which completes the proof of the message complexity. □

Proposition 3.10. *The algorithm described in this section can be implemented, without changing its output behavior, to*

use $O(s)$ memory and $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ expected total runtime at the coordinator, and $O(1)$ memory and $O(1)$ processing time per update at each site. Note that all four of these bounds are optimal.

Proof. We first consider the sites. Note that each site only generates a random variable, checks a bit to see if a level set is saturated, and then makes a single comparison to decide whether to send an item, all in $O(1)$ time. For space, note that each site only needs to store a bit that determines whether level set D_j is full for $j \leq \log(W)/\log(r)$. For all level sets j with $j \geq \log(W)/\log(r)$, no item will ever arrive in this level since no item has weight more than W. Thus, to store the bits $\texttt{saturated}_j$, it suffices to store a bit string of length $\log(W)/\log(r)$, which is at most $O(1)$ machine words.

We now consider the coordinator. For space, we demonstrate how the level-set procedure can be carried out using less than $O(sr\log(W)/\log(r))$ space (which is the total numer of items that can be sent to level sets). For each item that arrives at a level set, we generate its key right away. Now note that the items in the level sets with keys that are not among the s largest keys in all of the level sets will never be sampled, and thus never change the set S or the behavior of the algorithm, thus there is no point of keeping them. So at any time, we only store the identities of the items in the level sets with the top s keys (and which level they were in). Call this set S_{level} To determine when a level set becomes saturated, we keep an $O(\log(rs))$-bit counter for each level set D_j, which stores $|D_j|$. Note that we only need to store this for the levels $j \leq \log(W/s)/\log(r)$, since at most s items have weight more than W/s, and such level sets will never be saturated. When a level set D_j becomes saturated, for each item $(e, w, v) \in S_{level}$ such that (e, w) is in level j, we send this item to the main sampler (i.e., we decide whether or not to include it in S based on the size of its key). The result is the same had the entire level set been stored at all times, and only $O(s)$ machine words are required to store S_{level}, and $O(\log(sr)\log(W)/\log(r)) = O(\log(s)\log(W))$-bits for the counters, which is $O(\log(s)) < O(s)$ machine words, so the total space is $O(s)$ machine words.

For the runtime of the coordinator, each early item and regular item is processed in $O(1)$ time. When a set is saturated, it takes at most $O(s)$ work to send these new items into the main sampler, and this occurs at most $O(\log(W/s)/\log(r))$ times, for a total of $O(s\log(W/s)/\log(r))$ work. Since every other message the coordinator recieves requires $O(1)$ work, the result follows from Lemma 3.9. □

We now remark that, up until this point, we have not considered the bit complexity of our messages, or the runtime required to generate the exponentials to the required precision. We address this issue now.

Proposition 3.11. *The algorithm for weighted SWOR can be implemented so that each message requires an expected $O(1)$ machine words, and moreover, for any constant $c \geq 1$ uses $O(1)$ machine words with probability at least $1 - W^{-c}$. Each exponential can be generated in time $O(1)$ in expectation, and, for any constant $c \geq 1$, time $O(1)$ with probability at least $1 - W^{-c}$.*

Proof. To see this, we note that a site only needs to generate enough bits to determine whether a given key is large enough to be sent. Recall that the quantile function for the exponential distribution is given by $F^{-1}(p) = -\ln(1-p)$ and, since $1-p$ and p are distributed the same for a uniform variable, an exponential variable t_i can be generated by first generating a uniform random variable U, and outputting $-\ln(U)$ [27]. Thus, if the algorithm is in epoch j, one can simply generate U bit by bit, until one can determine whether $w_i/t_i > r^j$ or not. So each bit of U generated cuts the remaining probability space by a factor of 2. So for any threshold τ, it requires only $O(1)$ bits to be generated in expectation to determine whether $-\ln(U) < \tau$, and since the probability space is cut by a constant factor with each bit, only $O(\log(W))$ bits (or $O(1)$ machine words) are needed with high probability in W. Thus the coordinator generates the message in expected $O(1)$ time and with high probability, and the size of the message is an expected $O(1)$ machine words with high probability. Similarly, when the coordinator decides whether to accept a sample into the set, it again needs only to check the exponential against a threshold, which requires generating at most an additional expected $O(1)$ bits and with high probability, which completes the proof. □

Theorem 3.12. *The algorithm of this section for weighted sampling without replacement uses an expected $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ messages and maintains continuously at every point in the stream a uniform weighted sample size s of the items seen so far in the stream. The space required for the coordinator is $O(s)$ machine words, and the space required for each site is $O(1)$ words. Each site requires $O(1)$ processing time per update, and the total runtime of the coordinator is $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$.*

Proof. The message complexity follows from Lemma 3.9, and the space and time complexity follow from Proposition 3.10. The bit complexity issues which arise from dealing with exponentials are dealt with in Proposition 3.11. For correctness, note that at any point in time, the coordinator maintains two kinds of sets: the set S which consists of the top s samples, and the level sets $D = \cup_{j\geq 0} D_j$. To obtain a true weighted SWOR of the stream up to any point i, the coordinator can simply generate an exponential t_j for each $(e_j, w_j) \in D$, and set $v_j = w_j/t_j$. It then constructs $D' = \{(e_j, w_j, v_j) | (e_j, w_j) \in D\}$, and returns the identifiers e_j in

$D' \cup S$ with the s largest keys v_j. The result is that at any point in time, the coordinator can output the set S of the entries (e_j, w_j, v_j) with the top s largest values of v_j.

Let Δ_ℓ be the identifiers e_j with the top ℓ values of v_j. Since $1/v_j$ is exponentially distributed with rate w_j, for any $\ell \geq 1$, if $v_j \notin \Delta_{\ell-1}$ then it follows that the probability that $1/v_j$ is the ℓ-th smallest (or v_j is the ℓ-th largest) is $\frac{w_j}{\sum_{e_t \notin \Delta_{\ell-1}} w_t}$ (via Proposition 3.1). Thus the coordinator indeed holds a uniform weighted sample of the stream continuously at all points in time. Note that this sample may contain items in a level set D_j which has not yet been filled at the time of query. However, this does not affect the behavior of the algorithm, since an exponential can be generated for such an item early and used as the sample (see proof of Proposition 3.10). By this, we mean that when the algorithm is queried for a weighted sample S over *all* stream items seen so far, it can simply generate a key for each item that is held in an unsaturated level set D_j, and keep the items with the top s largest keys in $S \cup (\cup_{j \geq 0} D_j)$. Note, however, that the true set S will not be modified by this procedure, and the actual items will not be sent into the set S to potentially be sampled until the level set D_j is full. □

4 TRACKING HEAVY HITTERS WITH RESIDUAL ERROR

In this section we demonstrate that our sampling algorithm results in a new algorithm for continuously monitoring heavy hitters with a *residual* error guarantee. This is also known as the heavy hitters tracking problem. We show an $O\left(\left(\frac{k}{\log(k)} + \frac{\log(1/(\epsilon\delta))}{\epsilon}\right)\log(\epsilon W)\right)$ upper bound for the message complexity of the problem, along with a nearly tight lower bound $\Omega\left(\left(\frac{k}{\log(k)} + \frac{1}{\epsilon}\right)\log(\epsilon W)\right)$.

While the monitoring of heavy hitters has been studied substantially in the streaming and distributed streaming literature, to date no distributed algorithm has been designed which obtains heavy hitters with a residual error guarantee. We first formalize these variations of the heavy hitters problem. For a vector $x \in \mathbb{R}^n$, and any $t \in [n]$, let $x_{\text{tail}(t)} \in \mathbb{R}^n$ be the vector which is equal to x except that the top t largest coordinates $|x_i|$ of x are set to 0. The following is the standard notion of the heavy hitters tracking problem.

Definition 4.1. Let $\mathcal{S} = (e_1, w_1), \ldots, (e_n, e_n)$, and let $x^t \in \mathbb{R}^n$ be the vector with $x_i = w_i$ for $i \leq t$, and $x_i = 0$ otherwise. Then an algorithm $\mathcal{P}$ solves the (ϵ, δ) heavy hitters tracking problem if, for any fixed $t \in [n]$, with probability $1 - \delta$, it returns a set S with $|S| = O(1/\epsilon)$ such that for every $i \in [t]$ with $x_i \geq \epsilon \|x^t\|_1$, we have $x_i \in S$.

Now, we introduce the variant of the heavy hitters tracking problem that requires residual error.

Definition 4.2. Let $\mathcal{S} = (e_1, w_1), \ldots, (e_n, e_n)$, and let $x^t \in \mathbb{R}^n$ be the vector with $x_i = w_i$ for $i \leq t$, and $x_i = 0$ otherwise. Then an algorithm $\mathcal{P}$ solves the (ϵ, δ) heavy hitters tracking problem *with residual error* if, for any fixed $t \in [n]$, with probability $1 - \delta$ it returns a set S with $|S| = O(1/\epsilon)$ such that for every $i \in [t]$ with $x_i \geq \epsilon \|x^t_{\text{tail}(1/\epsilon)}\|_1$, we have $x_i \in S$.

Note that the residual error guarantee is strictly stronger, and captures substantially more information about the data set when there are very large heavy items.

Theorem 4.3. *There is a distributed streaming algorithm $\mathcal{P}$ which solves the (ϵ, δ) heavy hitters problem with residual error, and sends an expected $O\left(\left(\frac{k}{\log(k)} + \frac{\log(1/(\epsilon\delta))}{\epsilon}\right)\log(\epsilon W)\right)$ messages. The algorithm uses $O(1)$ space per site, $O(1)$ update time per site, $O(\frac{\log(1/(\delta\epsilon))}{\epsilon})$ space at the coordinator, and $O\left(\left(\frac{k}{\log(k)} + \frac{\log(1/(\epsilon\delta))}{\epsilon}\right)\log(\epsilon W)\right)$ overall runtime at the coordinator.*

Proof. To obtain the bound, we run our weighted SWOR of Theorem 3.12 with $s = 6\frac{\log(1/(\delta\epsilon))}{\epsilon}$, where $C > 0$ is a sufficently large constant. Fix a time step t, and let S^t be the sample obtained by the weighted SWOR at that time. We know that S^t is a weighted SWOR from $(e_1, w_1), \ldots, (e_t, w_t)$. Let x^t be the vector corresponding to these weights, and let $T \subset [t]$ be the set of coordinates such that $x_i \geq \epsilon \|x^t_{\text{tail}(1/\epsilon)}\|_1$. Let $T_0 \subset T$ be the subset of i such the x^t_i is one of the $1/\epsilon$ largest values in x^t. Now fix any $i \in T$, and break S into blocks $S^1, S^2, \ldots, S^{2\log(1/(\delta\epsilon))}$, each of size $3/\epsilon$. Now for each $j \in [2\log(1/(\delta\epsilon))]$, there are at least $2/\epsilon$ items which are not in T_0 (which follows since $|T_0| \leq 1/\epsilon$). Conditioned on a sample $s \in S^j$ not being in T_0, since $x_j \geq \epsilon \|x^t_{\text{tail}(1/\epsilon)}\|_1$ by assumption, the probability that $s = i$ is at least ϵ. Thus the probability that $i \notin S^j \leq (1-\epsilon)^{2/\epsilon} < 1/2$. Repeating $2\log(1/\epsilon)$ times, the probability that $i \in S$ is at least $1 - (\frac{1}{2})^{2\log(1/(\delta\epsilon))} < 1 - (\epsilon\delta)^2$. We can then union bound over all $|T| < 2/\epsilon$ items, so that $T \subset S$ with probability at least $1 - \delta$. To restrict the set S to $O(1/\epsilon)$ items, we simply order the items in S by weight and output the top $2/\epsilon$, which will contain T if S does, which completes the proof of correctness. The remainder of the Theorem then follows from the complexity bounds of Theorem 3.12. □

4.1 Lower Bound for Tracking Heavy Hitters

We now demonstrate an $\Omega(k\log(W)/\log(k) + \log(W)/\epsilon))$ lower bound on the expected message complexity of any distributed algorithm that monitors the heavy hitters in a weighted stream (Definition 4.1). Since the residual error guarantee is strictly stronger, this lower bound extends to monitoring heavy hitters with residual error. The proof can be found in Appendix A.1

Theorem 4.4. *Fix any constant $0 < q < 1$, and let $\epsilon \in (0, 1/2)$. Then any algorithm which $(\epsilon/2, \delta) = (\epsilon, q^2/64)$ solves the heavy hitters tracking problem (Definition 4.1), must send at least $\Omega(k \log(W)/\log(k) + \epsilon^{-1} \log(W))$ messages with probability at least $1 - q$ (assuming $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$). In particular, the expected message complexity of such an algorithm with $\delta = \Theta(1)$ is $\Omega(k \log(W)/\log(k) + \epsilon^{-1} \log(W))$.*

5 L_1 TRACKING

In this section, we demonstrate how our sampling algorithm from Section 3 can be used to design a message-optimal L_1 tracking algorithm, which will close the complexity of this problem. We first recall the definition of an L_1 tracking algorithm.

Definition 5.1. Given a distributed weighted data stream $\mathcal{S} = (e_1, w_1), (e_2, w_2), \ldots, (e_n, w_n)$ and parameters $\epsilon, \delta > 0$, a distributed streaming algorithm (ϵ, δ) solves the L_1 tracking problem if the coordinator continuously maintains a value $\widetilde{W}$ such that, at any fixed time $t \in [n]$, we have $\widetilde{W} = (1 \pm \epsilon)W_t$ with probability $1 - \delta$, where $W_t = \sum_{i=1}^{t} w_i$.

In [14], an L_1 tracker was given that had $O(k \log(W) \log(1/\delta))$ expected messages. An improved bound of $O((k + \sqrt{k} \log(1/\delta)/\epsilon) \log(W))$ expected messages was then provided in [23], along with a lower bound of $\Omega(\frac{\sqrt{k}}{\epsilon} \log(W))$ for all $k \leq \frac{1}{\epsilon^2}$. We remark that while the bounds of [23] are stated as $O(\sqrt{k}/\epsilon \log(W))$ (for constant δ), the actual expected message complexity of their algorithm is $O((k + \sqrt{k} \log(1/\delta)/\epsilon) \log(W))$ (in [23] the authors assume that $k < 1/\epsilon^2$). Nevertheless, the aforementioned bound is, up to this point, the current best known distributed algorithm for L_1 tracking.

In this section, we give tight bounds for the case $k > 1/\epsilon^2$. More specifically, we prove an L_1 tracking algorithm with expected message complexity $O(\frac{k \log(\epsilon W)}{\log(k)} + \frac{\log(\epsilon W) \log(1/\delta)}{\epsilon^2})$. In the following section, we also provide an $\Omega(k \frac{\log(W)}{\log(k)})$ lower bound for the problem. We remark that the results of [23] show an $\Omega(\frac{\sqrt{k}}{\epsilon} \log(W))$ lower bound assuming that $k < \frac{1}{\epsilon^2}$, and they give a matching upper bound in this regime. When $k > \frac{1}{\epsilon^2}$, the lower bound becomes $\Omega(\frac{1}{\epsilon^2} \log(W))$, which can be seen by simply applying the same lower bound on a subset of $1/\epsilon^2$ of the sites. Then when $k > 1/\epsilon^2$, if $k \log(W)/\log(k) < \log(W)/\epsilon^2$, our upper bound is $O(\log(W)/\epsilon^2)$, which is tight. Otherwise, our upper bound is $O(k \log(W)/\log(k))$, which matches our lower bound of $\Omega(k \log(W)/\log(k))$. Thus our protocol is optimal whenever $k > 1/\epsilon^2$. Since tight upper and lower bounds were known for the case of $k < 1/\epsilon^2$ by [23], this closes the complexity of the problem.

We also note that our lower bound assumes $1/\epsilon < W^{1-\xi}$ for some fixed constant $\xi > 0$. Thus our lower bound does not contradict our upper bound of $\Omega(k \log(\epsilon W)/\log(k) + \log(\epsilon W)/\epsilon^2)$ (for $\delta = \Theta(1)$). Observe that for $1/\epsilon = \Theta(W)$, one can simply send every stream update for $O(W)$ communication, which is also clearly a lower bound since any stream update not immediately sent would cause the coordinator to no longer have a $(1 \pm \epsilon)$ approximation.

Note for an L_1 tracking such that the coordinator has a value $\widetilde{W}$ such that $(1 - \epsilon)W_t \leq \widetilde{W} \leq (1 + \epsilon)W_T$ for *every* step $t = 1, 2, \ldots, n$ with probability $1 - \delta$, our algorithm uses $O(\frac{k \log(W/\epsilon)}{\log(k)} + \frac{\log(W/\epsilon) \log(\frac{\log(W)}{\epsilon\delta})}{\epsilon^2})$ expected message complexity. This second result simply comes from union bounding over the $\frac{\log(W)}{\epsilon\delta}$ points in the stream where the L_1 increases by a factor of $(1 + \epsilon)$.

Citation	Message Complexity (upper or lower bound)
[14] + folklore	$O\left(\frac{k}{\epsilon} \log(W)\right)$
[23]	$O\left(k \log(W) + \frac{\sqrt{k} \log(W) \log(1/\delta)}{\epsilon}\right)$
This work	$O\left(\frac{k \log(\epsilon W)}{\log(k)} + \frac{\log(\epsilon W)}{\epsilon^2}\right)$
[23]	$\Omega\left(\frac{\sqrt{\min\{k, 1/\epsilon^2\}}}{\epsilon} \log(W)\right)$
This work	$\Omega\left(k \frac{\log(W)}{\log(k)}\right)$

Algorithm 1: Tracking L_1

`Input`: Distributed stream $\mathcal{S}$
`Output`: a value $\tilde{W}$ such that $\tilde{W} = (1 \pm \epsilon)W_t$ at any step time t with probability $1 - \delta$.

Initalization:
Instantiate weighted SWOR algorithm $\mathcal{P}$ of Theorem 3.12 with $s = \Theta(\frac{1}{\epsilon^2} \log(\frac{1}{\delta}))$

On Stream item (e, w)

(1) Duplicate (e, w), a total of $\ell = \frac{s}{2\epsilon}$ times, to yield $(e^1, w), (e^2, w), \ldots, (e^\ell, w)$. Insert each into the algorithm $\mathcal{P}$.

On Query for L_1 at time t:

(1) Let u be the value stored at the coordinator of the s-th smallest key held in the sample set S.
(2) Output $\tilde{W} = \frac{su}{\ell}$.

The known upper and lower bounds for L_1 tracking on a single time step with failure probability $\delta = \Theta(1)$ are summarized below. Note again, constant L_1 tracking at all points in the stream is accomplished in all cases by setting $\delta = \log(W)/\epsilon$, and note that the $\log(1/\delta)$ does not multiply the $O(k \log(W)/\log(k))$ term in our upper bound. With the

combination of the upper and lower bounds of [23], this completely closes the complexity of the problem.

The proof of Proposition 5.2 and Theorem 5.3 can be found in Appendix A.

Proposition 5.2. *Let $E_1, E_2, \ldots, E_s$ be i.i.d. exponential random variables with mean 1. Then for any $\epsilon > 0$, we have:*

$$\Pr\left[|\sum_{j=1}^{s} E_j - s| > \epsilon s\right] < 2e^{-\epsilon^2 s/5}$$

Theorem 5.3. *There is a distributed algorithm (Algorithm 1), where at every time step the coordinator maintains a value $\widetilde{W}$ such that at any fixed time $\tau \in [n]$, we have*

$$(1-\epsilon)W_\tau \leq \widetilde{W} \leq (1+\epsilon)W_\tau$$

with probability $1-\delta$, where $W_\tau = \sum_{i=1}^{\tau} w_i$ is the total weight seen so far at step τ. The expected message complexity of the algorithm is $O\left(\left(\frac{k}{\log(k)} + \epsilon^{-2}\log(1/\delta)\right)\log\left(\frac{\epsilon W}{\log(\delta^{-1})}\right)\right)$

Corollary 5.4. *There is a distributed algorithm where at every time step the coordinator maintains a value $\widetilde{W}$ such that*

$$(1-\epsilon)W_\tau \leq \widetilde{W} \leq (1+\epsilon)W_\tau$$

for every $\tau \in [n]$ with probability $1-\delta$. The expected message complexity of the algorithm is

$$O\left(\left(\frac{k}{\log(k)} + \epsilon^{-2}\log(\frac{\log(W)}{\delta\epsilon})\right)\log\left(\frac{\epsilon W}{\log(\delta^{-1})}\right)\right).$$

Proof. The result follows from running Theorem 5.3 with $\delta' = \log(W)/(\delta\epsilon)$, and union bounding over the $\log(W)/\epsilon$ time steps t in the stream where W_t increases by a factor of $(1+\epsilon)$. □

5.1 Lower Bound for L_1 Tracking

We now demonstrate an $\Omega(k\log(W)/\log(k) + \log(W)/\epsilon))$ lower bound on the expected message complexity of any distributed L_1 tracker. We remark again that our lower bound does not contradict out upper bound of $O(k\log(\epsilon W)/\log(k) + \log(\epsilon W)/\epsilon^2)$, since it requires $1/\epsilon < W^{1-\xi}$ for some constant ξ.

Note that the lower bound holds for both weighted and unweighted streams, as the hard example in the proof is a stream where all weights are equal to 1. We remark that the $\Omega(\log(W)/\epsilon)$ portion of the bound is only interesting when $k < \frac{1}{\epsilon}$, and in this regime a better lower bound of $\Omega(\frac{\sqrt{k}}{\epsilon}\log(W))$ is given by [23]. We include this portion of the bound in the Theorem for completeness, but remark that the main contribution of the Theorem is the lower bound of $\Omega(k\log(W)/\log(k))$. The proof of the Theorem can be found in Appendix A.2

Theorem 5.5. *Fix any constant $0 < q < 1$. Then any algorithm which $(\epsilon, \delta) = (\epsilon, q^2/64)$ solves the L_1 tracking problem (Definition 5.1), must send at least $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$ messages with probability at least $1-q$ (assuming $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$). In particular, the expected message complexity of such an algorithm with $\delta = \Theta(1)$ is $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$.*

6 CONCLUSIONS

We presented message-efficient algorithms for maintaining a weighted random sample from a distributed stream. Our algorithm for weighted SWOR is optimal in its message complexity, space, as well as processing time. We also presented an optimal algorithm for L_1 tracking and the first distributed algorithms for tracking heavy hitters with residual error. One open question is whether we can obtain matching upper and lower bounds for weighted (or unweighted) sampling with replacement. While we can reduce weighted sampling with replacement to unweighted, known upper bounds are still a $\log(s)$ factor larger than the lower bounds. Another problem is to extend our algorithm for weighted sampling to the sliding window model of streaming, where only the most recent data is taken into account for the sample. Finally, for the residual heavy hitters problem, can one remove the $\log(1/\epsilon)$ factor from our bounds?

Acknowledgments: Rajesh Jayaram and David P. Woodruff thank the Simons Institute for the Theory of Computing where part of this work was done. They also acknowledge support by the National Science Foundation under Grant No. CCF-1815840. Srikanta Tirthapura acknowledges support by the National Science Foundation under grants 1527541 and 1725702.

REFERENCES

[1] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *ACM EuroSys*. 29–42.

[2] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. 2010. Streaming algorithms from precision sampling. *arXiv preprint arXiv:1011.1263* (2010).

[3] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. 2009. Functional Monitoring Without Monotonicity. In *ICALP*. 95–106.

[4] Brian Babcock, Mayur Datar, and Rajeev Motwani. 2002. Sampling from a Moving Window over Streaming Data. In *SODA*. 633–634.

[5] Brian Babcock and Chris Olston. 2003. Distributed Top-k Monitoring. In *ACM SIGMOD*. 28–39.

[6] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. 2009. Space-optimal heavy hitters with strong error bounds. *PODS*.

[7] Vladimir Braverman, Rafail Ostrovsky, and Gregory Vorsanger. 2015. Weighted sampling without replacement from data streams. *IPL* 115, 12 (2015), 923–926.

[8] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. 2012. Optimal sampling from sliding windows. *J. Comput. System Sci.* 78 (2012), 260 – 272.
[9] Jiecao Chen and Qin Zhang. 2017. Improved Algorithms for Distributed Entropy Monitoring. *Algorithmica* 78, 3 (2017), 1041–1066.
[10] Yung-Yu Chung, Srikanta Tirthapura, and David P. Woodruff. 2016. A Simple Message-Optimal Algorithm for Random Sampling from a Distributed Stream. *IEEE Trans. Knowl. Data Eng.* 28, 6 (2016), 1356–1368.
[11] Yung-Yu Chung and Srikanta Tirthapura. 2015. Distinct Random Sampling from a Distributed Stream. In *IPDPS*. 532–541.
[12] Graham Cormode and Minos Garofalakis. 2005. Sketching Streams Through the Net: Distributed Approximate Query Tracking. In *VLDB*. VLDB Endowment, 13–24. http://dl.acm.org/citation.cfm?id=1083592.1083598
[13] Graham Cormode, S. Muthukrishnan, and Ke Yi. 2011. Algorithms for Distributed Functional Monitoring. *ACM Trans. Algorithms* 7, 2 (2011), 21:1–21:20.
[14] Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. 2012. Continuous Sampling from Distributed Streams. *JACM* 59, 2 (2012), 10:1–10:25.
[15] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. 2003. Estimating flow distributions from sampled flow statistics. In *SIGCOMM*. 325–336.
[16] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. 2004. Flow sampling under hard resource constraints. In *SIGMETRICS*. 85–96.
[17] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. 2007. Priority sampling for estimation of arbitrary subset sums. *J. ACM* 54, 6 (2007), 32.
[18] Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted random sampling with a reservoir. *IPL* 97, 5 (2006), 181 – 185.
[19] Rainer Gemulla and Wolfgang Lehner. 2008. Sampling Time-based Sliding Windows in Bounded Space. In *ACM SIGMOD*. 379–392.
[20] Phillip B. Gibbons. 2001. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *VLDB*. 541–550.
[21] Phillip B. Gibbons and Srikanta Tirthapura. 2001. Estimating Simple Functions on the Union of Data Streams. In *ACM SPAA*. 281–291.
[22] Phillip B. Gibbons and Srikanta Tirthapura. 2002. Distributed Streams Algorithms for Sliding Windows. In *ACM SPAA*. 63–72.
[23] Zengfeng Huang, Ke Yi, and Qin Zhang. 2012. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *ACM PODS*. 295–306.
[24] Rajesh Jayaram and David P Woodruff. 2018. Perfect Lp Sampling in a Data Stream. In *IEEE FOCS*. 544–555.
[25] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. 2011. Tight Bounds for Lp Samplers, Finding Duplicates in Streams, and Related Problems. In *PODS*. 49–58.
[26] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. 2006. Communication-efficient Distributed Monitoring of Thresholded Counts. In *ACM SIGMOD*. 289–300.
[27] D. Knuth. 1997. *The art of computer programming: sorting and searching*. Vol. 3.
[28] D. Knuth. 1997. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*.
[29] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. 2005. Finding (Recently) Frequent Items in Distributed Data Streams. In *IEEE ICDE*. 767–778.
[30] HN Nagaraja. 2006. Order statistics from independent exponential random variables and the sum of the top order statistics. *Advances in Distribution Theory, Order Statistics, and Inference* (2006), 173–185.
[31] Srikanta Tirthapura and David P. Woodruff. 2011. Optimal Random Sampling from Distributed Streams Revisited. In *DISC*. Springer-Verlag, Berlin, Heidelberg, 283–297. http://dl.acm.org/citation.cfm?id=2075029.2075065
[32] Srikanta Tirthapura and David P. Woodruff. 2019. Optimal Random Sampling from Distributed Streams Revisited. *CoRR* (2019). https://arxiv.org/abs/1903.12065
[33] J. S. Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Software* 11, 1 (1985), 37–57.
[34] David P. Woodruff and Qin Zhang. 2017. When distributed computation is communication expensive. *Distributed Computing* 30, 5 (2017), 309–323.
[35] Bojian Xu, Srikanta Tirthapura, and Costas Busch. 2008. Sketching asynchronous data streams over sliding windows. *Distributed Computing* 20, 5 (2008), 359–374.
[36] Ke Yi and Qin Zhang. 2013. Optimal Tracking of Distributed Heavy Hitters and Quantiles. *Algorithmica* 65, 1 (2013), 206–223.

A MISSING PROOFS

We first include a proof of Proposition 5.2.

Proposition A.1 (Proposition 5.2). *Let $E_1, E_2, \ldots, E_s$ be i.i.d. exponential random variables with mean 1. Then for any $\epsilon > 0$, we have:*

$$\Pr\left[\left|\sum_{j=1}^{s} E_j - s\right| > \epsilon s\right] < 2e^{-\epsilon^2 s/5}$$

Proof. The moment generating function of $\sum_{j=1}^{s} E_j$ is given by $(\frac{1}{1-t})^s$ for $t < 1$,

$$\Pr\left[\sum_{j=1}^{s} E_j > (1+\epsilon)s\right] < \frac{(\frac{1}{1-t})^s}{e^{t(1+\epsilon)s}}$$
$$\leq \exp\left(-t(1+\epsilon)s + s\log(\frac{1}{1-t})\right)$$

Setting $t = \epsilon$ and taking $\epsilon < 1/2$ we obtain

$$\leq \exp\left(-\epsilon^2 s - \epsilon s - s\log(1-\epsilon)\right)$$
$$\leq \exp\left(-\epsilon^2 s - \epsilon s + s(\epsilon + \epsilon^2/2 + \epsilon^3/3 + \ldots)\right)$$
$$\leq \exp\left(-\epsilon^2 s/5\right)$$

as needed. Next, we have

$$\Pr\left[\sum_{j=1}^{s} E_j < (1-\epsilon)s\right] < \Pr\left[e^{-t\sum_{j=1}^{s} E_j} > e^{-t(1-\epsilon)s}\right]$$
$$\leq \frac{(\frac{1}{1+t})^s}{\exp(-t(1-\epsilon)s)}$$
$$\leq \exp\left(t(1-\epsilon)s - s\log(1+t)\right)$$

setting $t = \epsilon$:

$$\leq \exp\left(-\epsilon^2 s + \epsilon s - s(\epsilon + \epsilon^2/2 + \epsilon^3/3 + \ldots)\right)$$
$$\leq \exp\left(-\epsilon^2 s/5\right)$$

and union bounding over the upper and lower tail bound gives the desired result. □

Below is a restatement and the proof of Theorem 5.3, for the correctness of our L_1 tracking algorithm.

Theorem A.2 (Theorem 5.3). *There is a distributed algorithm (Algorithm 1), where at every time step the coordinator maintains a value $\widetilde{W}$ such that at any fixed time $\tau \in [n]$, we have*

$$(1-\epsilon)W_\tau \leq \widetilde{W} \leq (1+\epsilon)W_\tau$$

with probability $1-\delta$, where $W_\tau = \sum_{i=1}^{\tau} w_i$ is the total weight seen so far at step τ. The expected message complexity of the algorithm is $O\left(\left(\frac{k}{\log(k)} + \epsilon^{-2}\log(1/\delta)\right)\log\left(\frac{\epsilon W}{\log(\delta^{-1})}\right)\right)$

Proof. If W_τ is the total weight seen so far in the stream, then after duplication the weight of the new stream $\mathcal{S}'$ that is fed into $\mathcal{P}$ is $W_\tau \frac{2s}{\epsilon}$. Call the total weight seen up to time τ of the new stream $\overline{W}_\tau$. The duplication has the effect that at any time step τ in the original stream, every item $(e_j^i, w_j) \in \mathcal{S}'$, corresponding to a duplicate of $(e_j^i, w_j) \in \mathcal{S}$, is such that $w_j \leq \frac{\epsilon}{2s}\overline{W}_\tau$. Thus once an update to $\mathcal{S}$ is finished being processed, no item in $\mathcal{S}'$ is more than an $\frac{\epsilon}{2s}$ heavy hitter. Since there are $\ell > s$ duplications made to each update, the level sets used in $\mathcal{P}$ will be immediately saturated the moment the first update in $\mathcal{S}$ with a weight in that level set arrives. Thus there is never any intermediate point in the stream $\mathcal{S}$ where some of the weight of the stream is being withheld in the level sets used by $\mathcal{P}$. Thus the total weight of the stream seen so far at any given time has already been sent to the sample, so each item (e_i, w_i) seen so far in the stream has had a key v_i generated for it.

Now the coordinator holds the value u such that u is the s-th largest value in $\{v_1, v_2, \ldots, v_{\ell\tau}\}$, where ℓ was the number of duplications. Here $v_i = w_i/t_i$, where t_i is an exponetial variable and w_i is a weight of one of the duplicates sent to the stream $\mathcal{S}'$. By results on the order statistics of collections of exponential variables ([30]), we have the distributional equality

$$u = \left(\sum_{j=1}^{s} \frac{E_j}{\overline{W}_\tau - \sum_{q=1}^{j-1} w_{D(q)}}\right)^{-1}$$

where $E_1, E_2, \ldots, E_s$ are i.i.d. exponential random variables, and $D(q)$ are the random variable indices such that $v_{D(1)} \geq V_{D(2)} \geq \cdots \geq V_{D(\tau\ell)}$ (the $D(q)$'s are known as anti-ranks). Since $w_{D(q)} \leq \frac{\epsilon}{2s}\overline{W}_\tau$, it follows that

$$\begin{aligned} u &= \left((1\pm\epsilon)\sum_{j=1}^{s}\frac{E_j}{\overline{W}_\tau}\right)^{-1} \\ &= (1\pm O(\epsilon))\overline{W}_\tau\left(\sum_{j=1}^{s} E_j\right)^{-1} \\ &= (1\pm O(\epsilon))\ell W_\tau\left(\sum_{j=1}^{s} E_j\right)^{-1} \end{aligned}$$

Now by Proposition 5.2, we have $\Pr\left[\left|\sum_{j=1}^{s} E_j - s\right| > \epsilon s\right] < 2e^{-\epsilon^2 s/5} < \delta$, where here we set $s = 10\log(\delta^{-1})/\epsilon^2$. Conditioned on this not occurring, we have $u = (1\pm O(\epsilon))\ell W_\tau \frac{1}{s}$, thus $u\frac{s}{\ell} = (1\pm O(\epsilon))W_\tau$ as required, and the expected message complexity follows by Theorem 3.12 setting $s = \Theta(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$. □

A.1 Proof of Heavy Hitters Lower Bound

Theorem A.3 (Theorem 4.4). *Fix any constant $0 < q < 1$, and let $\epsilon \in (0, 1/2)$. Then any algorithm which $(\epsilon/2, \delta) = (\epsilon, q/64)$ solves the heavy hitters tracking problem (Definition 4.1), must send at least $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$ messages with probability at least $1-q$ (assuming $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$). In particular, the expected message complexity of such an algorithm with $\delta = \Theta(1)$ is $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$.*

Proof. Create $s = \Theta(\frac{1}{\epsilon}\log(W))$ global stream updates (e_i, w_i), such that $w_i = (1+\epsilon)^i\epsilon$ and $w_0 = 1$. Then note for each $i \geq 0$, we have that w_i is an $\epsilon/(1+\epsilon) > \epsilon/2$ heavy hitter in $w_1, w_2, \ldots, w_i$. Note that the total weight of the stream is $\sum_{i=1}^{s} w_i = W$.

Now consider any algorithm $\mathcal{P}$ for $(\epsilon/2, q^2/64)$ heavy hitter tracking, and at any time step t let S denote the set of heavy hitters held by $\mathcal{P}$,). On a given time step $t \in [n]$, let S_t denote the value of S at time t. and let R denote the concatenation of all random coin flips used by the algorithm $\mathcal{P}$ for both the sites and the coordinator. We first claim the following:

Claim 1. *Let $0 < q < 1$ be any constant. Suppose $\mathcal{P}$ is a randomized algorithm which $(\epsilon/2, q/64)$-solves the heavy hitter tacking problem. Suppose the set of heavy hitters it maintains at all times is called S. Then there is a constant $C' = C'(q) > 0$ such that:*

- *The set S changes at least $\frac{C'}{\epsilon}\log(n)$ times with probability at least $1-q$.*

Proof. Consider the above stream instance. Note that on each time step, the new update (e_i, w_i) becomes an $\epsilon/2$ heavy hitter, and therefore should be accepted into the set S. Note moreover that at any time step t, the total weight is

$(1+\epsilon)^t$. Suppose there is a randomized algorithm $\mathcal{P}$ which for each $t \in T$, succeeds in outputting all the ϵ heavy hitters in a set S at time t with probability $1 - q/64$. Let X_t be a random variable that indicates that $\mathcal{P}$ is correct on t. We have $\mathbb{E}[X_t] > 1 - q/64$, and clearly $\mathbb{E}[X_t X_{t'}] \leq 1$ for any t, t'. Thus $\text{Var}(\sum_{i=1}^{\frac{1}{\epsilon}\log(n)} X_i) \leq \frac{1}{\epsilon}\log(n)(1-q/64) + \frac{1}{\epsilon^2}\log^2(n) - \frac{1}{\epsilon^2}\log^2(n)(1-q/64)^2 \leq (q/30)\frac{1}{\epsilon^2}\log^2(n)$ for n larger than some constant. By Chebyshev's inequality:

$$\Pr\left[\left|\frac{1}{\epsilon}\log(n)(1-q/64) - \sum_{i \leq \epsilon^{-1}\log(n)} X_i\right| > \frac{1}{q}\sqrt{q/30}\frac{\log(n)}{\epsilon}\right] < q$$

Thus with probability at least $1-q$, $\mathcal{P}$ is correct on at least a $\frac{C'}{\epsilon}\log(n)$ number of the points in T, for $C' = (1 - q/64 - \frac{1}{\sqrt{30}} -) > 1/2$. Now suppose that, conditioned on this, the algorithm changed S fewer than $\frac{C'}{\epsilon}\log(n)$ times. Note that every time a new item (e_i, w_i) arrives, if $\mathcal{P}$ is correct on time i, it must be the case that $(e_i, w_i) \in S$ once update i is done processing. Thus if S did not change, but $(e_i, w_i) \in S_t$, this means $(e_i, w_i) \in S_{t-1}$ – the coordinator contained e_i in its set before the item arrived! By having e_i's being $O(\log(W))$ bit identifiers and randomizing the identities in the stream, it follows that the probability that this could occur at any time step t is less than $(W/\epsilon)(1/\text{poly}(W)) < W^{-100}$ for e_i's with $O(\log(W))$ bits and a sufficiently large constant. So we can safely condition on this event not occurring without affecting any of the claims of the theorem. It follows that S must change on every time step t on which it is correct, which completes the first claim that S must change at least $\frac{C'}{\epsilon}\log(n)$ times with probability at least $1-q$. □

Observe that the lower bound of of $\Omega(\log(W)/\epsilon)$ messages being sent with probability at least $1-q$ follows directly from the last claim. For the $\Omega(k\log(W)/\log(k))$ lower bound, we construct a new weighted stream. Define $\eta = \Theta(\frac{\log(W)}{\log(k)})$ *epochs*, as follows. In the i-th epoch, each site j receives *one* weighted update (e_i^j, k^i). Note then at the very beginning of the i-th epoch (for any i), the total weight seen in the stream so far is at most $2k^i$. Thus the first update (e_i^j, k^i) that arrives in the i-th epoch will be a $1/2$ heavy hitter, and thus must be accepted into S for the protocol to be correct on that time step.

Now consider any epoch i, and let X_i be the number of sites which sent or received a message from the coordinator in epoch i. Note that X_i is a lower bound on the number of messages sent in epoch i. Let X_i^j indicate that site j sent or recieved a message from the coordinator in epoch i, so that $X_i = \sum_{j \in [k]} X_i^j$. We claim $\mathbb{E}[X_i] = \Omega(k)$. To demonstrate this, consider the first site j^* which recieves an update in epoch i. Since item $(e_i^{j^*}, k^i)$ is immediately a $1/2$ heavy hitter, it must be sent to be correct. Since the protocol is correct on the $ki+1$'st time step with probability at least $1-q/64$, it follows that site j^* must send a message after receiving its update in epoch i with probability $1-q/64$, so $\mathbb{E}\left[X_i^{j^*}\right] > 1 - q/64$.

But note that when a site j receives its update in the i-th epoch, if it has not communicated to the coordinator since the start of the epoch, it does not know the order in which it recieved the item. In particular, it does not know whether it was the first site to receive an item. Since the randomized protocol must be correct on any advesarial fixing of the ordering, it follows that if site j does not know if it recieved the first update, it must nevertheless send it with probability at least $1 - q/64$ to be a correct protocol. In other words $\mathbb{E}\left[X_i^j\right] > 1 - q/64$ *for all* $j \in [k]$, from which $\mathbb{E}[X_i] > (1-q/64)k$, and therefore $\mathbb{E}\left[\sum_{i=1}^{\eta} X_i\right] > (1-q/64)k$ follows. Thus $\mathbb{E}\left[k\eta - \sum_{i=1}^{\eta} X_i\right] < k\eta q/64$, and so by a Markov bound over the last expectation, $\Pr\left[\sum_{i=1}^{\eta} X_i < (1-1/64)k\eta\right] < q$. Since X_i lower bounds the number of messages sent in the i-th epoch, this completes the proof. □

A.2 Proof of L_1 Tracking Lower Bound

We now give the proof of the lower bound stated in Theorem 5.5. We first restate the result.

Theorem A.4 (Theorem 5.5). *Fix any constant $0 < q < 1$. Then any algorithm which $(\epsilon, \delta) = (\epsilon, q/64)$ solves the L_1 tracking problem (Definition 5.1), must send at least $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$ messages with probability at least $1-q$ (assuming $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$). In particular, the expected message complexity of such an algorithm with $\delta = \Theta(1)$ is $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$.*

Proof. We proceed much in the way of Theorem 4.4. We define $\eta = \Theta(\frac{\log(W)}{\log(k)})$ epochs as follows: at the end of the the $i-th$ epoch, for $i = 0, 1, 2, \ldots, \eta - 1$, there will have been exactly k^i global stream updates processed since the start of the stream, each with weight 1. Thus each epoch i is deterministically defined, and contains $k^{i+1} - k^i$ global updates. Let the unweighted updates be $e_1, e_2, \ldots, e_n$ (we can make these weighted by simply adding a weight of 1. Here $n = W$, so the stream is both weighted and unweighted. In each epoch, we partition the updates arbitrarily over the sites k.

Now consider any algorithm $\mathcal{P}$ for L_1 tracking, and at any time step t let u denote the value held by $\mathcal{P}$ which tracks the L_1 (total weight seen so far in the stream). On a given time step $t \in [n]$, let u_t denote the value of u at time t, and let R denote the concatination of all random coin flips used by the algorithm $\mathcal{P}$ for both the sites and the coordinator. We first claim the following:

Claim 2. *Let $0 < q < 1$ be any constant. Suppose $\mathcal{P}$ is a randomized algorithm which $(\epsilon, q/64)$-solves the L_1 tracking problem. Suppose the value it maintains for the L_1 approximation at all times is called u. Then there is a constant $C' = C'(q) > 0$ such that the following holds:*

- *The value u changes at least $\frac{C'}{\epsilon}\log(W)$ times with probability at least $1 - q$.*

Proof. Consider the time steps $T = \{1, \mathsf{rnd}(1+\epsilon), \mathsf{rnd}((1+\epsilon)^2), \ldots, \ldots, n\}$, where the L_1 changes be a factor of $(1+\epsilon)$, where rnd rounds the vaue to the nearest integer (note we can assume n is a power of $(1+\epsilon)$ by construction). Suppose there is a randomized algorithm $\mathcal{P}$ which for each $t \in T$, succeeds in outputting a $(1 \pm \epsilon)$ approximation to t at time t with probability $1 - q^2/64$. Assuming that $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$, we have $|T| > \frac{1}{\epsilon}\log(W)$ distinct such points. Let X_t be a random variable that indicates that $\mathcal{P}$ is correct on t. We have $\mathbb{E}[X_t] > 1 - q/64$, and clearly $\mathbb{E}[X_t X_{t'}] \leq 1$ for any t, t'. Thus $\mathrm{Var}(\sum_{i=1}^{\frac{1}{\epsilon}\log(W)} X_i) \leq \frac{1}{\epsilon}\log(W)(1-q/64) + \frac{1}{\epsilon^2}\log^2(W) - \frac{1}{\epsilon^2}\log^2(W)(1-q/64)^2 \leq (q/30)\frac{1}{\epsilon^2}\log^2(n)$ for n larger than some constant. By Chebyshev's inequality:

$$\Pr\left[\left|\frac{1}{\epsilon}\log(W)(1-q/64) - \sum_{i \leq \epsilon^{-1}\log(W)} X_i\right| > \frac{1}{\sqrt{q}}\sqrt{q/30}\frac{\log(W)}{\epsilon}\right] < q$$

Thus with probability at least $1 - q$, $\mathcal{P}$ is correct on at least a $\frac{C'}{\epsilon}\log(W)$ number of the points in T, for $C' = (1 - q/64 - \frac{1}{\sqrt{30}} -) > 1/2$. Now suppose that, conditioned on this, the algorithm changed u less than $\frac{C'}{\epsilon}\log(W)$ times. But each time u changes, it can be correct for at most one of the values in T, which contradicts the fact that $\mathcal{P}$ is correct on at least a $\frac{C'}{\epsilon}\log(W)$ of these points. So with probability at least $1 - q$, the value of u must change at least $\frac{C'}{\epsilon}\log(W)$ times, as needed. □

Note that the lower bound of $\Omega(\log(W)/\epsilon)$ messages being sent with probability at least $1 - q$ follows directly from the last claim. Note moreover, that our lower bound reduces to $\Omega(\log(W)/\epsilon)$ when $k < 8/\epsilon$, so we can now assume that $k > 8/\epsilon$.

Now consider any epoch i, and let X_i be the number of sites which sent or received a message from the coordinator in epoch i. Let X_i^j indicate that site j sent or recieved a message from the coordinator in epoch i, so that $X_i = \sum_{j \in [k]} X_i^j$. We claim $\mathbb{E}[X_i] = \Omega(k)$. To demonstrate this, first note that at the beginning of an epoch, the current L_1 of the stream is k^i. First condition on the event $\mathcal{E}_i$ that the estimate of the algorithm is correct at the beginning of the epoch. Note $\Pr[\mathcal{E}_i] > 1 - q/64$. By the law of total expectation: $\mathbb{E}[X_i] \geq \mathbb{E}[X_i \mid \mathcal{E}_i](1 - q/64)$.

Now consider the stream of updates σ_i^j in epoch i which send exactly $2k^i$ consecutive updates to each site j. Consider any stream σ_i constructed by composing σ_i^j's for any arbitrary permutation of the j's. Note that if site j^* recieved the first set of updates in σ_i and does not send a message after these updates, the algorithm will have an incorrect L_1 estimate at time $3k^i$. Since this does not occur with probability at least $1 - q/64$, it follows that $\mathbb{E}\left[X_i^{j^*}\right] \geq (1 - q/64)$. But note that this must hold for all sites j, and once a site j receives its set of $2k^i$ updates, since it could have been the first one to recieve the updates (unless it hears otherwise from the coordinator). In other words, since the randomized protocol must maintain the same guarantee for any ordering of the stream, it follows that every site j must also send a message to the coordinator with probability at least $1 - q/64$ after it sees its $2k^i$ updates (unless the coordinator sends a message to it first). Thus $\mathbb{E}\left[X_i^j\right] \geq (1 - q/64)$ for all j, which completes the claim that $\mathbb{E}[X_i] \geq (1 - q/64)k$, giving $\mathbb{E}\left[\sum_{i=1}^{\eta} X_i\right] > k\eta(1 - q/64)$. Thus $\mathbb{E}\left[k\eta - \sum_{i=1}^{\eta} X_i\right] < k\eta q/64$, and so by a Markov bound over the last expectation, $\Pr\left[\sum_{i=1}^{\eta} X_i < (1 - 1/64)k\eta\right] < q$. Since X_i lower bounds the number of messages sent in the i-th epoch, this completes the proof. □