# Optimal Random Sampling from Distributed Streams Revisited

Srikanta Tirthapura[1] and David P. Woodruff[2]

[1] Dept. of ECE, Iowa State University, Ames, IA, 50011, USA.
`snt@iastate.edu`
[2] IBM Almaden Research Center, San Jose, CA, 95120, USA.
`dpwoodru@us.ibm.com`

**Abstract.** We give an improved algorithm for drawing a random sample from a large data stream when the input elements are distributed across multiple sites which communicate via a central coordinator. At any point in time the set of elements held by the coordinator represent a uniform random sample from the set of all the elements observed so far. When compared with prior work, our algorithms asymptotically improve the total number of messages sent in the system as well as the computation required of the coordinator. We also present a matching lower bound, showing that our protocol sends the optimal number of messages up to a constant factor with large probability. As a byproduct, we obtain an improved algorithm for finding the heavy hitters across multiple distributed sites.

**Keywords:** distributed streams, sampling, reservoir sampling

## 1 Introduction

For many data analysis tasks, it is impractical to collect all the data at a single site and process it in a centralized manner. For example, data arrives at multiple network routers at extremely high rates, and queries are often posed on the union of data observed at all the routers. Since the data set is changing, the query results could also be changing continuously with time. This has motivated the *continuous, distributed, streaming model* [8]. In this model there are $k$ physically distributed sites receiving high-volume local streams of data. These sites talk to a central coordinator, who has to continuously respond to queries over the union of all streams observed so far. The challenge is to minimize the communication between the different sites and the coordinator, while providing an accurate answer to queries at the coordinator at all times.

A fundamental problem in this setting is to obtain a random sample drawn from the union of all distributed streams. This generalizes the classic *reservoir sampling* problem (see, e.g., [15], where the algorithm is attributed to Waterman; see also [19]) to the setting of multiple distributed streams, and has applications to approximate query answering, selectivity estimation, and query planning. For example, in the case of network routers, maintaining a random sample from the union of the streams is valuable for network monitoring tasks involving the detection of global properties [13]. Other problems on distributed stream processing, including the estimation of the number of distinct elements [7, 8] and heavy hitters [4, 14, 17, 21], use random sampling as a primitive.

The study of sampling in distributed streams was initiated by Cormode *et al* [9]. Consider a set of $k$ different streams observed by the $k$ sites with the total number of current items in the union of all streams equal to $n$. The authors in [9] show how $k$ sites can maintain a random sample of $s$ items without replacement from the union of their streams using an expected $O((k+s)\log n)$ messages between the sites and the central coordinator. The memory requirement of the central coordinator is $s$ machine words, and the time requirement is $O((k+s)\log n)$. The memory requirement of the remote sites is a single machine word with constant time per stream update. Cormode *et al.* also prove that the expected number of messages sent in any scheme is $\Omega(k + s\log(n/s))$. Each message is assumed to be a single machine word, which can hold an integer of magnitude $(kns)^{O(1)}$.

**Notation.** All logarithms are to the base 2 unless otherwise specified. Throughout the paper, when we use asymptotic notation, the variable that is going to infinity is $n$, and $s$ and $k$ are functions of $n$.

## 1.1 Our Results

Our main contribution is an algorithm for sampling without replacement from distributed streams, as well as a matching lower bound showing that the message complexity of our algorithm is optimal. A summary of our results and a comparison with earlier work is shown in Figure 1.

**New Algorithm:** We present an algorithm which uses an expected

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

number of messages for continuously maintaining a random sample of size $s$ from $k$ distributed data streams of total size $n$. Notice that if $s < k/8$, this number is $O\left(\frac{k\log(n/s)}{\log(k/s)}\right)$, while if $s \geq k/8$, this number is $O(s\log(n/s))$.

The memory requirement in our protocol at the central coordinator is $s$ machine words, and the time requirement is $O\left(\frac{k\log n/s}{\log(1+k/s)}\right)$. The former is the same as that in the protocol of [9], while the latter improves their $O((k+s)\log n)$ time requirement. The remote sites in our scheme store a single machine word and use constant time per stream update, which is clearly optimal.

Our result leads to a significant improvement in the message complexity in the case when $k$ is large. For example, for the basic problem of maintaining a single random sample from the union of distributed streams ($s = 1$), our algorithm leads to a factor of $O(\log k)$ decrease in the number of messages sent in the system over the algorithm in [9].

Our algorithm is simple, and only requires the central coordinator to communicate with a site if the site initiates the communication. This is useful in a setting where a site may go offline, since it does not require the ability of a site to receive broadcast messages.

| | Upper Bound | | Lower Bound | |
|---|---|---|---|---|
| | Our Result | Cormode *et al.* | Our Result | Cormode *et al.* |
| $s < \frac{k}{8}$ | $O\left(\frac{k\log(n/s)}{\log(k/s)}\right)$ | $O(k\log n)$ | $\Omega\left(\frac{k\log(n/s)}{\log(k/s)}\right)$ | $\Omega(k + s\log n)$ |
| $s \geq \frac{k}{8}$ | $O(s\log(n/s))$ | $O(s\log n)$ | $\Omega(s\log(n/s))$ | $\Omega(s\log(n/s))$ |

**Fig. 1.** Summary of Our Results for Message Complexity of Sampling Without Replacement

**Lower Bound:** We also show that for any constant $q > 0$, any correct protocol must send $\Omega\left(\frac{k\log(n/s)}{\log(1+(k/s))}\right)$ messages with probability at least $1 - q$. This also yields a bound of $\Omega\left(\frac{k\log(n/s)}{\log(1+(k/s))}\right)$ on the expected message complexity of any correct protocol, showing the expected number of messages sent by our algorithm is optimal, upto constant factors.

In addition to being quantitatively stronger than the lower bound of [9], our lower bound is also qualitatively stronger, because the lower bound in [9] is on the expected number of messages transmitted in a correct protocol. However, this does not rule out the possibility that with large probability, much fewer messages are sent in the optimal protocol. In contrast, we lower bound the number of messages that must be transmitted in any protocol 99% of the time. Since the time complexity of the central coordinator is at least the number of messages received, the time complexity of our protocol is also optimal.

**Sampling With Replacement.** We also show how to modify our protocol to obtain a random sample of $s$ items from $k$ distributed streams with replacement. Here we achieve a protocol with $O\left(\left(\frac{k}{\log(2+(k/(s\log s)))} + s\log s\right)\log n\right)$ messages, improving the $O((k + s\log s)\log n)$-message protocol of [9]. We obtain the same improvement in the time complexity of the central coordinator.

**Heavy-Hitters.** As a corollary, we obtain a protocol for estimating the heavy hitters in distributed streams with the best known message complexity. In this problem we would like to find a set $H$ of items so that if an element $e$ occurs at least an $\varepsilon$ fraction of times in the union of the streams, then $e \in H$, and if $e$ occurs less than an $\varepsilon/2$ fraction of times in union of the streams, then $e \notin H$. It is known that $O(\varepsilon^{-2}\log n)$ random samples suffice to estimate the set of heavy hitters with high probability, and the previous best algorithm [9] was obtained by plugging $s = O(\varepsilon^{-2}\log n)$ into a protocol for distributed sampling. We thus improve the message complexity from $O((k + \varepsilon^{-2}\log n)\log n)$ to

$O\left(\frac{k\log(\varepsilon n)}{\log(\varepsilon k)} + \varepsilon^{-2}\log(\varepsilon n)\log n\right)$. This can be significant when $k$ is large compared to $1/\varepsilon$.

### 1.2 Related Work

In addition to work discussed above, other research in the continuous distributed streaming model includes estimating frequency moments and counting the number of distinct elements [7, 8], and estimating the entropy [2]. The reservoir sampling technique has been used extensively in large scale data mining applications, see for example [10, 16, 1]. Stream sampling under sliding windows has been considered in [6, 3]. Deterministic algorithms for heavy-hitters over distributed streams, and corresponding lower bounds were considered in [21].

Stream sampling under sliding windows over distributed streams has been considered in [9]. Their algorithm for sliding windows is already optimal upto lower-order additive terms (see Theorems 4.1 and 4.2 in [9]). Hence our improved results for the non-sliding window case do not translate into an improvement for the case of sliding windows.

A related model of distributed streams was considered in [11, 12]. In this model, the coordinator was not required to continuously maintain an estimate of the required aggregate, but when the query was posed to the coordinator, the sites would be contacted and the query result would be constructed. In their model, the coordinator could be said to be "reactive", whereas in the model considered in this paper, the coordinator is "pro-active".

**Roadmap:** We first present the model and problem definition in Section 2, and then the algorithm followed by a proof of correctness in Section 3. The analysis of message complexity and the lower bound are presented in Sections 4 and 5 respectively, followed by an algorithm for sampling with replacement in Section 6.

## 2 Model

Consider a system with $k$ different sites, numbered from 1 till $k$, each receiving a local stream of elements. Let $\mathcal{S}_i$ denote the stream observed at site $i$. There is one "coordinator" node, which is different from any of the sites. The coordinator does not observe a local stream, but all queries for a random sample arrive at the coordinator. Let $\mathcal{S} = \cup_{i=1}^{n} \mathcal{S}_i$ be the entire stream observed by the system, and let $n = |\mathcal{S}|$. The sample size $s$ is a parameter supplied to the coordinator and to the sites during initialization.

The task of the coordinator is to continuously maintain a random sample $\mathcal{P}$ of size $\min\{n, s\}$ consisting of elements chosen uniformly at random without replacement from $\mathcal{S}$. The cost of the protocol is the number of messages transmitted.

We assume a synchronous communication model, where the system progresses in "rounds". In each round, each site can observe one element (or none), and send a message to the coordinator, and receive a response from the coordinator. The coordinator may receive up to $k$ messages in a round, and respond to each of them in the same round. This model is essentially identical to the model assumed in previous work [9]. Later we discuss how to handle the case of a site observing multiple elements per round.

The sizes of the different local streams at the sites, their order of arrival, and the interleaving of the streams at different sites, can all be arbitrary. The algorithm cannot make any assumption about these.

## 3 Algorithm

The idea in the algorithm is as follows. Each site associates a random "weight" with each element that it receives. The coordinator then maintains the set $\mathcal{P}$ of $s$ elements with the minimum weights in the union of the streams at all times, and this is a random sample of $\mathcal{S}$. This idea is similar to the spirit in all centralized reservoir sampling algorithms. In a distributed setting, the interesting aspect is at what times do the sites communicate with the coordinator, and vice versa.

In our algorithm, the coordinator maintains $u$, which is the $s$-th smallest weight so far in the system, as well as the sample $\mathcal{P}$, consisting of all the elements that have weight no more than $u$. Each site need only maintain a single value $u_i$, which is the site's view of the $s$-th smallest weight in the system so far. Note that it is too expensive to keep the view of each site synchronized with the coordinator's view at all times – to see this, note that the value of the $s$-th smallest weight changes $O(s \log(n/s))$ times, and updating every site each time the $s$-th minimum changes takes a total of $O(sk \log(n/s))$ messages.

In our algorithm, when site $i$ sees an element with a weight smaller than $u_i$, it sends it to the central coordinator. The coordinator updates $u$ and $\mathcal{P}$, if needed, and then replies back to $i$ with the current value of $u$, which is the true minimum weight in the union of all streams. Thus each time a site communicates with the coordinator, it either makes a change to the random sample, or, at least, gets to refresh its view of $u$.

The algorithm at each site is described in Algorithms 1 and 2. The algorithm at the coordinator is described in Algorithm 3.

**Algorithm 1**: Initialization at Site $i$.

```
/* $u_i$ is site $i$'s view of the $s$-th smallest weight in the
   union of all streams so far. Note this may ``lag'' the
   value stored at the coordinator.                          */
$u_i \leftarrow 1$;
```

**Algorithm 2**: When Site $i$ receives element $e$.

```
Let $w(e)$ be a randomly chosen weight between 0 and 1;
if $w(e) < u_i$ then
    Send $(e, w(e))$ to the Coordinator and receive $u'$ from Coordinator;
    Set $u_i \leftarrow u'$;
```

**Algorithm 3**: Algorithm at Coordinator.

```
/* The random sample $\mathcal{P}$ consists of tuples $(e, w)$ where $e$ is an
   element, and $w$ the weight, such that the weights are the $s$
   smallest among all the weights so far in the stream      */
$\mathcal{P} \leftarrow \phi$;
/* $u$ is the value of the $s$-th smallest weight in the stream
   observed so far. If there are less than $s$ elements so far,
   then $u$ is 1.                                           */
$u \leftarrow 1$;
while true do
    if a message $(e_i, u_i)$ arrives from site $i$ then
        if $u_i < u$ then
            Insert $(e_i, u_i)$ into $\mathcal{P}$;
            if $|\mathcal{P}| > s$ then
                Discard the element $(e, w)$ from $\mathcal{P}$ with the largest weight;
                Update $u$ to the current largest weight in $\mathcal{P}$ (which is also
                the $s$-th smallest weight in the entire stream);
        Send $u$ to site $i$;
    if a query for a random sample arrives then
        return $\mathcal{P}$
```

### 3.1 Correctness

The following two lemmas establish the correctness of the algorithm.

**Lemma 1.** *Let $n$ be the number of elements in $\mathcal{S}$ so far. (1) If $n \leq s$, then the set $\mathcal{P}$ at the coordinator contains all the $(e, w)$ pairs seen at all the sites so far. (2) If $n > s$, then $\mathcal{P}$ at the coordinator consists of the $s$ $(e, w)$ pairs such that the weights of the pairs in $\mathcal{P}$ are the smallest weights in the stream so far.*

The proof of this lemma has been omitted due to space constraints.

**Lemma 2.** *At the end of each round, sample $\mathcal{P}$ at the coordinator consists of a uniform random sample of size $\min\{n, s\}$ chosen without replacement from $\mathcal{S}$.*

*Proof.* In case $n \leq s$, then from Lemma 1, we know that $\mathcal{P}$ contains every element of $\mathcal{S}$. In case $n > s$, from Lemma 1, it follows that $\mathcal{P}$ consists of $s$ elements with the smallest weights from $\mathcal{S}$. Since the weights are assigned randomly, each element in $\mathcal{S}$ has a probability of $\frac{s}{n}$ of belonging in $\mathcal{P}$, showing that this is an uniform random sample. Since an element can appear no more than once in the sample, this is a sample chosen without replacement. $\square$

## 4 Analysis of the Algorithm (Upper Bound)

We now analyze the message complexity of the maintenance of a random sample.

For the sake of analysis, we divide the execution of the algorithm into "epochs", where each epoch consists of a sequence of rounds. The epochs are defined inductively. Let $r > 1$ be a parameter, which will be fixed later. Recall that $u$ is the $s$-th smallest weight so far in the system (if there are fewer than $s$ elements so far, $u = 1$). Epoch 0 is the set of all rounds from the beginning of execution until (and including) the earliest round where $u$ is $\frac{1}{r}$ or smaller. Let $m_i$ denote the value of $u$ at the end of epoch $i-1$. Then epoch $i$ consists of all rounds subsequent to epoch $i-1$ until (and including) the earliest round when $u$ is $\frac{m_i}{r}$ or smaller. Note that the algorithm does not need to be aware of the epochs, and this is only used for the analysis.

Suppose we call the original distributed algorithm described in Algorithms 3 and 2 as Algorithm $A$. For the analysis, we consider a slightly different distributed algorithm, Algorithm $B$, described below. *Algorithm $B$ is identical to Algorithm $A$ except for the fact that at the beginning of each epoch, the value $u$ is broadcast by the coordinator to all sites.*

While Algorithm $A$ is natural, Algorithm $B$ is easier to analyze. We first note that on the same inputs, the value of $u$ (and $\mathcal{P}$) at the coordinator at any round in Algorithm $B$ is identical to the value of $u$ (and $\mathcal{P}$) at the coordinator in Algorithm $A$ at the same round. Hence, the partitioning of rounds into epochs is the same for both algorithms, for a given input. The correctness of Algorithm $B$ follows from the correctness of Algorithm $A$. The only difference between them is in the total number of messages sent. In $B$ we have the property that for all $i$ from 1 to $k$, $u_i = u$ at the beginning of each epoch (though this is not necessarily true throughout the epoch), and for this, $B$ has to pay a cost of at least $k$ messages in each epoch.

**Lemma 3.** *The number of messages sent by Algorithm $A$ for a set of input streams $\mathcal{S}_j, j = 1 \ldots k$ is never more than twice the number of messages sent by Algorithm $B$ for the same input.*

*Proof.* Consider site $v$ in a particular epoch $i$. In Algorithm $B$, $v$ receives $m_i$ at the beginning of the epoch through a message from the coordinator. In Algorithm $A$, $v$ may not know $m_i$ at the beginning of epoch $i$. We consider two cases.

Case I: $v$ sends a message to the coordinator in epoch $i$ in Algorithm $A$. In this case, the first time $v$ sends a message to the coordinator in this epoch, $v$ will receive the current value of $u$, which is smaller than or equal to $m_i$. This communication costs two messages, one in each direction. Henceforth, in this epoch, the number of messages sent in Algorithm $A$ is no more than those sent in $B$. In this epoch, the number of messages transmitted to/from $v$ in $A$ is at most twice the number of messages as in $B$, which has at least one transmission from the coordinator to site $v$.

Case II: $v$ did not send a message to the coordinator in this epoch, in Algorithm $A$. In this case, the number of messages sent in this epoch to/from site $v$ in Algorithm $A$ is smaller than in Algorithm $B$. $\square$

Let $\xi$ denote the total number of epochs.

**Lemma 4.** *If $r \geq 2$,*

$$E[\xi] \leq \left(\frac{\log(n/s)}{\log r}\right) + 2$$

*Proof.* Let $z = \left(\frac{\log(n/s)}{\log r}\right)$. First, we note that in each epoch, $u$ decreases by a factor of at least $r$. Thus after $(z+\ell)$ epochs, $u$ is no more than $\frac{1}{r^{z+\ell}} = (\frac{s}{n})\frac{1}{r^\ell}$. Thus, we have

$$\Pr[\xi \geq z+\ell] \leq \Pr\left[u \leq \left(\frac{s}{n}\right)\frac{1}{r^\ell}\right]$$

Let $Y$ denote the number of elements (out of $n$) that have been assigned a weight of $\frac{s}{nr^\ell}$ or lesser. $Y$ is a binomial random variable with expectation $\frac{s}{r^\ell}$. Note that if $u \leq \frac{s}{nr^\ell}$, it must be true that $Y \geq s$.

$$\Pr[\xi \geq z+\ell] \leq \Pr[Y \geq s] \leq \Pr[Y \geq r^\ell E[Y]] \leq \frac{1}{r^\ell}$$

where we have used Markov's inequality.

Since $\xi$ takes only positive integral values,

$$\begin{aligned} E[\xi] = \sum_{i>0}\Pr[\xi \geq i] &= \sum_{i=1}^{z}\Pr[\xi \geq i] + \sum_{\ell\geq 1}\Pr[\xi \geq z+\ell] \\ &\leq z + \sum_{\ell \geq 1}\frac{1}{r^\ell} \leq z + \frac{1}{1-1/r} \leq z+2 \end{aligned}$$

where we have assumed $r \geq 2$. □

Let $n_j$ denote the total number of elements that arrived in epoch $j$. We have $n = \sum_{j=0}^{\xi-1} n_j$. Let $\mu$ denote the total number of messages sent during the entire execution. Let $\mu_i$ denote the total number of messages sent in epoch $i$. Let $X_i$ denote the number of messages sent from the sites to the coordinator in epoch $i$. $\mu_i$ is the sum of two parts, (1) $k$ messages sent by the coordinator at the start of the epoch, and (2) two times the number of messages sent from the sites to the coordinator.

$$\mu_i = k + 2X_i \tag{1}$$

$$\mu = \sum_{j=0}^{\xi-1}\mu_i = \xi k + 2\sum_{j=0}^{\xi-1}X_j \tag{2}$$

Consider epoch $i$. For each each element $j = 1 \ldots n_i$ in epoch $i$, we define a 0-1 random variable $Y_j$ as follows. $Y_j = 1$ if observing the $j$-th element in the epoch resulted in a message being sent to the coordinator, and $Y_j = 0$ otherwise.

$$X_i = \sum_{j=1}^{n_i} Y_j \tag{3}$$

Let $F(\eta, \alpha)$ denote the event $n_i = \eta$ and $m_i = \alpha$. The following Lemma gives a bound on a conditional probability that is used later.

**Lemma 5.** *For each* $j = 1 \ldots n_i - 1$

$$\Pr[Y_j = 1 | F(\eta, \alpha)] \le \frac{\alpha - \alpha/r}{1 - \alpha/r}$$

*Proof.* Suppose that the $j$-th element in the epoch was observed by site $v$. For this element to cause a message to be sent to the coordinator, the random weight assigned to it must be less than $u_v$ at that instant. Conditioned on $m_i = \alpha$, $u_v$ is no more than $\alpha$.

Note that in this lemma we exclude the last element that arrived in epoch $i$, thus the weight assigned to element $j$ must be greater than $\alpha/r$. Thus, the weight assigned to $j$ must be a uniform random number in the range $(\alpha/r, 1)$. The probability this weight is less than the current value of $u_v$ is no more than $\frac{\alpha-\alpha/r}{1-\alpha/r}$, since $u_v \le \alpha$.

□

**Lemma 6.** *For each epoch* $i$

$$E[X_i] \le 1 + 2rs$$

*Proof.* We first obtain the expectation conditioned on $F(\eta, \alpha)$, and then remove the conditioning. From Lemma 5 and Equation 3 we get:

$$E[X_i | F(\eta, \alpha)] \le 1 + E\left[ (\sum_{j=1}^{\eta-1} Y_j) | F(\eta, \alpha) \right] \le 1 + \sum_{j=1}^{\eta-1} E[Y_j | F(\eta, \alpha)] \le 1 + (\eta - 1)\frac{\alpha - \alpha/r}{1 - \alpha/r}.$$

Using $r \ge 2$ and $\alpha \le 1$, we get: $E[X_i | F(\eta, \alpha)] \le 1 + 2(\eta - 1)\alpha$.

We next consider the conditional expectation $E[X_i | m_i = \alpha]$.

$$\begin{aligned} E[X_i | m_i = \alpha] &= \sum_{\eta} \Pr[n_i = \eta | m_i = \alpha] E[X_i | n_i = \eta, m_i = \alpha] \\ &\le \sum_{\eta} \Pr[n_i = \eta | m_i = \alpha](1 + 2(\eta - 1)\alpha) \\ &\le E[1 + 2(n_i - 1)\alpha | m_i = \alpha] \\ &\le 1 + 2\alpha(E[n_i | m_i = \alpha] - 1) \end{aligned}$$

Using Lemma 7, we get

$$E[X_i | m_i = \alpha] \le 1 + 2\alpha\left(\frac{rs}{\alpha} - 1\right) \le 1 + 2rs$$

Since $E[X_i] = E[E[X_i | m_i = \alpha]]$, we have $E[X_i] \le E[1 + 2rs] = 1 + 2rs$. □

**Lemma 7.**
$$E[n_i | m_i = \alpha] = \frac{rs}{\alpha}$$

*Proof.* Recall that $n_i$, the total number of elements in epoch $i$, is the number of elements observed till the $s$-th minimum in the stream decreases to a value that is less than or equal to $\alpha/r$.

Let $Z$ denote a random variable that equals the number of elements to be observed from the start of epoch $i$ till $s$ new elements are seen, each of whose weight is less than or equal to $\alpha/r$. Clearly, conditioned on $m_i = \alpha$, it must be true that $n_i \leq Z$. For $j = 1$ to $s$, let $Z_j$ denote the number of elements observed from the state when $(j-1)$ elements have been observed with weights that are less than $\alpha/r$ till the state when $j$ elements have been observed with weights less than $\alpha/r$. $Z_j$ is a geometric random variable with parameter $\alpha/r$.

We have $Z = \sum_{j=1}^{s} Z_j$ and $E[Z] = \sum_{j=1}^{s} E[Z_j] = \frac{sr}{\alpha}$. Since $E[n_i | m_i = \alpha] \leq E[Z]$, the lemma follows. □

**Lemma 8.**
$$E[\mu] \leq (k + 4rs + 2)\left(\frac{\log(n/s)}{\log r} + 2\right)$$

*Proof.* Using Lemma 6 and Equation 1, we get the expected number of messages in epoch $i$:
$$E[\mu_i] \leq k + 2(2rs + 1) = k + 2 + 4rs$$

Note that the above is independent of $i$. The proof follows from Lemma 4, which gives an upper bound on the expected number of epochs. □

**Theorem 1.** *The expected message complexity $E[\mu]$ of our algorithm is as follows.*

*I: If $s \geq \frac{k}{8}$, then $E[\mu] = O\left(s \log\left(\frac{n}{s}\right)\right)$*
*II: If $s < \frac{k}{8}$, then $E[\mu] = O\left(\frac{k \log\left(\frac{n}{s}\right)}{\log\left(\frac{k}{s}\right)}\right)$*

*Proof.* We note that the upper bounds on $E[\mu]$ in Lemma 8 hold for any value of $r \geq 2$.

Case I: $s \geq \frac{k}{8}$. In this case, we set $r = 2$. From Lemma 8,

$$E[\mu] \leq (8s + 8s + 2)\left(\frac{\log(n/s)}{\log 2}\right) = (16s + 2)\log\left(\frac{n}{s}\right) = O\left(s \log\left(\frac{n}{s}\right)\right)$$

Case II: $s < \frac{k}{8}$. We minimize the expression setting $r = \frac{k}{4s}$, and get: $E[\mu] = O\left(\frac{k \log\left(\frac{n}{s}\right)}{\log\left(\frac{k}{s}\right)}\right)$. □

## 5 Lower Bound

**Theorem 2.** *For any constant $q, 0 < q < 1$, any correct protocol must send $\Omega\left(\frac{k\log(n/s)}{\log(1+(k/s))}\right)$ messages with probability at least $1-q$, where the probability is taken over the protocol's internal randomness.*

*Proof.* Let $\beta = (1+(k/s))$. Define $e = \Theta\left(\frac{\log(n/s)}{\log(1+(k/s))}\right)$ epochs as follows: in the $i$-th epoch, $i \in \{0,1,2,\ldots,e-1\}$, there are $\beta^{i-1}k$ global stream updates, which can be distributed among the $k$ servers in an arbitrary way.

We consider a distribution on orderings of the stream updates. Namely, we think of a totally-ordered stream $1,2,3,\ldots,n$ of $n$ updates, and in the $i$-th epoch, we randomly assign the $\beta^{i-1}k$ updates among the $k$ servers, independently for each epoch. Let the randomness used for the assignment in the $i$-th epoch be denoted $\sigma_i$.

Consider the global stream of updates $1,2,3,\ldots,n$. Suppose we maintain a sample set $\mathcal{P}$ of $s$ items without replacement. We let $\mathcal{P}_i$ denote a random variable indicating the value of $\mathcal{P}$ after seeing $i$ updates in the stream. We will use the following lemma about reservoir sampling.

**Lemma 9.** *For any constant $q > 0$, there is a constant $C' = C'(q) > 0$ for which*

- *$\mathcal{P}$ changes at least $C's\log(n/s)$ times with probability at least $1-q$, and*
- *If $s < k/8$ and $k = \omega(1)$ and $e = \omega(1)$, then with probability at least $1-q/2$, over the choice of $\{\mathcal{P}_i\}$, there are at least $(1-(q/8))e$ epochs for which the number of times $\mathcal{P}$ changes in the epoch is at least $C's\log(1+(k/s))$.*

*Proof.* Consider the stream $1,2,3,\ldots,n$ of updates. In the classical reservoir sampling algorithm [15], $\mathcal{P}$ is initialized to $\{1,2,3,\ldots,s\}$. Then, for each $i > s$, the $i$-th element is included in the current sample set $\mathcal{P}_i$ with probability $s/i$, in which case a random item in $\mathcal{P}_{i-1}$ is replaced with $i$.

For the first part of Lemma 9, let $X_i$ be an indicator random variable if $i$ causes $\mathcal{P}$ to change. Let $X = \sum_{i=1}^{n} X_i$. Hence, $\mathbf{E}[X_i] = s/i$ for all $i$, and $\mathbf{E}[X] = H_n - H_s$, where $H_i = \ln i + O(1)$ is the $i$-th Harmonic number. Then all of the $X_i$, $i > s$ are independent indicator random variables. It follows by a Chernoff bound that

$$\Pr[X < \mathbf{E}[X]/2] \le \exp(\mathbf{E}[X]/8) \le \exp(-(\ln n/s)/8) \le \left(\frac{s}{n}\right)^{1/8}.$$

For any $s = o(n)$, this is less than any constant $q$, and so the first part of Lemma 9 follows since $\mathbf{E}[X]/2 = 1/2 \cdot \ln(n/s)$.

For the second part of Lemma 9, consider the $i$-th epoch, $i > 0$, which contains $\beta^{i-1}k$ consecutive updates. Let $Y_i$ be the number of changes in this epoch. Then $\mathbf{E}[Y_i] = s\ln\beta + O(1)$. Since $Y_i$ can be written as a sum of independent indicator random variables, by a Chernoff bound,

$$\Pr[Y_i < \mathbf{E}[Y_i]/2] \le \exp(-\mathbf{E}[Y_i]/8) \le \exp(-(s\ln\beta + O(1))/8) \le \frac{1}{\beta^{s/8}}.$$

Hence, the expected number of epochs $i$ for which $Y_i < \mathbf{E}[Y_i]/2$ is at most $\sum_{i=1}^{e-1} \frac{1}{\beta^{s/8}}$, which is $o(e)$ since we're promised that $s < k/8$ and $k = \omega(1)$ and $e = \omega(1)$. By a Markov bound, with probability at least $1 - q/2$, at most $o(e/q) = o(e)$ epochs $i$ satisfy $Y_i \geq \mathbf{E}[Y_i]/2$. It follows that with probability at least $1 - q/2$, there are at least $(1 - q/8)e$ epochs $i$ for which the number $Y_i$ of changes in the epoch $i$ is at least $\mathbf{E}[Y_i]/2 \geq \frac{1}{2} s \ln \beta$, as desired.

$\square$

**Corner Cases:** When $s \geq k/8$, the statement of Theorem 2 gives a lower bound of $\Omega(s \log(n/s))$. In this case Theorem 2 follows immediately from the first part of Lemma 9 since these changes in $\mathcal{P}$ must be communicated to the central coordinator. Hence, in what follows we can assume $s < k/8$. Notice also that if $k = O(1)$, then $\frac{k \log(n/s)}{\log(1+(k/s))} = O(s \log(n/s))$, and so the theorem is independent of $k$, and follows simply by the first part of Lemma 9. Notice also that if $e = O(1)$, then the statement of Theorem 2 amounts to proving an $\Omega(k)$ lower bound, which follows trivially since every site must send at least one message.

Thus, in what follows, we may apply the second part of Lemma 9.

**Main Case:** Let $C > 0$ be a sufficiently small constant, depending on $q$, to be determined below. Let $\Pi$ be a possibly randomized protocol, which with probability at least $q$, sends at most $Cke$ messages. We show that $\Pi$ cannot be a correct protocol.

Let $\tau$ denote the random coin tosses of $\Pi$, i.e., the concatenation of random strings of all $k$ sites together with that of the central coordinator.

Let $\mathcal{E}$ be the event that $\Pi$ sends less than $Cke$ messages. By assumption, $\Pr_\tau[\mathcal{E}] \geq q$. Hence, it is also the case that

$$\Pr_{\tau, \{\mathcal{P}_i\}, \{\sigma_i\}}[\mathcal{E}] \geq q.$$

For a sufficiently small constant $C' > 0$ that may depend on $q$, let $\mathcal{F}$ be the event that there are at least $(1 - (q/8))e$ epochs for which the number of times $\mathcal{P}$ changes in the epoch is at least $C' s \log(1 + (k/s))$. By the second part of Lemma 9,

$$\Pr_{\tau, \{\mathcal{P}_i\}, \{\sigma_i\}}[\mathcal{F}] \geq 1 - q/2.$$

It follows that there is a fixing of $\tau = \tau'$ as well as a fixing of $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_e$ to $P'_0, P'_1, \ldots, P'_e$ for which $\mathcal{F}$ occurs and

$$\Pr_{\{\sigma_i\}}[\mathcal{E} \mid \tau = \tau',\ (\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_e) = (P'_0, P'_1, \ldots, P'_e)] \geq q - q/2 = q/2.$$

Notice that the three (sets of) random variables $\tau, \{P_i\}$, and $\{\sigma_i\}$ are independent, and so in particular, $\{\sigma_i\}$ is still uniformly random given this conditioning.

By a Markov argument, if event $\mathcal{E}$ occurs, then there are at least $(1-(q/8))e$ epochs for which at most $(8/q) \cdot C \cdot k$ messages are sent. If events $\mathcal{E}$ and $\mathcal{F}$

both occur, then by a union bound, there are at least $(1-(q/4))e$ epochs for which at most $(8/q)\cdot C\cdot k$ messages are sent and $S$ changes in the epoch at least $C's\log(1+(k/s))$ times. Call such an epoch *balanced*.

Let $i^*$ be the epoch which is most likely to be balanced, over the random choices of $\{\sigma_i\}$, conditioned on $\tau=\tau'$ and $(\mathcal{P}_0,\mathcal{P}_1,\ldots,\mathcal{P}_e)=(P'_0,P'_1,\ldots,P'_e)$. Since at least $(1-(q/4))e$ epochs are balanced if $\mathcal{E}$ and $\mathcal{F}$ occur, and conditioned on $(\mathcal{P}_0,\mathcal{P}_1,\ldots,\mathcal{P}_e)=(P'_0,P'_1,\ldots,P'_e)$ event $\mathcal{F}$ does occur, and $\mathcal{E}$ occurs with probability at least $q/2$ given this conditioning, it follows that

$$\Pr_{\{\sigma_i\}}[i^* \text{ is balanced } \mid \tau=\tau',\ (\mathcal{P}_0,\mathcal{P}_1,\ldots,\mathcal{P}_e)=(P'_0,P'_1,\ldots,P'_e)]\geq q/2-q/4=q/4.$$

The property of $i^*$ being balanced is independent of $\sigma_j$ for $j\neq i^*$, so we also have

$$\Pr_{\sigma_{i^*}}[i^* \text{ is balanced } \mid \tau=\tau',\ (\mathcal{P}_0,\mathcal{P}_1,\ldots,\mathcal{P}_e)=(P'_0,P'_1,\ldots,P'_e)]\geq q/4.$$

If $C's\log(1+(k/s))\geq 1$, then $\mathcal{P}$ changes at least once in epoch $i^*$. Suppose, for the moment, that this is the case. Suppose the first update in the global stream at which $\mathcal{P}$ changes is the $j^*$-th update. In order for $i^*$ to be balanced for at least a $q/4$ fraction of the $\sigma_{i^*}$, there must be at least $qk/4$ different servers which receive $j^*$, for which $\varPi$ sends a message. In particular, since $\varPi$ is deterministic conditioned on $\tau$, at least $qk/4$ messages must be sent in the $i^*$-th epoch. But $i^*$ was chosen so that at most $(8/q)\cdot C\cdot k$ messages are sent, which is a contradiction for $C<q^2/32$.

It follows that we reach a contradiction unless $C's\log(1+(k/s))<1$. Notice, though, that since $C'$ is a constant, if $C's\log(1+(k/s))<1$, then this implies that $k=O(1)$. However, if $k=O(1)$, then $\frac{k\log(n/s)}{\log(1+(k/s))}=O(s\log(n/s))$, and so the theorem is independent of $k$, and follows simply by the first part of Lemma 9.

Otherwise, we have reached a contradiction, and so it follows that $Cke$ messages must be sent with probability at least $1-q$. Since $Cke=\varOmega\left(\frac{k\log(n/s)}{\log(1+(k/s))}\right)$, this completes the proof. □

## 6 Sampling With Replacement

We now present an algorithm to maintain a random sample of size $s$ with replacement from $\mathcal{S}$. The basic idea is to run in parallel $s$ copies of the single item sampling algorithm from Section 3. Done naively, this will lead to a message complexity of $O(sk\frac{\log n}{\log k})$. We obtain an improved algorithm based on the following ideas.

We view the distributed streams as $s$ logical streams, $\mathcal{S}^i, i=1\ldots s$. Each $\mathcal{S}^i$ is identical to $\mathcal{S}$, but the algorithm assigns independent weights to the different copies of the same element in the different logical streams. Let $w^i(e)$ denote the weight assigned to element $e$ in $\mathcal{S}^i$. $w^i(e)$ is a random number between 0 and

1. For each $i = 1 \dots s$, the coordinator maintains the minimum weight, say $w^i$, among all elements in $\mathcal{S}^i$, and the corresponding element.

Let $\beta = \max_{i=1}^{s} w^i$; $\beta$ is maintained by the coordinator. Each site $j$ maintains $\beta_j$, a local view of $\beta$, which is always greater than or equal to $\beta$. Whenever a logical stream element at site $j$ has weight less than $\beta_j$, the site sends it to the coordinator, receives in response the current value of $\beta$, and updates $\beta_j$. When a random sample is requested at the coordinator, it returns the set of all minimum weight elements in all $s$ logical streams. It can be easily seen that this algorithm is correct, and at all times, returns a random sample of size $s$ selected with replacement. The main optimization relative to the naive approach described above is that when a site sends a message to the coordinator, it receives $\beta$, which provides partial information about all $w^i$s. This provides a substantial improvement in the message complexity and leads to the following bounds.

**Theorem 3.** *The above algorithm continuously maintains a sample of size $s$ with replacement from $\mathcal{S}$, and its expected message complexity is $O(s \log s \log n)$ in case $k \leq 2s \log s$, and $O\left(k \frac{\log n}{\log(\frac{k}{s \log s})}\right)$ in case $k > 2s \log s$.*

*Proof.* We provide a sketch of the proof here. The analysis of the message complexity is similar to the case of sampling without replacement. We sketch the analysis here, and omit the details. The execution is divided into epochs, where in epoch $i$, the value of $\beta$ at the coordinator decreases by at least a factor of $r$ (a parameter to be determined later). Let $\xi$ denote the number of epochs. It can be seen that $E[\xi] = O(\frac{\log n}{\log r})$. In epoch $i$, let $X_i$ denote the number of messages sent from the sites to the coordinator in the epoch, $m_i$ denote the value of $\beta$ at the beginning of the epoch, and $n_i$ denote the number of elements in $\mathcal{S}$ that arrived in the epoch.

The $n_i$ elements in epoch $i$ give rise to $sn_i$ logical elements, and each logical element has a probability of no more than $m_i$ of resulting in a message to the coordinator. Similar to the proof of Lemma 6, we can show using conditional expectations that $E[X_i] \leq rs \log s$ (the $\log s$ factor comes in due to the fact that $E[n_i | m_i = \alpha] \leq \frac{r \log s}{\alpha}$. Thus the expected total number of messages in epoch $i$ is bounded by $(k + 2sr \log s)$, and in the entire execution is $O((k + 2sr \log s)\frac{\log n}{\log r})$. By choosing $r = 2$ for the case $k \leq (2s \log s)$, and $r = k/(s \log s)$ for the case $k > (2s \log s)$, we get the desired result. □

## References

1. C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *VLDB*, pages 607–618, 2006.
2. C. Arackaparambil, J. Brody, and A. Chakrabarti. Functional monitoring without monotonicity. In *ICALP (1)*, pages 95–106, 2009.
3. B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634, 2002.
4. B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD Conference*, pages 28–39, 2003.

5. V. Braverman and R. Ostrovsky. Effective computations on sliding windows. *SIAM Journal on Computing.*, 39(6):2113–2131, 2010.
6. V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *PODS*, pages 147–156, 2009.
7. G. Cormode and M. N. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24, 2005.
8. G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In *SODA*, pages 1076–1085, 2008.
9. G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *PODS*, pages 77–86, 2010.
10. M. Dash and W. Ng. Efficient reservoir sampling for transactional data streams. In *Sixth IEEE International Conference on Data Mining (Workshops)*, pages 662 –666, 2006.
11. P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *SPAA*, pages 281–291, 2001.
12. P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, pages 63–72, 2002.
13. L. Huang, X. Nguyen, M. N. Garofalakis, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM*, pages 134–142, 2007.
14. R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD Conference*, pages 289–300, 2006.
15. D. E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition.* Addison-Wesley, 1981.
16. V. Malbasa and S. Vucetic. A reservoir sampling algorithm with adaptive estimation of conditional expectation. In *IJCNN 2007, International Joint Conference on Neural Networks*, pages 2200 –2204, 2007.
17. A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, pages 767–778, 2005.
18. S. Muthukrishnan. *Data Streams: Algorithms and Applications.* Foundations and Trends in Theoretical Computer Science. Now Publishers, August 2005.
19. J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.
20. B. Xu, S. Tirthapura, and C. Busch. Sketching asynchronous data streams over sliding windows. *Distributed Computing*, 20(5):359–374, 2008.
21. K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *PODS*, pages 167–174, 2009.