6-1-2016

A Simple Message-Optimal Algorithm for Random Sampling from a Distributed Stream

Yung-Yu Chung
Iowa State University

Srikanta Tirthapura
Iowa State University, snt@iastate.edu

IBM Almaden Research Center

A Simple Message-Optimal Algorithm for Random Sampling from a Distributed Stream

Abstract

We present a simple, message-optimal algorithm for maintaining a random sample from a large data stream whose input elements are distributed across multiple sites that 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. We present a matching lower bound, showing that our protocol sends the optimal number of messages up to a constant factor with large probability. We also consider the important case when the distribution of elements across different sites is non-uniform, and show that for such inputs, our algorithm significantly outperforms prior solutions.

Keywords

Complexity theory, Algorithm design and analysis, Protocols, reservoirs, Silicon, Distributed databases, Electronic mail

Disciplines

Electrical and Computer Engineering

Comments

This is a manuscript of an article published as Chung, Yung-Yu, Srikanta Tirthapura, and David P. Woodruff. "A Simple Message-Optimal Algorithm for Random Sampling from a Distributed Stream." IEEE Transactions on Knowledge and Data Engineering 28, no. 6 (2016): 1356-1368. 10.1109/TKDE.2016.2518679. Posted with permission.

Rights

A Simple Message-Optimal Algorithm for Random Sampling from a Distributed Stream

Yung-Yu Chung, Srikanta Tirthapura, and David P. Woodruff

Abstract—We present a simple, message-optimal algorithm for maintaining a random sample from a large data stream whose input elements are distributed across multiple sites that 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. We present a matching lower bound, showing that our protocol sends the optimal number of messages up to a constant factor with large probability. We also consider the important case when the distribution of elements across different sites is non-uniform, and show that for such inputs, our algorithm significantly outperforms prior solutions.

Index Terms—Distributed stream, random sampling, reservoir sampling, skew

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* [1]. In this model there are multiple 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 approximate 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., [2], where the algorithm is attributed to Waterman; see also [3]) 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 [4]. Other problems on distributed stream processing, including the estimation of the number of distinct elements [1], [5] and heavy hitters [6], [7], [8], [9], use random sampling as a primitive (we note, though, that better solutions for the heavy hitters problem in terms of the accuracy parameter may be possible [9] than those provided by random sampling). Distributed random sampling is already used in current day "big data" systems such as BlinkDB [10], which use stored random samples to process queries quickly, in exchange for relaxed accuracy guarantees. These systems operate on tens of terabytes of data, spread over hundreds of machines, and have shown dramatic speedups for common aggregate queries, using sampling.

The distributed random sampling problem that we consider is as follows. There are k distributed sites, numbered 1 through k, in addition to a coordinator. Each site i observes its own local stream $\mathcal{S}_i$. The task is for the coordinator to continuously maintain a random sample of size s from the union of all streams $\cup_{i=1}^{k}\mathcal{S}_i$. The requirement is to minimize the number of messages sent between the sites and the coordinator.

1.1 Our Results

Our main contributions are a simple algorithm for sampling without replacement from a distributed stream, as well as a matching lower bound showing that the message complexity of our algorithm is optimal. The message complexity is the number of message transmission between sites and the coordinator. The algorithm is easy to implement, and as our experiments show, has very good practical performance.

Algorithm. We present an algorithm that 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. Note 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 at the coordinator is s words. The remote sites in our scheme store a single machine word and use constant time per stream update, both of which are clearly optimal.

Lower bound. We also show that any correct protocol which succeeds with constant probability must send $\Omega\left(\frac{k\log(n/s)}{\log(1+(k/s))}\right)$ messages with constant probability. 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

- *Y.-Y. Chung and S. Tirthapura are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011. E-mail: {ychung, snt}@iastate.edu.*
- *D.P. Woodruff is with the IBM Almaden Research Center, San Jose, CA 95120-6099. E-mail: dpwoodru@us.ibm.com.*

Manuscript received 30 Dec. 2014; revised 15 Dec. 2015; accepted 1 Jan. 2016. Date of publication 18 Jan. 2016; date of current version 27 Apr. 2016.
Recommended for acceptance by A. Banerjee.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2518679

correct protocol, showing that the expected number of messages sent by our algorithm is optimal, up to constant factors.

Impact of skew. The algorithm and lower bound as discussed do not assume any distribution on the numbers of elements that arrive at the different sites. The worst case message complexity arises in the setting where different sites receive nearly the same number of elements. In a practical setting, however, it is common to have a skewed distribution of the arrivals at different local streams. For example, traffic sensors posted on streets observing cars pass by may see different volumes of activity depending on how busy the streets are. It is important to consider the performance in the skewed case when the different streams $\mathcal{S}_i$ are of different sizes.

To measure the performance of our algorithm in the presence of skew, we considered the following model of data arrival. Suppose a set of probabilities $p_i, i = 1 \ldots k$, one per site, such that $\sum_{i=1}^{k} p_i = 1$. Each arrival into the system is sent to stream $\mathcal{S}_i$ with a probability of p_i. The p_is need not be equal, so that the numbers of elements seen by different sites can be significantly different from each other.

We show that under such a model, our algorithm has a message complexity of $O(s \log(n/s) \sum_{i, p_i \neq 0} \frac{1}{\log(\frac{2}{p_i})})$. Note that the algorithm remains the same as in the worst case analysis, but the message complexity is different, due to the assumed input model.

The message complexity under skew can be much smaller than the upper bound for general case. For example, suppose there were only five sites receiving data, and the rest did not receive any data. Further, suppose we want a sample of size 1. We have $p_i = 0.2$ for $i = 1 \ldots 5$, and $p_i = 0, i > 5$. In such a case the message complexity of our algorithm is only $O(\log n)$. In contrast, the worst case upper bound for a sample size of 1 is $O(\frac{k \log n}{\log k})$.

In another case, suppose that $p_1 = 1 - c(k-1)/n$, and $p_i = c/n$ for $i \geq 2$, where c is a constant independent of n. This models the case when one site receives a majority of elements, and each of the other sites receives only about c elements. In such a case, the message complexity of our algorithm is $O(k + \log n)$, to be contrasted with the general bound of $O(\frac{k\log n}{\log k})$.

Experimental evaluation. We conducted an experimental study evaluating the message cost of our algorithm, comparing with the cost of the algorithm from [11]. We observe that the number of messages in our algorithm has a linear dependence on the number of sites and sample size. In general, our algorithm achieves better performance than the algorithm from [11]. The benefit is especially clear when the required sample size is high. In the presence of skew in data, our algorithm shows significantly improved performance when compared with the case of no skew.

Sampling with replacement. We also show how to modify our protocol to obtain a random sample of size s chosen with replacement. The message complexity of this protocol is $O((\frac{k}{\log(2 + (k/(s \log s)))} + s \log s) \log n)$ messages.

1.2 Comparison with Prior Work

The performance of our algorithm should be compared with the algorithm due to Cormode, Muthukrishnan, Yi, and Zhang [11], which is the current state of the art. Their algorithm, which we henceforth call "CMYZ", has an expected message complexity of $O(k \log_{k/s} n + s \log n)$ where n is the total number of elements observed across all streams. The memory requirement of the central coordinator is s machine words. The message complexity of our algorithm improves upon the bounds of CMYZ, as shown in Table 1.

TABLE 1
Summary of Our Results for Message Complexity of Sampling without Replacement

	Upper Bound		Lower Bound
	Our Result	CMYZ [11]	
$s < \frac{k}{8}$	$O\left(\frac{k \log(n/s)}{\log(k/s)}\right)$	$O\left(\frac{k \log n}{\log(k/s)}\right)$	$\Omega\left(\frac{k \log(n/s)}{\log(k/s)}\right)$
$s \geq \frac{k}{8}$	$O(s \log(n/s))$	$O(s \log n)$	$\Omega(s \log(n/s))$

Our algorithm can be viewed as a refinement and polishing of the CMYZ algorithm. Both their algorithm and our algorithm are based on each site maintaining a threshold, assigning a random value to each input item, and comparing to a threshold to determine whether to forward the item to a central location. There are several important differences in the details of such an algorithm, which turn out to have a noticeable impact on the complexity as we show here.

In the presence of skew in data, our algorithm's performance improves significantly, as described above, but the performance of CMYZ remains unaffected. Thus, in the presence of skew, our algorithm will send significantly fewer messages. As an example, consider the extreme case when all elements are directed to a single site, no other site receives any element, and we are interested in a sample size of 1. The message complexity of our algorithm is $O(\log n)$, while that of CMYZ is $O(\frac{k \log n}{\log k})$, which could be substantially larger. In subsequent sections, we present an analysis of both algorithms in the presence of skew. The analysis shows that while the performance of our algorithm improves with increasing skew, the performance of CMYZ is unchanged.

One reason for this difference is the manner in which the coordinator communicates with the sites in the two algorithms. The CMYZ algorithm is based on repeated broadcasts from the coordinator to all sites, while our algorithm does not use a broadcast from the coordinator. Instead, all communication is initiated by a site, and the coordinator only responds back to the initiating site. In our algorithm, a site does not have to be able to receive broadcast messages. The ability not to have to receive broadcast messages is useful in a setting where a site may go offline.

1.3 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 [1], [5], and estimating the entropy [12]. Stream sampling has a long history of research, starting from the popular *reservoir sampling* algorithm, attributed to Waterman (see Algorithm R from [13]) that has been known since the 1960s. Follow-up work includes speeding up reservoir sampling [13], weighted reservoir sampling [14], sampling over a sliding window and stream evolution [15], [16], [17], [18],

[19]. Stream sampling has been used extensively in large scale data mining applications, see for example [20], [21], [22], [23]. Stream sampling under sliding windows has been considered in [17], [24]. Deterministic algorithms for finding the heavy-hitters in distributed streams, and corresponding lower bounds for this problem were considered in [9]. Stream sampling under sliding windows over distributed streams has been considered in [25]. Continuous random sampling from the set of distinct elements in a stream has been considered in [26]. The question of how to process a "sampled stream", i.e., once a stream has been sampled, is considered in [27].

A model of distributed streams related to ours was considered in [28], [29]. 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 present the model and problem definition in Section 2, and then the algorithm followed by a proof of correctness in Section 3. We then present the analysis of message complexity, the lower bound, the analysis under skew, followed by the algorithm for sampling with replacement in Sections 4, 5, 6, and 7 respectively. We finally present experimental results in Section 8.

2 Model

The coordinator node is assumed to be 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. It is straightforward to modify our algorithms for the case when the coordinator also observes a local stream. Let $\mathcal{S} = \cup_{i=1}^{n} \mathcal{S}_i$ be the entire stream observed by the system, and note that the total number of elements $n = |\mathcal{S}|$. The sample size s supplied to the coordinator and to the sites during initialization. The task of the coordinator is to continuously maintain a random sample s of size $\min\{n, s\}$ consisting of elements chosen uniformly at random without replacement from $\mathcal{S}$.

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 [25].

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. For instance, it is possible that a single site receives a large number of elements before a different site receives its first element. It is also possible that all sites receive elements streams that are of the same size and whose elements arrive in the same rounds. In Section 6, we analyze the performance of the algorithm under certain specific input distributions. However, the algorithm still remains the same, and does not depend on the input distribution.

Note that what matters for the algorithm is the global ordering of the stream items by their time of arrival onto one of the sites. The "speed" of the stream, i.e., how long it takes for the next item to arrive on a site does not play any role in the complexity of our algorithm, which is concerned with the total number of messages transmitted.

3 Algorithm

Algorithm intuition: 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. Reservoir sampling is a method for maintaining a random item from a stream of items, while using memory that is small when compared with the size of the stream. Indeed, one way to implement reservoir sampling is to assign a random weight to each stream item and maintain the item with the minimum weight. 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 sth 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 needs only maintain a single value u_i, which is the site's view of the sth 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 sth smallest weight changes $O(s \log(n/s))$ times, and updating every site each time the sth 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 makes a change to the random sample, or, at least, gets to refresh its view of u.

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, in
   the sense that it may not agree with the $s$th
   smallest weight held by the coordinator.      */
$u_i \leftarrow 1$;
```

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

Algorithm 2. When Site i Receives Element e

```
Let $w(e)$ be randomly chosen weight between 0 and 1;
if $w(e) < u_i$ then
    Send $(e, w(e))$ to the Coordinator;
    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                */
$\mathcal{P} \leftarrow \emptyset$;
/* $u$ 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 arrivesreturn $\mathcal{P}$;
```

Algorithm correctness. Lemmas 1 and 2 establish the correctness of the algorithm.

Lemma 1. *(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.*

Proof. The variable u is stored at the coordinator, and u_i is stored at site i. First we note that the variables u and u_i are non-increasing with time; this can be verified from the algorithms. Next, we note that for every i from 1 till k, at every round, $u_i \geq u$. This can be seen because initially, $u_i = u = 1$, and u_i changes only in response to receiving u from the coordinator.

Thus, if fewer than s elements have appeared in the stream so far, u is 1, and hence u_i is also 1 for each site i. The next element observed in the system is also sent to the coordinator. Thus, if $n \leq s$, then the set $\mathcal{P}$ consists of all elements seen so far in the system.

Next, we consider the case $n > s$. Note that u maintains the sth smallest weight seen at the coordinator, and $\mathcal{P}$ consists of the s elements seen at the coordinator with the smallest weights. We only have to show that if an element e, observed at site i is such that $w(e) < u$ then i must have sent $(e, w(e))$ to the coordinator. This follows because $u_i \geq u$ at all times, and if $w(e) < u$, then it must be true that $w(e) < u_i$, and in this case, $(e, w(e))$ is sent to the coordinator. □

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. □

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 sth 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_j denote the value of u at the end of epoch $j - 1$. Epoch j consists of all rounds subsequent to epoch $(j - 1)$ until (and including) the earliest round when u is $\frac{m_j}{r}$ 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 $\mathcal{S}_i, i = 1 \ldots k$ 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 j. In Algorithm B, v receives m_j at the beginning of the epoch through a message from the coordinator. In Algorithm A, v may not know m_j at the beginning of epoch j. We consider two cases.

Case I: v sends a message to the coordinator in epoch j 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_j. 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. □

Let ξ denote the total number of epochs.

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

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

Proof. Let $z = (\frac{\log(n/s)}{\log r})$. 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 $Y \geq s$

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

where we have used Markov's inequality.

Since ξ takes only positive integral values,

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

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 μ denote the total number of messages sent during the entire execution. Let μ_j denote the total number of messages sent in epoch j and X_j the total number of messages sent from the sites to the coordinator in epoch j. μ_j is the sum of two parts, (1) k messages sent by the coordinator at the start of the epoch, and (2) twice the number of messages sent from the sites to the coordinator

$$\mu_j = k + 2X_j, \tag{1}$$

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

For each $\kappa = 1 \ldots n_j$ in epoch j, we define a 0-1 random variable Y_κ as follows. $Y_\kappa = 1$ if observing the κth element in the epoch resulted in a message being sent to the coordinator, and $Y_\kappa = 0$ otherwise

$$X_j = \sum_{\kappa=1}^{n_j} Y_\kappa. \tag{3}$$

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

Lemma 5. *For each* $\kappa = 1 \ldots n_j - 1$

$$\Pr[Y_\kappa = 1 | F(\eta,\alpha)] \leq \frac{\alpha - \alpha/r}{1-\alpha/r}.$$

Proof. Suppose that the jth 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 $m_j = \alpha$, u_v is no more than α.

Note that in this lemma we exclude the last element that arrived in epoch j, thus the weight assigned to element j must be greater than α/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 $u_v \leq \alpha$. □

Lemma 6. *For each epoch* j

$$\mathbb{E}[X_j] \leq 1 + 2rs.$$

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

$$\begin{aligned}\mathbb{E}[X_j | F(\eta,\alpha)] &\leq 1 + \mathbb{E}\left[\left(\sum_{\kappa=1}^{\eta-1} Y_\kappa\right)|F(\eta,\alpha)\right] \\ &\leq 1 + \sum_{\kappa=1}^{\eta-1}\mathbb{E}[Y_\kappa|F(\eta,\alpha)] \\ &\leq 1 + (\eta-1)\frac{\alpha-\alpha/r}{1-\alpha/r}.\end{aligned}$$

Using $r \geq 2$ and $\alpha \leq 1$, we get: $\mathbb{E}[X_j|F(\eta,\alpha)] \leq 1 + 2(\eta-1)\alpha$.

We next consider the conditional expectation $\mathbb{E}[X_j | m_j = \alpha]$

$$\begin{aligned}&\mathbb{E}[X_j|m_j=\alpha] \\ &= \sum_{\eta}\Pr[n_j=\eta|m_j=\alpha]\mathbb{E}[X_j|n_j=\eta, m_j=\alpha] \\ &\leq \sum_{\eta}\Pr[n_j=\eta|m_j=\alpha](1+2(\eta-1)\alpha) \\ &\leq \mathbb{E}[1+2(n_j-1)\alpha|m_j=\alpha] \\ &\leq 1+2\alpha(\mathbb{E}[n_j|m_j=\alpha]-1).\end{aligned}$$

Using Lemma 7, we get

$$\mathbb{E}[X_j|m_j=\alpha] \leq 1 + 2\alpha\left(\frac{rs}{\alpha}-1\right) \leq 1+2rs.$$

Since $\mathbb{E}[X_j] = \mathbb{E}[\mathbb{E}[X_j|m_j=\alpha]]$, we have $\mathbb{E}[X_j] \leq \mathbb{E}[1+2rs] = 1+2rs$. □

Lemma 7.

$$\mathbb{E}[n_j|m_j=\alpha] = \frac{rs}{\alpha}.$$

Proof. Recall that n_j, the total number of elements in epoch j, is the number of elements observed till the sth minimum in the stream decreases to a value that is less than or equal to α/r.

Let Z denote a random variable that equals the number of elements to be observed from the start of epoch j

till s new elements are seen, each of whose weight is less than or equal to α/r. Clearly, conditioned on $m_j = \alpha$, it must be true that $n_j \leq Z$. For $\delta = 1$ to s, let Z_δ denote the number of elements observed from the state when $(\delta - 1)$ elements have been observed with weights that are less than α/r till the state when δ elements have been observed with weights less than α/r. Z_δ is a geometric random variable with parameter α/r.

We have $Z = \sum_{\delta=1}^{s} Z_\delta$ and $\mathbb{E}[Z] = \sum_{\delta=1}^{s} \mathbb{E}[Z_\delta] = \frac{sr}{\alpha}$. Since $\mathbb{E}\left[n_j | m_j = \alpha\right] \leq \mathbb{E}[Z]$, the lemma follows. □

Lemma 8.

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

Proof. Using Lemma 6 and Equation (1), we get the expected number of messages in epoch j:

$$\mathbb{E}\left[\mu_j\right] \leq k + 2(2rs + 1) = k + 2 + 4rs.$$

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

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

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

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

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,

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

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

5 Lower Bound

Theorem 2. *For a constant $q, 0 < q < 1$, any correct protocol must send $\Omega(\frac{k \log(n/s)}{\log(1+(k/s))})$ 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 $\xi = \Theta(\frac{\log(n/s)}{\log(1+(k/s))})$ epochs as follows: in the jth epoch, $j \in \{0, 1, 2, \ldots, \xi - 1\}$, there are $\beta^{j-1}k$ 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 jth epoch, we randomly assign the $\beta^{j-1}k$ updates among the k servers, independently for each epoch. Let the randomness used for the assignment in the jth epoch be denoted σ_j.

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 $\mathcal{P}_\kappa$ denote a random variable indicating the value of $\mathcal{P}$ after seeing κ updates in the stream. We will use the following lemma about reservoir sampling.

Lemma 9. *For any constant $q, 0 < q < 1$, 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 $\xi = \omega(1)$, then with probability at least $1 - q/2$, over the choice of $\{\mathcal{P}_\kappa\}$, there are at least $(1 - (q/8))\xi$ 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 [2], $\mathcal{P}$ is initialized to $\{1, 2, 3, \ldots, s\}$. Then, for each $\kappa > s$, the κth element is included in the current sample set $\mathcal{P}_\kappa$ with probability s/κ, in which case a random item in $\mathcal{P}_{\kappa-1}$ is replaced with κ.

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

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

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

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

$$\Pr[Y_j < \mathbb{E}\left[Y_j\right]/2] \leq \exp(-\mathbb{E}\left[Y_j\right]/8)$$
$$\leq \exp(-(s \ln \beta + O(1))/8)$$
$$\leq \frac{1}{\beta^{s/8}}.$$

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

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 the changes implied by the first part Lemma 9 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 $\xi = O(1)$, 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 Π be a possibly randomized protocol, which with probability at least q, sends at most $Ck\xi$ messages. We show that Π cannot be a correct protocol.

Let τ denote the random coin tosses of Π, 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 Π sends less than $Ck\xi$ messages. By assumption, $\Pr_\tau[\mathcal{E}] \geq q$. Hence, it is also the case that

$$\Pr_{\tau,\{\mathcal{P}_j\},\{\sigma_j\}}[\mathcal{E}] \geq q.$$

For a sufficiently small constant $C' > 0$ that may depend on q, let $\mathcal{F}$ be the event that there are at least $(1-(q/8))\xi$ 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,

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

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

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

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

By a Markov argument, if event $\mathcal{E}$ occurs, then there are at least $(1-(q/8))\xi$ 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 $(1-(q/4))\xi$ 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 j^* be the epoch which is most likely to be balanced, over the random choices of $\{\sigma_j\}$, conditioned on $\tau = \tau'$ and $(\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_\xi) = (P'_0, P'_1, \ldots, P'_\xi)$. Since at least $(1-(q/4))\xi$ epochs are balanced if $\mathcal{E}$ and $\mathcal{F}$ occur, and conditioned on $(\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_\xi) = (P'_0, P'_1, \ldots, P'_\xi)$ event $\mathcal{F}$ does occur, and $\mathcal{E}$ occurs with probability at least $q/2$ given this conditioning, it follows that

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

The property of j^* being balanced is independent of $\sigma_{j'}$ for $j' \neq j^*$, so we also have

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

If $C's \log(1+(k/s)) \geq 1$, then $\mathcal{P}$ changes at least once in epoch j^*. 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 j^* to be balanced for at least a $q/4$ fraction of the σ_{j^*}, there must be at least $qk/4$ different servers which receive j^*, for which Π sends a message. In particular, since Π is deterministic conditioned on τ, at least $qk/4$ messages must be sent in the j^*th epoch. But j^* 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 $Ck\xi$ messages must be sent with probability at least $1-q$. Since $Ck\xi = \Omega\left(\frac{k \log(n/s)}{\log(1+(k/s))}\right)$, this completes the proof. □

6 Analysis Under Skew

For analyzing the performance in the case of skew, we consider the following model of arrival of data. Suppose a set of parameters $p_i, i = 1 \ldots k$, one per site, such that $\sum_{i=1}^k p_i = 1$. Each new arrival into the system is sent to stream $\mathcal{S}_i$ with a probability of p_i. The p_is need not be equal, so that the numbers of elements seen by different sites can be significantly different from each other. Note that we do not present an algorithm tailored for this model of arrival. Instead, we analyze the same algorithm as before, under this model. The main result of this section is presented in the following theorem.

Theorem 3. *The method described in Algorithms 1, 2, and 3 continuously maintains a sample of a distributed system with an expected total number of messages $O(s \log(n/s) \sum_{i,p_i \neq 0} \frac{1}{\log(\frac{2}{p_i})})$ over n arrivals.*

In our algorithm, all communication is initiated by the sites. Note that in our analysis in Section 4, we make use of an algorithm, Algorithm B, where at the end of an epoch there is communication initiated by the coordinator. Algorithm B is only to help with the analysis, and in our algorithm, no communication is initiated by the coordinator. Hence, if $p_i = 0$ for some site i, then the site will not receive any elements, and will not send any messages to the coordinator. We can ignore such sites and henceforth, we assume that for each site i, $p_i > 0$.

For the purpose of analysis, we divide the execution of the system into epochs. Unlike Section 4, where epochs were defined globally, we define epochs differently for different sites here. For each site i where $p_i \neq 0$, let $\rho_i = \frac{2}{p_i}$. The jth epoch at site i, for $j = 1\ldots$ is defined inductively as follows. The first epoch at site i is the first $s\rho_i$ elements received in the system (note this is not the number of elements received in $\mathcal{S}_i$, but the number of elements in $\mathcal{S}$). For $j \geq 2$, the jth epoch at site i consists of the next $s\rho_i^j$ arrivals in the system after the $(j-1)$th epoch.

Lemma 10. *The number of epochs at site i is no more than* $\lceil \frac{\log(n/2s)}{\log\ \rho_i} \rceil$.

Proof. Let $\eta(r)$ be the number of elements received by the system in epoch r of site i, and $\zeta(r)$ be the total number of elements received by the system until (and including) epoch r of site i.

$$\zeta(r) = \sum_{j=1}^{r} \eta(r) = \sum_{j=1}^{r} s\rho_i^j \leq 2s\rho_i^r.$$

The final inequality above is true since $\rho_i \geq 2$. When there are n elements observed so far, there are ℓ_i epochs for site i. We set $\zeta(\ell_i) = n$, and conclude $\ell_i = \lceil \frac{\log(n/2s)}{\log \rho_i} \rceil$. □

For site i and epoch j, let X_i^j denote the number of messages sent by site i to the coordinator in epoch j.

Lemma 11.

$$\mathbb{E}\left[X_i^j\right] \leq (s+1).$$

Proof. Consider the jth epoch at site i. Suppose that the elements that arrived in the system during this epoch are $Q = \{e_1, e_2, \ldots, e_{\rho_i^j}\}$. We split Q into two parts: 1) Q_1 is the set of all elements observed before site i sends the first message to the coordinator in its jth epoch; and 2) Q_2 is the set of the remaining elements in the epoch.

After observing Q_1, site i sends out one message. Let $X_{i,2}^j$ be the number of messages that site i sends to the coordinator when observing Q_2. Note that $X_i^j = 1 + X_{i,2}^j$.

For each element e in Q_2, let $X_{i,2}^j(e)$ be a random variable defined as follows. $X_{i,2}^j(e) = 1$ if site i sends a message to coordinator after receiving element e, and 0 otherwise. Note that for site i to send a message to the coordinator upon receiving an element e in part 2 of the epoch, the random number chosen for this element must be smaller than the sth smallest random weights in the first $(j-1)$ epochs at site i

$$\begin{aligned}\Pr\left[X_{i,2}^j(e) = 1\right] &= \Pr[e \text{ was sent to } i] \cdot \\ &\quad \Pr\left[X_{i,2}^j(e) = 1 | e \text{ was sent to } i\right] \\ &\leq p_i \cdot \frac{s}{2s\rho_i^{j-1}} = \frac{p_i}{2\rho_i^{j-1}}.\end{aligned}$$

Note that $X_{i,2}^j = \sum_{e\in Q_2} X_{i,2}^j(e)$, and $|Q_2| \leq s\rho_i^j$

$$\mathbb{E}\left[X_{i,2}^j\right] = \sum_{e\in Q_2} \Pr\left[X_{i,2}^j(e) = 1\right] \leq s\rho_i^j \cdot \frac{p_i}{2\rho_i^{j-1}} = \frac{sp_i\rho_i}{2} = s.$$

Therefore, we conclude $\mathbb{E}\left[X_i^j\right] = 1 + \mathbb{E}\left[X_{i,2}^j\right] \leq (s+1)$. □

Proof of Theorem 3. Let X_i be the total number of messages sent by site i, and let X be the total number of messages sent by all sites. Note that $X_i = \sum_{j=1}^{\xi_i} X_i^j$ and $X = \sum_{i=1}^{k} X_i$, where ξ_i is the number of epochs at site i. Using Lemma 10, we get:

$$\mathbb{E}[X_i] = \sum_{j=1}^{\xi_i} \mathbb{E}\left[X_i^j\right] \leq (s+1)\xi_i \leq \frac{(s+1)\log(n/2s)}{\log \rho_i},$$

$$\mathbb{E}[X] \quad = \sum_{i=1}^{k} \mathbb{E}[X_i] \leq \sum_{i=1}^{k} \tfrac{(s+1)\log(n/2s)}{\log \rho_i}.$$

□

Observation 1. With a uniform data distribution, i.e., $p_i = \frac{1}{k}$, the upper bound of communication complexity in Theorem 3 is of the same order as the upper bound from Lemma 8.

Proof. For $p_i = \frac{1}{k}$, $\rho_i = \frac{2}{p_i} = 2k$. Using Theorem 3, with a uniform distribution, the communication complexity is bounded by $O\big(ks\frac{\log(n/2s)}{\log(2k)}\big)$.

Using Lemma 8, the communication complexity is $\mathbb{E}[\mu] \leq (k + 4rs + 2)\big(\frac{\log(n/s)}{\log r} + 2\big)$. If we choose $r = 2k$, this expression is $O\big(ks\frac{\log(n/s)}{\log 2k}\big)$, which is of the same order as the expression derived from Theorem 3. □

The following theorem makes use of Algorithms 5 and 4 below.

Theorem 4. *For the CMYZ algorithm, the expected message complexity is unaffected by the skew, and is $\Theta(k \log_{k/s} n + s \log n)$ under any element distribution.*

Proof. In the CMYZ algorithm, the execution is divided into rounds (see Algorithms 4, 5 where we have reproduced a description of these algorithms). In each round a sample is collected at the coordinator, and when the size of this sample reaches s, the coordinator broadcasts a signal to advance to the next round.

In Algorithm 5 (coordinator), it is clear that for a given round, it does not matter who communicates with the coordinator during the round, the messages sent by the coordinator within the round are the same—there is a single broadcast from the coordinator to all the sites. From Algorithm 4, we see that the communication between a site and the coordinator also depends only on the round number and the random bit string assigned to an element, and is unaffected by which site actually sees the element. Hence, if we redistribute the elements to sites in a different manner, but kept the random bit strings (for each elements) the same, then the same set of elements will lead to messages to the coordinator as before. Hence, the communication from the site to the coordinator, and the progression from one round to another, are both unaffected by skew. Overall, we see that both the sequence of messages sent by the sites to the coordinator, and the messages from the coordinator to the site, are unaffected by the actual distribution of elements across sites. □

7 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 as follows. Let the algorithm from Section 3, when specialized to maintain a random sample of size one, be called the "single item algorithm". To maintain a random sample of size s, we run s independent copies of the single item algorithm. When there is a query for a random sample of size s, we return the (multi-set) union of the samples returned by the s copies. Since each element in the set returned is chosen uniformly at random from $\mathcal{S}$, the returned set is indeed a random sample chosen from $\mathcal{S}$ with replacement. Performed 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, $\mathcal{S}^\delta, \delta = 1 \ldots s$. Each $\mathcal{S}^\delta$ 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 $w^\delta(e)$ denote the weight assigned to element e in $\mathcal{S}^\delta$. $w^\delta(e)$ is a random number between 0 and 1. For each $\delta = 1 \ldots s$, the coordinator maintains the minimum weight, say w^δ, among all elements in $\mathcal{S}^\delta$, and the corresponding element.

Let $\beta = \max_{\delta=1}^{s} w^\delta$; β is maintained by the coordinator. Each site i maintains β_i, a local view of β, which is always greater than or equal to β. Whenever a logical stream element at site i has weight less than β_i, the site sends it to the coordinator, receives in response the current value of β, and updates β_i. When there is a query 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 β, which provides partial information about all w^δs. This provides a substantial improvement in the message complexity and leads to the following bounds.

Theorem 5. *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 $O\big(k \frac{\log n}{\log(\frac{k}{s \log s})}\big)$ in case $k > 2s \log s$.*

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

The n_j elements in epoch j give rise to sn_j logical elements, and each logical element has a probability of no more than m_j of resulting in a message to the coordinator. Similar to the proof of Lemma 6, we can show using conditional expectations that $\mathbb{E}[X_j] \leq rs \log s$ (the $\log s$ factor comes in due to the fact that $\mathbb{E}[n_j | m_j = \alpha] \leq \frac{r \log s}{\alpha}$. 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. □

8 Experiments

We report on an experimental evaluation of our algorithm. We implemented our algorithm using Java, and tested it on data derived from an OC48 Internet Trace [30], which has anonymous traffic traces taken at a US west coast OC48 peering link for a large ISP in 2002 and 2003. The dataset has 42,268,510 elements. We also evaluated its performance on a different dataset (Enron email [31]). But as expected, the performance of the algorithms do not depend on the specific dataset used, so we only present the results for the OC48 dataset.

As a point of comparison, we implemented the CMYZ algorithm [11]. Each data point in the graph below is the mean of 50 independent runs of the experiment. For reference, we reproduce the CMYZ algorithm from [11] in Algorithms 4, and 5. Note that our implementation of CMYZ is slightly different from the one given in [11], and corrects their stated algorithm, for the following reason. In Algorithm 5, T_j is the set of possible samples in the coordinator in round j. Upon request, the coordinator responds with a sample of size s from $T_j \cup T_{j+1}$. With a small probability $2^{-(s+1)}$, all elements in T_{j+1} at the coordinator will be moved to T_{j+2}, and the next element is added to T_{j+2} in round $j+1$. This will lead to $|T_{j+2}| > s$ and the size of T_{j+2} will not decrease in the future, and the memory of the coordinator can grow. Therefore, we revise the condition for the "if" statement (Line 5 in Algorithm 5) to be $|T_{j+1}| \geq s$. Also note that while our method keeps s elements in the coordinator at all times, CMYZ may require more memory at the coordinator. In particular, in round j, $|T_j|$ can be large if $|T_{j+1}| < s$.

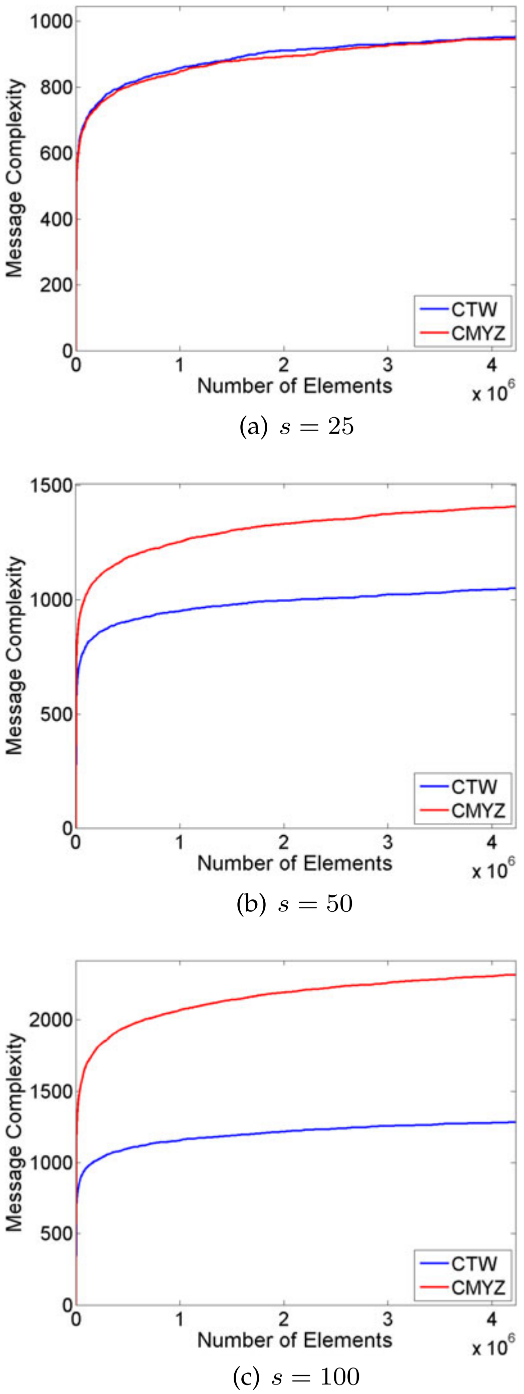
Algorithm 4. Algorithm for Site in Round j, from CMYZ [11]

```
Upon receiving element $e$, let $b(e)$ be a random bit string
assigned to $e$. ;
if the first $j$ bits of $b(e)$ are all zero then send $e$ to Coordinator;
```

Algorithm 5. Algorithm for the Coordinator in Round j, from CMYZ [11]

```
foreach $e$ received do
    if $b(e)[j+1] = 0$ then
        $T_{j+1} \leftarrow T_{j+1} \cup \{e\}$
    else $T_j \leftarrow T_j \cup \{e\}$;
    if $|T_{j+1}| = s$ then
        foreach $e \in T_{j+1}$ do
            if $b(e)[j+2] = 0$ then
                $T_{j+2} \leftarrow T_{j+2} \cup \{e\}$;
                $T_{j+1} \leftarrow T_{j+1} \backslash \{e\}$;
            discard $T_j$;
            $j \leftarrow j+1$ and signal all sites to advance to the next
            round;
```

Fig. 1 shows the number of message transmissions against the number of elements observed so far. For this set of experiments, each stream element is sent to a site chosen

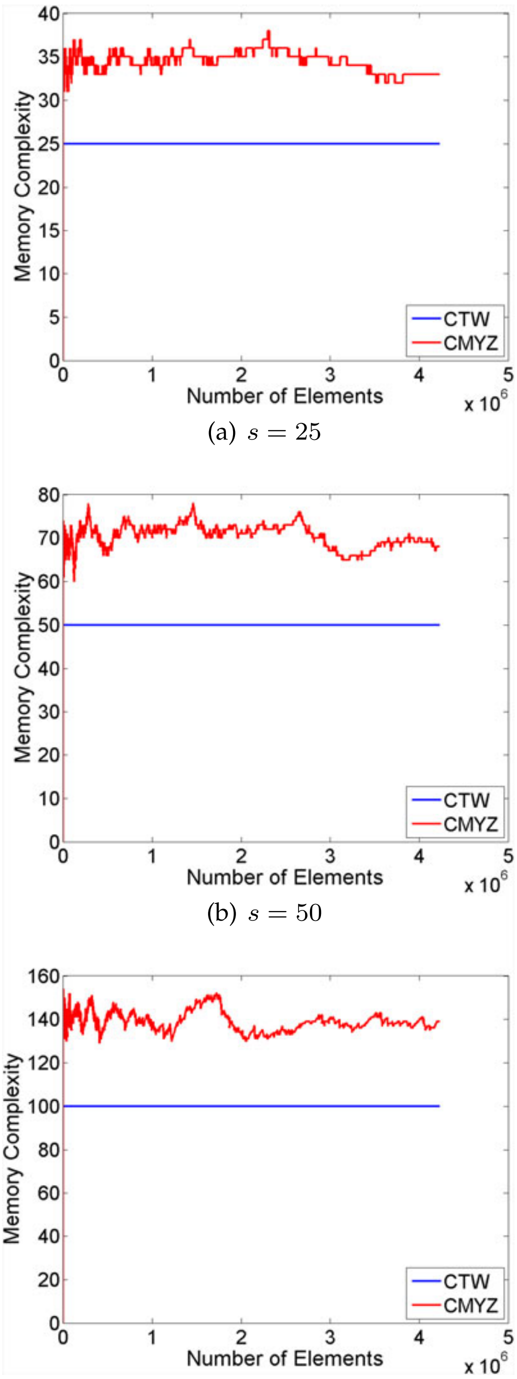


(a) $s = 25$

(b) $s = 50$

(c) $s = 100$

Fig. 1. The number of messages transmitted as a function of number of elements observed. The number of sites is set to 20. "CTW" refers to our algorithm while CMYZ is the algorithm of Cormode et al. [11].



(a) $s = 25$

(b) $s = 50$

(c) $s = 100$

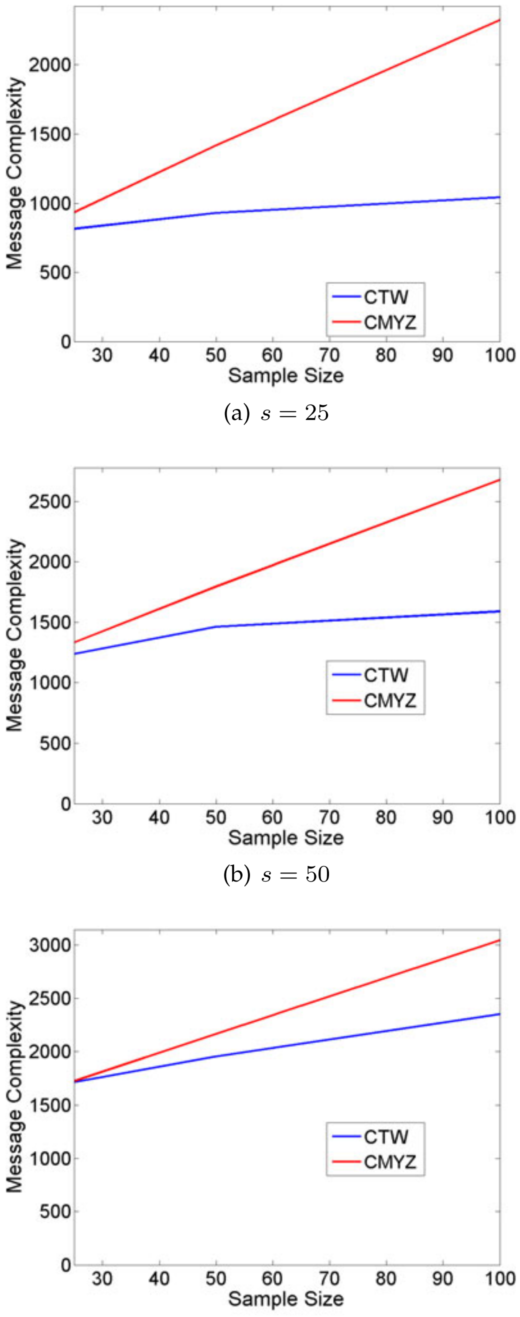
Fig. 2. Memory consumption versus stream size for 20 sites.

uniformly at random, so that all sites receive approximately the same number of elements. In all Figures "CTW" refers to our algorithm while CMYZ is the algorithm of Cormode et al. [11].

From Fig. 1, we observe that the communication cost of both algorithms have a logarithmic dependence on the number of elements observed so far. As is expected, at the beginning of the observation, the number of message transmissions grows fast, since an incoming element has a high probability of being sampled. As more elements are observed, this probability decreases. Note that in general, our algorithm has a lower communication cost than CMYZ, and the benefit of our algorithm increases as the sample size increases from 25 to 100.

The memory consumption of the two methods is presented in Fig. 2. Note that while our method consumes a constant amount of memory, CMYZ takes more memory, and its memory consumption is a random variable.

In Fig. 3, it shows the message transmission as a function of sample size. The message complexity increases almost linearly with the sample size, for both algorithms. We observe that our algorithm performs better on high sample size than CMYZ. In the algorithm of CMYZ, the procedure remains in the same round if $|T_{j+1}| < s$. The sampling probability for

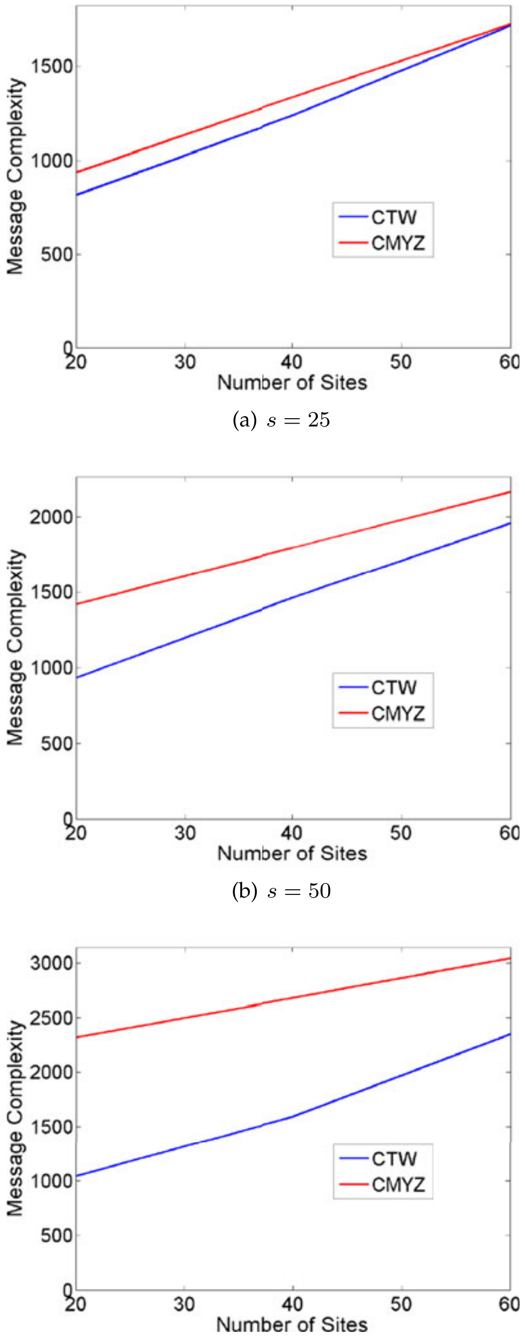


(a) $s = 25$

(b) $s = 50$

(c) $s = 100$

Fig. 3. Number of messages versus sample size, where k is the number of sites.



(a) $s = 25$

(b) $s = 50$

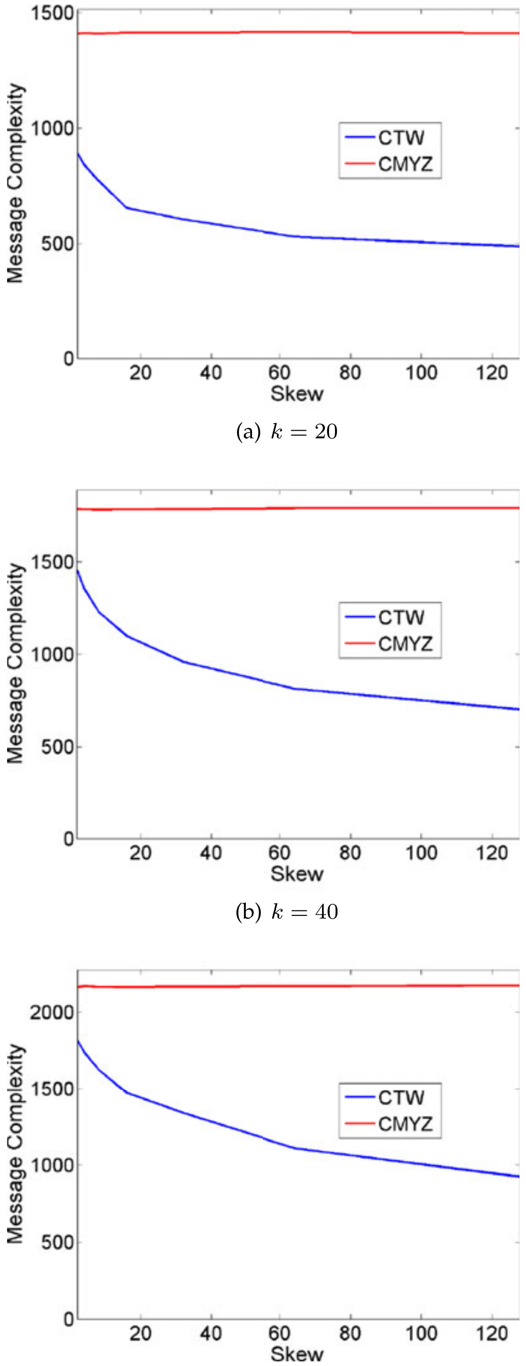
(c) $s = 100$

Fig. 4. Number of messages versus number of sites.

any new coming element in the round is the same. Different from CMYZ, the probability that a new element is sampled in a epoch is decreasing. With higher sample size, the number of elements in an epoch/turn increases, which makes CMYZ require more communication than our algorithm.

Fig. 4 shows the message transmission as a function of number of sites. We observe our method performs better under these three scenarios. We also observe that when the number of sites increases, the communication cost of our algorithm gets closer to that of CMYZ. The reason is that with a greater number of sites, our algorithm is more likely to have sites whose values of u_i are not synchronized with the coordinator. As a result, there are more messages sent that do not change the sample at the coordinator.

Fig. 5 shows the message transmission under skew. For this set of experiments, we selected one site to have a higher probability to receive incoming elements than the other sites. Thus site 1 receives the next item with probability p, while the remaining sites 2 till $(k-1)$ received the element with a probability of $(1-p)/(k-1)$. We call the ratio between the probability of 1 receiving the element and the probability that another site receives it, as the *skew* in the data. According to theoretical analysis, the performance of our method should improve with increasing skew, while that of CMYZ should remain constant. From Fig. 5 we observe that this is indeed the case, and for larger values of the skew, our algorithm significantly outperforms CMYZ.

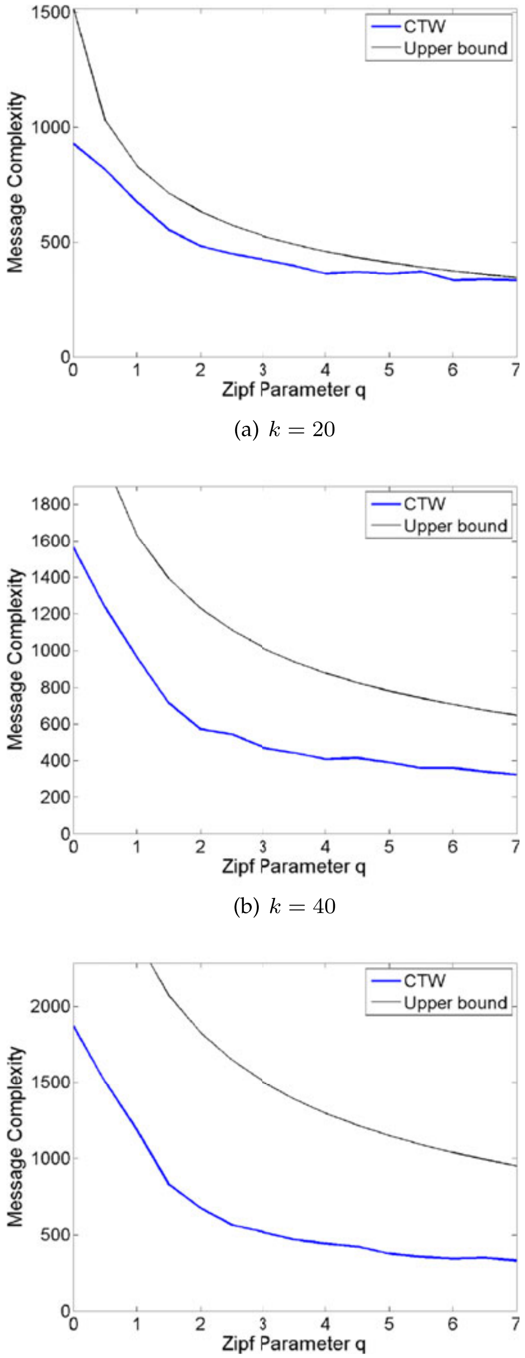


(a) $k = 20$

(b) $k = 40$

(c) $k = 60$

Fig. 5. Number of messages versus the skew, when sample size is 50.



(a) $k = 20$

(b) $k = 40$

(c) $k = 60$

Fig. 6. Number of messages versus Zipf Distribution variable q, when the sample size is 50.

Fig. 6 shows an experiment with a different type of input. We apply the Zipf Distribution [32], where the probability of each site receiving new elements, p_i, is assigned to be $\frac{c}{i^q}$, where c is normalization constant, and q is a parameter. From Fig. 6, we note that the number of messages transmitted decreases when q increases (as the degree of skewness increases).

9 Conclusion

We presented a simple message-optimal algorithm for maintaining a uniform random sample, with or without replacement, from a distributed stream. Our theoretical results and experiments show that this algorithm outperforms previous methods with respect to message complexity under a variety of scenarios, especially when the distribution of elements across sites is skewed. The correctness of our algorithm is simple to establish. Though our analysis of its message complexity is based on a synchronous model as described in Section 2 the algorithm works in the absence of synchronous communication, in the following sense. Consider an asynchronous distributed system where the message delays are not predictable. Consider the state of the system at some instant in time, and suppose that we stopped adding further elements to the local streams from that instant onwards. Then, if every site processes all elements according to our algorithm, all messages sent by the sites were delivered to the coordinator, and responses from the coordinator were delivered back to the sites, and so on until the system quiesced. At the end, the coordinator will have a random sample of size s from the union of all elements across all local streams—it is only necessary that the messages due to the s elements with the smallest weight are received by the coordinator.

References

[1] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," in *Proc. 19th Annu. ACM-SIAM Symp. Discr. Algorithm*, 2008, pp. 1076–1085.

[2] D. E. Knuth, *The Art of Computer Programming, Volume II: Seminumerical Algorithms*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1981.

[3] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[4] 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 *Proc. IEEE INFOCOM*, 2007, pp. 134–142.

[5] G. Cormode and M. N. Garofalakis, "Sketching streams through the net: Distributed approximate query tracking," in *Proc. 31st Int. Conf. Very Large Databases*, 2005, pp. 13–24.

[6] B. Babcock and C. Olston, "Distributed top-k monitoring," in *Proc. SIGMOD Conf.*, 2003, pp. 28–39.

[7] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 289–300.

[8] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston, "Finding (recently) frequent items in distributed data streams," in *Proc. 21st Int. Conf. Data Eng.*, 2005, pp. 767–778.

[9] K. Yi and Q. Zhang, "Optimal tracking of distributed heavy hitters and quantiles," in *Proc. 28th ACM SIGMOD-SIGACT-SIGART Symp. Principles Distrib. Comput.*, 2009, pp. 167–174.

[10] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: Queries with bounded errors and bounded response times on very large data," in *Proc. 8th Eurosys Conf.*, 2013, pp. 29–42.

[11] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Continuous sampling from distributed streams," *J. ACM*, vol. 59, no. 2, pp. 10:1–10:25, 2012.

[12] C. Arackaparambil, J. Brody, and A. Chakrabarti, "Functional monitoring without monotonicity," in *Proc. 36th Int. Colloquium Automata, Languages Program.*, 2009, pp. 95–106.

[13] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[14] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Inf. Process. Lett.*, vol. 97, no. 5, pp. 181–185, 2006.

[15] V. Braverman, R. Ostrovsky, and C. Zaniolo, "Optimal sampling from sliding windows," *J. Comput. Syst. Sci.*, vol. 78, pp. 260–272, 2012.

[16] B. Xu, S. Tirthapura, and C. Busch, "Sketching asynchronous data streams over sliding windows," *Distrib. Comput.*, vol. 20, no. 5, pp. 359–374, 2008.

[17] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *Proc. 13th Annu. ACM-SIAM Symp. Discr. Algorithm*, 2002, pp. 633–634.

[18] R. Gemulla and W. Lehner, "Sampling time-based sliding windows in bounded space," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 379–392.

[19] C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 607–618.

[20] A. Pavan, K. Tangwongsan, S. Tirthapura, and K. Wu, "Counting and sampling triangles from a graph stream," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 1870–1881, 2013.

[21] M. Dash and W. Ng, "Efficient reservoir sampling for transactional data streams," in *Proc. IEEE 6th Int. Conf. Data Mining*, 2006, pp. 662–666.

[22] V. Malbasa and S. Vucetic, "A reservoir sampling algorithm with adaptive estimation of conditional expectation," in *Proc. Int. Joint Conf. Neural Netw.*, 2007, pp. 2200–2204.

[23] C. C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *Proc. 32nd Int. Conf. Very Large Databases*, 2006, pp. 607–618.

[24] V. Braverman, R. Ostrovsky, and C. Zaniolo, "Optimal sampling from sliding windows," in *Proc. 28th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2009, pp. 147–156.

[25] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Optimal sampling from distributed streams," in *Proc. 29th ACM-SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2010, pp. 77–86.

[26] Y.-Y. Chung and S. Tirthapura, "Distinct random sampling from a distributed stream," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 532–541.

[27] A. McGregor, A. Pavan, S. Tirthapura, and D. P. Woodruff, "Space-efficient estimation of statistics over sub-sampled streams," in *Proc. 31st ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2012, pp. 273–282.

[28] P. B. Gibbons and S. Tirthapura, "Estimating simple functions on the union of data streams," in *Proc. 13th Annu. ACM Symp. Parallel Algorithms Archit.*, 2001, pp. 281–291.

[29] P. B. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," in *Proc. 14th Annu. ACM Symp. Parallel Algorithms Archit.*, 2002, pp. 63–72.

[30] CAIDA OC48 Trace Project. (2006). The CAIDA UCSD OC48 internet traces dataset - (2002-2003) [Online]. Available: http://imdc.datcat.org/contact/1-0037-M=CAIDA+OC48+Trace+Project

[31] CALO Project. (2009, Aug.). Enron email dataset [Online]. Available: http://www.cs.cmu.edu/enron/

[32] D. M. W. Powers. (1998). Applications and explanations of zipf's law. *Proc. Joint Conf. New Methods Language Process. Comput. Natural Language Learn.*, pp. 151–160 [Online]. Available: http://dl.acm.org/citation.cfm?id=1603899.1603924

Yung-Yu Chung received the BS degree in computer science and information engineering from National Taiwan University in 2006 and the MS degree in computer engineering from Iowa State University in 2013. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering at Iowa State University. He is interested in high-performance computation system on a data stream, data stream mining, and machine learning.

Srikanta Tirthapura received the BTech degree in computer science and engineering from IIT Madras in 1996 and the PhD degree in computer science from Brown University in 2002. He is an associate professor in the Department of Electrical and Computer Engineering, Iowa State University. He is interested in algorithm design for big data, including data stream algorithms and parallel and distributed algorithms. He received the IBM Faculty Award and the Warren Boast Award for excellence in Undergraduate Teaching.

David P. Woodruff received the PhD degree from MIT in theoretical computer science, and spent a year at Tsinghua University. He is a research staff member at IBM Almaden Research Center. His current research interests include communication complexity, data stream algorithms and lower bounds, graph algorithms, machine learning, numerical linear algebra, sketching, and sparse recovery. He received the 2014 EATCS Presburger Award.