Optimal Sampling From Distributed Streams*

Graham Cormode
AT&T Labs - Research
graham@research.att.com

S. Muthukrishnan
Rutgers University
muthu@cs.rutgers.edu

Ke Yi, Qin Zhang
HKUST
{yike,qinzhang}@cse.ust.hk

ABSTRACT

A fundamental problem in data management is to draw a sample of a large data set, for approximate query answering, selectivity estimation, and query planning. With large, streaming data sets, this problem becomes particularly difficult when the data is shared across multiple distributed sites. The challenge is to ensure that a sample is drawn uniformly across the union of the data while minimizing the communication needed to run the protocol and track parameters of the evolving data. At the same time, it is also necessary to make the protocol lightweight, by keeping the space and time costs low for each participant. In this paper, we present communication-efficient protocols for sampling (both with and without replacement) from k distributed streams. These apply to the case when we want a sample from the full streams, and to the sliding window cases of only the W most recent items, or arrivals within the last w time units. We show that our protocols are optimal, not just in terms of the communication used, but also that they use minimal or near minimal (up to logarithmic factors) time to process each new item, and space to operate.

Categories and Subject Descriptors

F.2.2 [**Analysis of algorithms and problem complexity**]: Nonnumerical algorithms and problems; H.2.4 [**Database management**]: Systems—*distributed databases*

General Terms

Algorithms, theory

Keywords

Distributed tracking, random sampling

*Ke Yi and Qin Zhang are supported in part by Hong Kong Direct Allocation Grant (DAG07/08). Qin Zhang is in addition supported by Hong Kong CERG Grant 613507.

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

1. INTRODUCTION

It is increasingly important for data management systems to cope with large quantities of data that are observed at geographically distributed locations. As data volumes increase (through greater power of measurement in sensor networks, or increased granularity of measurements in network monitoring settings), it is no longer practical to collect all the data together in a single location and perform processing using centralized methods. Further, in many of the motivating settings, various monitoring queries are lodged which must be answered *continuously*, based on the total data that has arrived so far. These additional challenges have led to the formalization of the *continuous, distributed, streaming model* [8]. In this model, defined more formally below, a number of distributed peers each observe a high-speed stream of data, and collaborate with a centralized coordinator node to continuously answer queries over the union of the input streams.

Protocols have been defined in this model for a number of classes of queries, which aim to minimize the communication, space and time needed by each participant. However, the protocols proposed so far seem to have overlooked the fundamental problem of producing a sample drawn from the distributed streams. A sample is a powerful tool, since it can be used to approximately answer many queries. Various statistics over the sample can indicate the current distribution of data, especially if it is maintained to be drawn only from a recent history of data. In this paper, we present techniques for drawing a sample over distributed data streams either with or without replacement, and show how to adapt them for sliding windows of recent updates.

In doing so, we build on the long history of drawing a sample from a single stream of elements. Random sampling, as a fundamental problem and a basic tool for many applications, had been studied in the streaming setting long before formal models of data streams were first introduced. The classical *reservoir sampling* algorithm [15] (attributed to Waterman) maintains a random sample of size s without replacement over a stream. It is initialized with the first s elements; when the i-th element arrives for $i > s$, with probability $1/i$ it adds the new element, replacing an element uniformly chosen from the current sample. It is clear that this algorithm uses optimal $O(s)$ space and $O(1)$ time per element.

There have been various extensions to the basic reservoir sampling algorithm. Using an appropriate distribution, it is possible to determine how many forthcoming elements to "skip" over until the next sample will be drawn [21]. More recently, there has been much interest in understanding how to maintain a uniform sample efficiently over a *sliding win-*

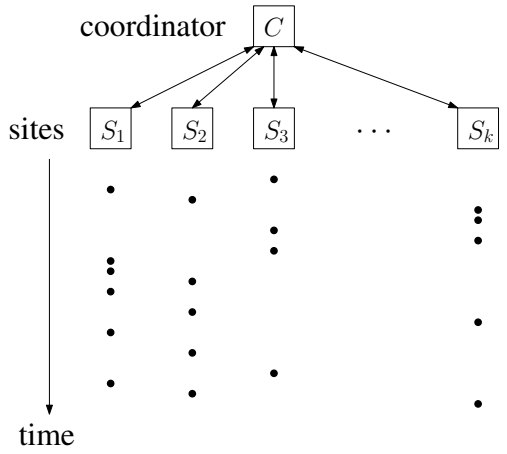

Figure 1: The distributed streaming model.

dow [2, 4, 10]. There are two models for sliding windows: in *sequence-based windows*, we must maintain a sample over the last W elements in the stream; in *time-based windows*, every element arrives at a particular time (called its *timestamp*), and we want to maintain a sample over the elements that have arrived in the time interval $[t-w, t]$ for a window length w, where t denotes the current time. Time-based windows are more intuitively useful than sequence-based windows but usually require more complex algorithms and more space and time resources to handle.

Distributed streaming. Many streaming applications involve multiple, say k, streams distributed in different locations linked by a network. The goal is to track some function at a designated coordinator over the combined data received from all the streams, as opposed to just one. This is shown schematically in Figure 1. For example, consider a collection of routers in a network, each of which processes a high-speed stream of packets. Maintaining a random sample of the packets from the union of these streams is valuable for many network monitoring tasks where the goal is to detect some global features [13]. Beyond network monitoring, similar problems also arise naturally in applications like distributed databases, telecommunications, web services, sensor networks, and many others.

In this setting, the communication cost is the primary measure of complexity of a tracking algorithm, while its space and time costs are also important to bound. Motivated by the many applications in networking and databases, there has been a lot of work on designing communication-efficient algorithms for tracking certain functions (including frequent items [3, 14, 16, 22], quantiles [7, 22], frequency moments [6, 8], various sketches [7, 9], entropy [1], and other non-linear functions [18, 19]) over distributed streams. But surprisingly, the important and fundamental problem of random sampling has not yet been addressed.

Problem definition. We formally define our problem as follows. Let $A = (a_1, \ldots, a_n)$ be a sequence of elements. The sequence A is observed in an online fashion by k remote *sites* $S_1, \ldots, S_k$ collectively, i.e., a_i is observed by exactly one of the sites at time t_i, where $t_1 < t_2 < \cdots < t_n$. It is assumed that a_i arrives with its timestamp t_i, but the index i, namely its global sequence number, is unknown to a_i. Let $A(t)$ be the set of elements received up until the current time t from all sites. There is a designated *coordinator* C who has a two-way communication channel to each of the k sites and needs to maintain a random sample of size s (with or without replacement) of $A(t)$ at all times.

We also consider two sliding-window versions of the problem. In a *sequence-based window*, for a given window size W, the coordinator needs to maintain a random sample over the last W elements received across all sites; in a *time-based window*, for a given window duration w, the coordinator maintains a random sample over the set $A(t) \setminus A(t-w)$ at all times t.

In the above setting, our primary concern is the total communication cost between the coordinator and the k sites. There are no other direct communications allowed between sites; but up to a factor of 2 this is not a restriction. We assume that communication is instantaneous. Other than communication we also care about the space/time costs for the coordinator as well as for each site. Note that this model is more general than the standard streaming model: by setting $k = 1$ and ignoring the communication cost, our model degenerates into the latter for both the non-windowed and windowed cases.

One of the major challenges of distributed sampling is that the current value of i (the total number of elements that have arrived) is not known. In fact, tracking i exactly requires every site to notify the coordinator upon the arrival of each element, costing $\Theta(n)$ communication. In the standard streaming model with a single stream, i is trivial to track, and algorithms in this model can rely on this knowledge. For example, the reservoir sampling algorithm samples the i-th element with probability $1/i$. Similarly, the sliding-window algorithms of [2, 4], track exactly how many elements have arrived since a "landmark" time T. One approach would be to use existing methods to track i approximately [8, 14]. But this does not immediately yield efficient algorithms, and any sample maintained will be somewhat approximate in nature: some elements will be more likely to be sampled than others. Such non-uniformity is undesirable, since it is unclear how this error will impact approximations based on the sample, and how this will propagate in various applications.

Our results. In this paper, we present algorithms that maintain a true random sample (i.e., no approximation) over distributed streams, without explicitly tracking i. Our asymptotic bounds are summarized in Table 1 for sampling without replacement under different settings. We measure the communication and space costs in terms of *words*, and assume that quantities polynomial in k, s and n can be represented in $O(1)$ words. For the two sliding-window cases, the bounds are for each window where W denotes the number of elements in the window; note that W may vary from window to window in the time-based case. In the table we do not show the results for sampling-with-replacement, since the bounds are quite similar.

Ignoring the log factors, our algorithms are optimal simultaneously in *all* the five measures; in fact most bounds are even optimal while considering the log factors. Some bounds are clearly optimal, such as a site taking constant time to process each new element and the coordinator requiring $\Omega(s)$ space, while others are less obvious. We prove the lower bounds after presenting the algorithms for each case.

In the centralized setting, time-based windows are usually more difficult to handle than sequence-based windows

window	communication	coordinator space	coordinator total time	site space	site time (per element)
infinite	$(k+s)\log n$	s	$(k+s)\log n$	1	1
sequence-based	$ks\log(W/s)$	s	$ks\log(W/s)$	1	1
time-based	$(k+s)\log W$	$s\log W$	$(k+s)\log W$	$s\log W$	1

Table 1: Our algorithms' asymptotic costs for sampling without replacement over distributed streams.

because the number of active items can vary dramatically over time. There is a space lower bound of $\Omega(s\log W)$ [10] for time-based windows while sequence-based windows only need space $O(s)$ [2, 4]. One interesting observation from Table 1 is that in the distributed setting, time-based windows turn out to be *easier* than sequence-based windows, and there is a quadratic difference (when $k \approx s$) in terms of communication, while the space bounds match those for the centralized case. The fundamental reason is that it is much harder to determine whether an element a_i has "expired" in the sequence-based case, since we do not have the global sequence number i. Meanwhile, in the time-based window case, expiration can be determined by comparing an item's timestamp to the current time. A formal proof of this hardness is given in Theorem 4.2.

As mentioned above, the distributed streaming model degenerates into the standard streaming model if we set $k = 1$ and ignore the communication aspect. When restricted to this case, our algorithms achieve the same bounds as the previous centralized streaming algorithms over infinite-streams (reservoir sampling), as well as the two sliding-window cases [2, 4]. So our algorithms generalize previous techniques with the same space/time bounds while achieving optimal communication. Note however that the space bounds in [4] are worst-case, while ours and those in [2] are probabilistic.

All our algorithms are in fact based on one simple idea, which we call *binary Bernoulli sampling*. We describe this idea on a conceptual level in Section 2. Then we present our sampling algorithms (with and without replacement) for the three cases in Table 1 in Section 3, 4, and 5, respectively.

Applications. Our results immediately yield protocols to track a number of interesting functions in the distributed streaming setting. Some of them improve (for certain parameter ranges) previous results while others are new.

Tracking the frequent items (a.k.a. the *heavy hitters*) and quantiles (approximately) over distributed streams have received a lot of attention [3, 7, 14, 16, 22]. There is a deterministic algorithm that costs $\tilde{O}(k/\epsilon)$ communication[1], which is optimal [22], where ϵ is the approximation error. On the other hand, it is well known that a random sample without replacement of size $\tilde{O}(1/\epsilon^2)$ can be used to extract these statistics with high probability. Our result immediately gives a probabilistic algorithm for tracking the heavy hitters and quantiles with communication $\tilde{O}(k+1/\epsilon^2)$, which breaks the deterministic lower bound when $k > 1/\epsilon$. However, the optimality of the sampling algorithm does not imply that it is optimal for these two problems; in fact it remains an open problem to determine the randomized communication complexity for these two problems.

If the elements in the streams are d-dimensional points, for example IP packets in the source-destination space, then a random sample of size $\tilde{O}(1/\epsilon^2)$ is an ϵ-*approximation* [20] with high probability. Such a sample allows us to approximately count the number of points in any range from a *range space* with bounded VC dimensions, such as rectangles, circles, halfspaces, etc. With our algorithm we can now track all these range-counts with communication $\tilde{O}(k+1/\epsilon^2)$. If one only needs to determine if a range is large enough, that is, contains at least a ϵ-fraction of all points, then a random sample of size $\tilde{O}(1/\epsilon)$ suffices for this purpose, and is known as an ϵ-*net* [12]. Thus our algorithm tracks an ϵ-net with communication $\tilde{O}(k+1/\epsilon)$. There are numerous other applications of random samples which we do not enumerate here.

2. BINARY BERNOULLI SAMPLING

Binary Bernoulli sampling is a way of implementing Bernoulli random sampling which makes the analysis of the cost of the various sampling protocols more convenient. In its simplest form, the method associates each element in the input, e, with a (conceptually unbounded) binary string $b(e)$. The string $b(e)$ is chosen uniformly at random: each bit is independently set to 0 or 1 with probability $\frac{1}{2}$. From this, we can extract a Bernoulli sample of items each chosen independently with probability $p = 2^{-j}$ for any (integer) j: we simply select all those items whose binary strings have a prefix of 0^j (i.e., the first j bits are all 0).

A key feature of this method is that we do not need to materialize each bit string $b(e)$ immediately. Instead, it is often sufficient to materialize a prefix of $b(e)$ to determine whether an item in the input passes some initial filter. By the principle of deferred decisions, more bits of $b(e)$ can be generated later, to break ties or to accommodate a smaller p, when needed. In what follows, we treat $b(e)$ as if it is fully defined, with the understanding that if an algorithm accesses $b(e)[i]$ that is not yet fixed, it sets the value of $b(e)[i]$ as needed by drawing a random value.

One straightforward way of using this idea to maintain a random sample of size s without replacement is to keep the s elements with the (lexicographically) smallest $b(e)$'s. Implementing this idea over k distributed streams costs communication $O(ks\log n)$: the coordinator makes sure that all the sites know the global s-th smallest $b(e)$, say τ, and a site sends in a newly arrived element e iff its $b(e)$ is smaller than this threshold; every time τ changes, the coordinator broadcasts the new τ. Standard analysis shows that τ changes $O(s\log n)$ times, hence giving the claimed communication cost. It is also easy to show that the length of each string $|b(e)|$ that we need is $O(\log n)$ with high probability to break all ties, so it fits in $O(1)$ words. However, this simple way of using binary Bernoulli sampling is far from optimal. Below we present protocols that implement this idea in a smarter way so as to achieve optimal communication.

[1]The $\tilde{O}$ notation suppresses log factors.

Algorithm 1: ISWoR(s) for site in round j

```
foreach e do
    f ← 1;
    for l = 1, ..., j do
        if b(e)[l] ≠ 0 then f ← 0
    if f=1 then send e to Coordinator
```

Algorithm 2: ISWoR(s) for coordinator in round j

```
foreach e received do
    if b(e)[j+1] = 0 then
        T_{j+1} ← T_{j+1} ∪ {e}
    else T_j ← T_j ∪ {e} ;
    if |T_{j+1}| = s then
        foreach e ∈ T_{j+1} do
            if b(e)[j+2] = 0 then
                T_{j+2} ← T_{j+2} ∪ {e} ;
                T_{j+1} ← T_{j+1} \ {e} ;
        discard T_j;
        broadcast to halve p;
        j ← j+1 and go to next round;
```

3. SAMPLING OVER AN INFINITE WINDOW

3.1 Sampling without replacement

We define the protocol ISWoR(s) (for Infinite window Sampling Without Replacement) as follows: The coordinator ensures that all sites are kept up to date with a current sampling probability p, which is a power of two. Initially, p is 1, and periodically the coordinator will broadcast to all sites to reduce p by half. We call the time while $p = 2^{-j}$ the jth round. On receiving an element e, a site tests the first j bits of $b(e)$, and reports this element to the coordinator if they are all zero.

The coordinator maintains the invariant that its sample is of size at least s wherein all items are selected with probability p (so this sample is initialized with the first s items from all streams). In fact, the coordinator maintains two subsamples in round j, denoted by T_j and T_{j+1}. On receiving a new item e sent by a site to add to the sample, the coordinator assigns the item to T_j if the $(j+1)$-th bit of $b(e)$ is 1, or to T_{j+1} if it is 0.

The coordinator proceeds until $|T_{j+1}| = s$. At this time it sends out a broadcast message to halve p, and discards T_j. The coordinator then examines bit $j+2$ of b for each item in T_{j+1} to determine whether it remains in T_{j+1} (the bit is 1), or is "promoted" to T_{j+2} (the bit is 0). Pseudo-code for the protocol as executed by each site and by the coordinator is shown in Algorithm 1 and 2 respectively.

At any moment in round j, a sample without replacement of size s can be derived from the active set of sampled items via sub-sampling: we take $T_j \cup T_{j+1}$ (so that $|T_j \cup T_{j+1}| \geq s$) and sample s items from this set without replacement. In some situations, it is desirable to ensure that the sample produced is *consistent* over time: that is, after an item is ejected from the sample it never returns to the sample. To achieve this in our setting, we can pick the s elements e with the smallest $b(e)$ values (generating extra bits of $b(e)$ as needed to break ties).

THEOREM 3.1. *The protocol* ISWoR(s) *continuously maintains a sample of size s drawn without replacement uniformly from all items in $A(t)$. The amount of communication is $O((k+s)\log n)$; the coordinator needs $O(s)$ space and $O((k+s)\log n)$ time; each site needs $O(1)$ space and $O(1)$ time per item. These bounds hold with high probability.*

PROOF. The correctness of the protocol (it draws a uniform sample with replacement) is seen most easily by imagining that $b(e)$ is generated in full at the remote site when the item is seen. We can observe that any item e reaches T_j when $b(e)$ has a prefix of j zeros, which happens with probability 2^{-j}. This is true irrespective of the round the item is first sent to the coordinator, since the decisions are made based on the bits of $b(e)$, which we can treat as being fixed. Since each $b(e)$ is chosen independently, it is easy to see that the items constituting $T_j \cup T_{j+1}$ at the coordinator in the j-th round are each picked with the same uniform probability of 2^{-j}. By the properties of the protocol, there are always at least s items in this Bernoulli sample, from which it is straightforward to subsample to pick a sample of size exactly s.

Now we analyze the various costs. For the j-th round, the amount of communication can be bounded as follows. In expectation, there is $O(s)$ communication: at the start of the round there are (in expectation) $s/2$ items in T_{j+1}, and each item sent to the coordinator is placed in T_{j+1} with probability $1/2$. We can bound the communication with high probability by analyzing the number of items received by the coordinator before a round is terminated. This can be modeled using the Binomial distribution with $p = 1/2$; certainly, after s positive outcomes (when an item is placed in T_{j+1}) are obtained, the current round must be over. Let X_m be a random variable denoting the number of heads observed after m trials. By an additive Chernoff bound, $\Pr[X_{4s} < s]$ is exponentially small in s.

The total number of rounds as a function of the total number of items in all streams, n, is bounded similarly. Observe that the protocol will terminate round j after $O(s)$ events each of which occurs with probability 2^{-j-1} for every item. In expectation, after n items we should be in round $\log_2(n/s)$. We bound the probability that we reach round $2\log n$: this is polynomially small in n by another Chernoff bound. So with high probability, there are $O(\log n)$ rounds. In each round, with high probability $O(s)$ samples are sent, and the coordinator sends $O(k)$ messages to signal the end of the round. Therefore, the communication cost is bounded by $O((k+s)\log n)$.

The coordinator's time cost is asymptotically the same as the communication. The other bounds are immediate. □

The coordinator's space and the site's space/time bounds are clearly optimal. We will show below that the communication cost is also nearly optimal. Note that the coordinator's time is at least proportional to its communication.

THEOREM 3.2. *Any protocol that maintains a sample of size s (with or without replacement) over k distributed streams needs communication $\Omega(k + s\log n)$ in expectation.*

PROOF. The number of items that will ever appear in the sample is $\Theta(s\log n)$, since the i-th element should have

Algorithm 3: ISWR(s) for site in round j

```
foreach e do
    for i = 1, ..., s do
        f ← 1;
        for l = 1, ..., j do
            if b(e)[l] ≠ 0 then f ← 0
        if f=1 then send (e, i) to Coordinator
```

Algorithm 4: ISWR(s) for coordinator in round j

```
foreach (e, i) received do
    if b_i(e) < b[i] then T[i] ← e, b[i] ← b(e);
    if ∀1 ≤ i ≤ s : b[i][j] = 0 then
        j ← j + 1;
        broadcast new j and go to the next round;
```

probability s/i of being sampled. Thus this also gives a lower bound on the communication, since these elements have to be sent to the coordinator.

To argue $\Omega(k)$ is also a lower bound, just notice that a site cannot be totally ignored since the coordinator must know whether it contains the first element. □

3.2 Sampling with replacement

The simple solution to sampling with replacement is to run the ISWoR(1) protocol in parallel s times. Naively extrapolating the above bounds indicates that the cost is $\tilde{O}(ks)$. However, this is suboptimal, and can be improved as follows.

We define a protocol ISWR(s) (Infinite window Sampling With Replacement) that uses the idea of "round sharing": effectively, it runs a modified version of the ISWoR(1) protocol in parallel s times; however, the jth round is terminated only when *every* instance has terminated the jth round. That is, in round $j \geq 0$ for each item e that arrives at a local site, the site generates s binary strings $b_1(e), \ldots, b_s(e)$. If any of these strings has a prefix of j 0s, then the item is forwarded to the coordinator, along with the index (or indices) of the successes.

The coordinator receives a sequence of items, each tagged with some index i. For each index i, the coordinator retains a single item as $T[i]$, along with its current binary string $b[i]$. During round 0, the first time that an item arrives for a particular index i, the coordinator stores it as $T[i]$. Then for each item e received in round j for index i with binary string $b(e)$, the coordinator ensures that both strings $b(e)$ and $b[i]$ have enough bits generated so that $b(e) \neq b[i]$. The two strings are interpreted as integers: if $b[i] < b(e)$, then e is discarded; else, $b(e) < b[i]$, and e replaces $T[i]$, and $b[i]$ is overwritten with $b(e)$. The jth round terminates when the jth bit of $b[i]$, $b[i][j]$ is 0 for all i. At this point, the coordinator begins the $(j+1)$-th round by informing all sites to sample with $p = 2^{-j-1}$. At any moment, the coordinator can obtain a sample of size s with replacement by reporting $T[i]$ for all $i = 1, \ldots, s$. Pseudo-code for the site and coordinator is shown in Algorithm 3 and 4 respectively.

THEOREM 3.3. *The protocol* ISWR(s) *continuously maintains a sample of size s with replacement drawn uniformly from $A(t)$. The communication is $O((k+s\log s)\log n)$ items; the coordinator needs $O(s)$ space and $O((k+s\log s)\log n)$ total time; each site needs $O(1)$ space and $O(1)$ time per item. These bounds hold with high probability.*

PROOF. For uniformity of the sampling, consider just a single value of i and the corresponding $T[i]$. The protocol described carries out the Bernoulli Binary sampling procedure to track a single sampled item. The effect is to select the item from the input whose $b(e)$, interpreted as an integer, is the least. Each item has an equal chance of attaining the least such string, since nothing specific to the item or the order in which it arrives influences this process. Therefore $T[i]$ is a uniform sample over the input.

To analyze the length of each round, observe that this is essentially a coupon collector problem. That is, conditioned on an item being selected in round j to be sent to the coordinator, it is equally likely to have been selected for any of the s samples. A round must have terminated by the time we have "collected" one item for every i. So the round terminates after $O(s\log s)$ items are received, with high probability. Here, we do not consider that the same item might be selected for multiple samples, or that the previous round may have already provided items with the required prefix to their binary string, since this only helps to reduce the cost, but does not change the asymptotic bound.

For the number of rounds, we can use a variant of the analysis of the ISWoR(s) protocol to bound: the protocol will finish round j when there are $O(s\log s)$ events which each occur with probability 2^{-j}, out of ns trials (since each item is chosen with probability p with s independent repetitions). A variant of the previous analysis indicates that there is a polynomially small chance of reaching round $2\log(ns/s\log s) = O(\log n)$. Therefore, the total communication cost is bounded by $O((k+s\log s)\log n)$, with high probability.

The other costs follow straightforwardly from the protocol description. □

4. SAMPLING FROM A SEQUENCE-BASED SLIDING WINDOW

In this and the next section, we consider sampling in a sliding window. We emphasize that the sliding window is defined on the union of the k streams, not on the individual streams.

We first consider how to sample from a sequence-based sliding window, that is, the sample is uniform from the last W elements received by the whole system. This model becomes particularly challenging in the distributed setting, due to the need to decide *when* an item in the sample expires (is no longer among the W most recent): its expiration is an implicit event defined by the arrivals of sufficiently many new items. To this end, we make use of a "threshold monitoring" protocol which can determine, for a given r, when exactly r new items have arrived. For completeness we present a simplified protocol to achieve this, based on a more general solution presented in [8].

Threshold protocol. The Threshold(r) protocol proceeds in $O(\log r)$ rounds. The protocol is initiated by a message from the coordinator to all k sites telling them to begin round 1. The coordinator should terminate the protocol when exactly r items have been observed across the k sites. Each site maintains a counter of items that have been observed but not reported to the coordinator. In round j, each site counts each arriving item. When the count reaches

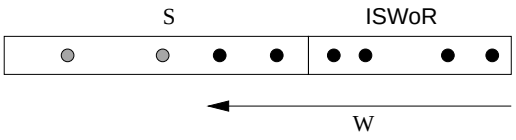

Figure 2: Schematic of SSWoR protocol for $s = 4$

(or exceeds) $\lfloor r2^{-j}/k \rfloor$, the site announces this fact to the coordinator, and reduces its counter by $\lfloor r2^{-j}/k \rfloor$. Correspondingly, the coordinator increases its global counter by $\lfloor r2^{-j}/k \rfloor$. After the coordinator receives k such messages, it starts round $j+1$ by announcing this to all sites. (Note that a round change can trigger messages from sites whose counter c is in the range $\frac{r2^{-j}}{2k} \leq c < \frac{r2^{-j}}{k}$). The protocol reaches the final round when the global counter maintained at the coordinator is in the range of $[r - O(k), r]$. In the final round, each site simply sends a message to the coordinator at every arrival of the items to increase the global counter by 1, until exactly r item arrivals have been counted. It is not difficult to see that each round requires $O(k)$ communication, so the total cost of this protocol is therefore $O(k \log r)$. The protocol correctly identifies exactly the moment when r items have arrived across all sites since the protocol was initiated.

4.1 Sampling without replacement

A simple solution to sampling from a sequence-based sliding window is periodic sampling, that is, whenever a sampled item expires, the next arriving item is sampled. This trivial predictability is not acceptable in many applications (see the discussions in e.g. [2, 4]), so usually we require that samples from disjoint windows must be independent.

Instead, we define a new protocol, SSWoR(s) (for Sequence-based window Sampling Without Replacement), which makes extensive use of the above Threshold protocol. The coordinator runs an instance of the protocol to demark every multiple of W arrivals: as soon as an instance of Threshold(W) terminates, a fresh instance is initiated. Within each such window of W, an instance of the ISWoR protocol is executed to draw a sample of size s. At the end of a window, the current sample S drawn by the ISWoR protocol is "frozen", and a new instance of the protocol is initiated. Assume for now that the coordinator can determine which items in S have "expired" (i.e. fall outside of the window of W most recent items). To draw a sample of size s without replacement, the coordinator extracts all items in S that have not expired. It then uniformly samples without replacement from the sample provided by the instance of ISWoR on the current window to make up the shortfall. Note that all these items are by definition unexpired.

Figure 2 shows a schematic view of the samples stored by the protocol for a sample size of $s = 4$. The frozen sample, S, is drawn from a window of W items. Two items (shown in gray) have expired, so a sample of non-expired items is found by taking the two remaining samples (shown in black). Two more samples are taken from the 4 samples picked by ISWoR by subsampling uniformly. When the threshold protocol indicates that the ISWoR instance has seen W total items, all the items in S will have expired, and the new sample is frozen, and forms the new S.

THEOREM 4.1. *The* SSWoR(s) *protocol continuously maintains a uniform sample of size s drawn without replacement from the W most recent items. Samples from disjoint windows are independent. The communication cost is $O(ks \log(W/s))$ per window; the coordinator needs $O(s)$ space and $O(ks \log(W/s))$ time per window; each site needs $O(1)$ space and $O(1)$ time per item. These bounds hold with high probability.*

PROOF. The correctness of the protocol, in that the resulting sample is drawn uniformly, follows from the results of Braverman *et al.* [4]. They show that given a uniform sample without replacement from a window with some expired elements, and a uniform sample of all subsequent (non-expired) elements, combining the two samples as described above for the SSWoR protocol results in a sample without replacement that is uniform from the non-expired elements only. Besides, they also show that by this method the samples from disjoint windows are independent. So the bulk of our work is in analyzing the costs of our protocol.

First, observe that the communication cost of running the ISWoR(s) protocol is $O((k+s)\log W)$ per window, so tracking the size of window with Threshold(W) at a cost of $O(k \log W)$ is dominated by this. However, the bulk of the cost of the protocol arises from deciding when each sampled item expires. The straightforward way of doing this is to initialize a separate Threshold(W) protocol whenever an item is sampled by ISWoR. An instance of the Threshold(W) protocol for an item e could be terminated prematurely if e is replaced by another item in ISWoR; else the protocol terminates normally at the time e expires. Since there are $O(s)$ running instances of the Threshold(W) protocol in the system, this requires $O(s)$ space at each site to maintain their state. Below we show how to reduce this space cost by running only one instance of the protocol at any given time.

For every item a_i sampled by ISWoR, we contact all k sites to compute its index i. Provided each site counts how many items have arrived locally, the index i for an item is the sum of all these local counts when it is sampled. When a window freezes and S is produced, the sampled items in S will start to expire. These s sampled items are stored in order of their computed indices. Based on these indices, it is possible to compute how many more items must arrive before each of them expires one by one. This can be achieved by running a Threshold(r) protocol for every sampled item with possibly a different r. Since the *order* of expiration is known in advance, it makes more sense to run these instances sequentially rather than in parallel. In this case, the parameters of each instance of Threshold(r_i), $r_1, \ldots, r_s$, satisfy $\sum_{j=1}^{s} r_j \leq W$. Thus the communication cost per window is $O(k \sum_{j=1}^{s} \log r_j) = O(ks \log \frac{W}{s})$ (and so is the coordinator's running time), which dominates the other communication costs. All the items in S must expire before the next window is frozen, so the coordinator's space is $O(s)$ and each site's space is $O(1)$.

Each site spends $O(W + s \log \frac{W}{s})$ time per window, which is $O(1 + \frac{s}{W} \log \frac{W}{s}) = O(1)$ per item. □

Although the $\tilde{O}(ks)$ communication cost may be much more than the infinite-window case, we show that this is actually the best that can be hoped for with sequence-based windows.

THEOREM 4.2. *Any (Las Vegas) protocol that maintains a sample of size s (with or without replacement) over k dis-*

tributed streams for a sequence-based sliding window of size W needs communication $\Omega(ks\log(W/ks))$ per window in expectation.

PROOF. Consider the process when the window slides from the first W items to the next W items. When the first window completes, the algorithm returns s sampled items. We will argue that the algorithm has to know the precise time when each of these s items expire. Suppose the algorithm only knows that a sampled item e expires in a time interval $[t_1, t_2]$. In order to not make a mistake by returning an expired item in the sample, it has to remove e from the sample with probability one before t_1. But if e's actual expiration time is t_2, then the probability that e is sampled within $[t_1, t_2]$ is zero, rendering the protocol incorrect.

With s sampled items in a window of size W, there must be $\Omega(s)$ pairs of adjacent items, each of which are at least $\Omega(W/s)$ items apart. This becomes $\Omega(s)$ independent threshold problems with $r = \Omega(W/s)$. A lower bound in [8] shows that any randomized algorithm (with no errors) for the threshold problem with has to communicate $\Omega(k\log\frac{r}{k})$ messages in expectation. So the total communication cost is $\Omega(ks\log\frac{W}{ks})$. □

4.2 Sampling with replacement

We define a SSWR(s) protocol (for Sequence-based window Sampling With Replacement). As in the infinite window case, the core idea is to run a protocol to sample a single item s times in parallel. Here, things are somewhat simpler than the ISWR case, because the need to track whether items have expired dominates the other costs; as a result, we do not use the round-sharing approach since it does not generate an asymptotic improvement in this case. Hence, the SSWR(s) protocol runs s instances of the SSWoR(1) protocol in parallel, which all share the same instance of Threshold(W) to determine when to freeze the current window and start a new one. The result is slightly simpler, in that each parallel instance of the protocol retains only a single item in S, and a single item from the current window, which replaces S when the item expires. It is straightforward to analyze the correctness and cost of this protocol given the above analysis, so we state without proof:

THEOREM 4.3. *The* ISWR(s) *protocol continuously maintains a uniform sample of size s drawn with replacement from the W most recent items. The communication cost is $O(ks\log(W/s))$ per window; the coordinator needs $O(s)$ space and $O(ks\log(W/s))$ time per window; each site needs $O(1)$ space and $O(1)$ time per item. These bounds hold with high probability.*

5. SAMPLING FROM A TIME-BASED SLIDING WINDOW

The case of sampling from a time-based sliding window allows reduced communication bounds. This is because the coordinator can determine when an item expires from the window directly, based on the current time and the timestamp of the item. Therefore, it is not necessary to run any instances of the Threshold protocol, resulting in a much lower cost. This stands in contrast to the centralized case, where typically time-based windows are more costly to compute over.

Nevertheless, the fact that sampling from time-based windows is less costly does not mean the problem is easier. In fact the protocols here are more complicated than the previous ones in order to achieve the (asymptotically) optimal bounds. We first provide a relatively simple protocol for sampling without replacement over a time-based sliding window, which identifies the key challenges for this model. Here, we use w to denote the duration of the sliding window in time, and n_t to denote the size of the window ending at time t, i.e., the number of items with timestamps in $[t-w, t]$.

5.1 A simple protocol

The idea is to maintain a sample of size $s' \geq s$ over a partial stream starting from some "landmark" time T. At time t, as long as there are at least s active items (namely, in the window $[t-w, t]$) among the sample of size s' that we are maintaining, then a sample of size s for the current sliding window can be obtained by sub-sampling from these items. When there are fewer than s unexpired items left in the sample, we restart the protocol.

A good value for s', on one hand, should be large enough so as to minimize the number of restarts. On the other hand, it cannot be too large since otherwise maintaining a sample of size s' will be expensive. It turns out that setting $s' = c\cdot(s+\log n_T)$ (for some constant c) strikes the right balance.

The protocol. In the protocol each site retains all active items. We first run the ISWoR(s') protocol until $t = w$. The protocol at the coordinator side then does the following.

1. Set $T = t$, and compute n_T by contacting all sites. Restart the ISWoR(s') protocol to maintain a sample of size s' from time $T - w$. This is possible since each site retains all items after time $T - w$.
2. At time t,
   (a) If there are fewer than s' items in the sample, then the coordinator must have actually collected all the items in the window $[T-w, t]$. From these, we can subsample s items from the active ones (or just report all active items if there are fewer than s in total).
   (b) Else, we check if there are at least s active items in the sample. If so, we subsample from these items to pick a sample of size s. Otherwise we go back to step 1.

The correctness of the protocol is straightforward: given two partial streams D_1, D_2 with $D_2 \subseteq D_1$, if S is a random sample in D_1, then $S \cap D_2$ is a random sample of D_2.

To bound the communication cost we need to prove that the protocol restarts $O(\log W)$ times in a window with W elements. Consider the first time T in this window when we restart. Recall that we maintain a random sample of size s' for the window $[T-w, t]$. As long as half of the items in this window have timestamps in the range $[t-w, t]$, then with high probability we will have at least s unexpired items from the s' sampled items for some constant c large enough, by a Chernoff bound. Therefore with high probability, the first restart happens when $n_1 \geq W/2$ items arriving after time $T - w$ expire, causing T to reset. Similarly, we can show that with high probability, the second restart happens when another $n_2 \geq (W - n_1)/2 \geq W/4$ items arriving after time $T - w$ expire (note that the number of items arriving after time $T - w$ is at least $W - n_1$), and so on. Therefore,

Algorithm 5: Update Level-Sampling

```
foreach e do
    l ← 0 ;
    repeat
        l ← l + 1 ;
        Insert e in queue l ;
        while queue l has > s items from levels > l do
        delete oldest item from queue l;
    until b(e)[l] = 1 ;
```

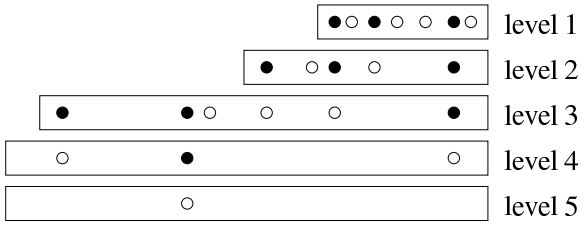

Figure 3: Example level-sampling data structure for $s = 3$

with high probability, the total number of rounds will be $O(\log W)$. So the total communication is $O((s+k)\log^2 W)$ with high probability.

However, the simple protocol above needs each site to keep all the active items. Below we show how to reduce the space cost to $\tilde{O}(s)$ at each site, which is shown to be optimal. The new protocol below also improves the communication by an $O(\log W)$ factor.

5.2 Sampling without replacement

The key challenge in the time-based sliding window setting arises because the item arrivals may not occur uniformly over time. In particular, later arrivals may be much less frequent than earlier ones. As a result, any current sample may be dominated by earlier items. When these items expire, there may be insufficiently many later items sampled to provide a sample of size s. In the above protocol, this necessitates the "restart" step, to revisit the earlier items and redraw a sufficiently large sample of them. In this section, we provide a protocol which allows the coordinator to draw a sample of sufficiently many items from the history without having to resample from past items.

We define the TSWoR(s) (Time-based window Sampling Without Replacement) protocol based on the ISWoR protocol. To keep track of the current window, an instance of the ISWoR(s) protocol is run. This is terminated after w time units, and a fresh instance begun—we refer to each such period of w as an "epoch", and we denote the start point of the most recent epoch as T. In addition, each site maintains a sample of recent items at various rates of sampling. These are kept private to the site until the end of an epoch, at which point the coordinator collects certain information from the sites about their current samples.

Level sampling at sites. For each item e that is observed at a site, the site assigns it to several "levels" based on $b(e)$. e is assigned to level $l \geq 1$ if the first $l-1$ bits of $b(e)$ are all zero. That is, if $b(e)$ has a prefix of i zeroes, it is assigned to all levels $l \leq i+1$. Note that an item is assigned to level l with probability 2^{-l+1}. The site then retains a queue for each level l consisting of the most recent items assigned to this level until either (i) at least s are also assigned to level $l+1$, or (ii) all active items at that level are retained. The update process is shown in Algorithm 5.

An example is shown in Figure 3 with $s = 3$. In the figure, circles represent items that have been sampled at particular levels, where a hollow circle is sampled at level l and a filled circle is an item that was also sampled at a higher level. Level 5 has a single item that was sampled with $p = 1/32$, while level 4 contains three items, two of which were sampled at level 4, and the one that was sampled at level 5 (so condition (ii) holds). Level 2 retains only the most recent items so that there are $s = 3$ included which were sampled at levels 3 and above (the three filled circles at level 2), meeting condition (i). The same is true at level 1.

Collection of sampled items. At the end of each epoch, the coordinator aims to collect a Level-Sampling data structure equivalent to the one resulting if one site had seen the union of all items. This is done most efficiently as a k-way merge: starting at the greatest level with any items sampled, each site sends the most recent sampled item at level l. The coordinator determines which of these is the most recent globally, adds this to its queue at level l, and prompts the corresponding site for the next most recent sampled item at level l. The current level concludes when the coordinator has either obtained all unexpired items from all sites at that level, or until at least s items have been collected that also belong to level $l+1$.

Production of a sample. At any time t, the coordinator can produce a uniform random sample of size s from the current window of duration w. The coordinator considers the level-sampling data structure of the most recent complete epoch, and identifies the level l which still has at least s non-expired items: this is guaranteed to exist by definition of the procedure (except in the extreme case when there are fewer than s non-expired items from that epoch, in which case all these items are retained). It also takes the current set of items from the instance of ISWoR which is operating in round j. Then we have a set of items A from the current epoch $[T, t]$ (Bernoulli) sampled with probability 2^{-j}, and a set of items B from $[t-w, T]$ sampled with probability 2^{-l+1}. If $j = l-1 = 0$, then it means that we have actually collected all items in the window $[t-w, t]$ and sampling will be trivial. Otherwise at least one of A and B has at least s items. Letting $\ell = \max(j, l-1)$, the coordinator selects all those items whose $b(e)$ has a prefix of ℓ 0's, so these represent a Bernoulli sample with probability $2^{-\ell}$. This results in the set C of at least s items, from which uniformly selecting s items gives the final sample.

THEOREM 5.1. *The protocol* TSWoR(s) *maintains a random sample of size s without replacement from all items with timestamps in the range* $[t-w, t]$. *The communication cost is* $O((k+s)\log W)$ *per window (where* W *is the number of items within it); the coordinator needs* $O(s \log W)$ *space and* $O((k+s)\log W)$ *time per window; each site needs* $O(s \log W)$ *space and* $O(1)$ *time per item. These bounds hold with high probability.*

PROOF. For the correctness, we claim that the result of the sampling process is to draw a Bernoulli sample C of size at least $\min(s, n_t)$ so that each item in C is from the range

$[t-w,t]$, and every item in this range selected into C with equal probability. Having established this, the fact that the resulting sample is a uniform sample without replacement of size $\min(s, n_t)$ follows easily.

To see this uniformity, consider the two epochs with unexpired items. First, each item in the current epoch (by definition, unexpired) is Bernoulli sampled by the ISWoR(s) protocol running round j with probability 2^{-j}. Meanwhile, each (unexpired) item in the previous epoch that is retained at level l is also the result of Bernoulli sampling with probability 2^{-l+1}, irrespective of which site it was observed at. The coordinator picks the level l where at least s non-expired items are retained. Such a level is guaranteed to exist based on the definition of the data structure: consider a level l whose earliest item is expired, and where the earliest item retained at level $l-1$ is not expired. By the requirements on $l-1$, it must contain at least s items which are present at level l, and these are unexpired. So there are at least s unexpired items stored at level l. The subsampling procedure then reduces the probability of sampling of whichever set (A or B) is at the higher probability, so the result set C is drawn with the same probability $2^{-\ell}$. So the probability of any unexpired item from any site to reach C is the same, $2^{-\ell}$.

For the communication cost, we have to analyze the number of items at each level in the data structure stored by the coordinator. If the coordinator collects s' items from sites to fill level l, the communication cost is $O(k+s')$ to do the k-way merge. We argue that the total number of items collected at level l is bounded with high probability. The items stored in the level sampling data structure at level l all have 0^l as a prefix of $b(e)$. The items that are sampled to level l have $b(e)[l+1]=1$, whereas the items also sampled to higher levels have $b(e)[l+1]=0$. So the probability that an item is sampled to level l, conditioned on the fact that it is sampled to level l or higher is $\frac{1}{2}$. Hence the number of items at level l is bounded by $O(s)$, with high probability. Likewise, over the W items in the epoch $[T-w,T]$, with high probability the highest level reached is $O(\log W)$. Therefore the communication cost of collected the sampled items to the coordinator at time T is $O((k+s)\log W)$, which is also the coordinator's running time in this epoch. The other costs follow straightforwardly. □

Since sampling from time-based window is more general than the infinite-window case, so the communication lower bound of Theorem 3.2 also applies here with $n=W$. A space bound of $\Omega(s\log W)$ follows from [10] since our model is more general than the centralized setting. However it is unclear if all the parties (coordinator and each site) need to pay this much space; all our previous protocols only needed $O(1)$ space on each remote site while only the coordinator has $\tilde{O}(s)$ space. Below we argue that this is not possible for time-based windows.

THEOREM 5.2. *Any (Las Vegas) protocol that maintains a sample of size s over k distributed streams for a time-based sliding window requires that each of at least $k/2$ sites must store at least $s/2$ items, unless the protocol incurs a communication cost of $\Omega(W)$.*

PROOF. We generate inputs as follows. We first send s items to site S_1. By the sampling requirement S_1 needs to send all of them to the coordinator as sampled items. Then we send the next s items to S_2. Assume S_2 does not have space to store $s/2$ items. If S_2 sends fewer than $s/2$ items to the coordinator, then some of the s items must have been discarded. Then we stop generating further items until these s items received by S_2 are the only active ones in the sliding window. This causes a failure in the sampling protocol. Otherwise we continue sending s items to $S_3, S_4, \ldots, S_k$, one by one. By the same argument, for any site that does not have $s/2$ space, it has to send at least $s/2$ elements to the coordinator. If there are more than $k/2$ such sites, then a constant fraction of the elements have been transmitted. Finally we can use the same construction in a round-robin fashion to cause $\Omega(W)$ communication in each window. □

5.3 Sampling with replacement

A naive solution to drawing a sample with replacement for a time-based sliding window is to execute the TSWoR(1) protocol s times in parallel. This is certainly correct, but would be costly, requiring $\tilde{O}(ks)$ communication. Instead, we propose to achieve (almost) the same result by more careful use of communication.

The protocol. We use again the notion of epochs to define the TSWR(s) protocol. For the current epoch, the ISWoR(s) protocol is used to maintain a sample at the coordinator of unexpired items in the range $[T,t]$. For the previous epoch, each site maintains s independent copies of the level-sampling data structure, where the ith one is based on the $b_i(e)$ values. Simply merging each of these in turn would cost $\tilde{O}(ks)$ communication. Instead, the coordinator merges all these in parallel: for each level l, it obtains the most recent sampled item from each site for any of the s instances of the data structure. It then does the k-way merge on these sequences until it has obtained the necessary samples for each of the s instances.

To form the ith sample at time t, the coordinator extracts all samples from the appropriate level of the merged data structure as B_i, at level l (note that B_i may contain more than one sampled item). It also extracts the ith sample from the ISWoR(s) protocol from round j, along with its associated binary string $b[i]$, as A. For each item in B_i, we also have its binary string $b_i(e)$. We then interpret the binary strings as integers, and pick the item e with the smallest $b_i(e)$ as the ith sampled item (ties are broken by examining longer prefixes of the bitstrings and drawing more random bits as needed).

THEOREM 5.3. *The protocol* TSWR(s) *draws a uniform sample of size s with replacement from all items with timestamps in the range $[t-w,t]$. The communication cost is $O((k+s\log s)\log W)$, and so is the coordinator's running time. The other bounds are the same as in Theorem 5.1.*

PROOF. For the communication cost, we focus on the merging procedure. For each level l, we consider all those items that were selected by any site for any sample at level l or above. Walking backwards from T, we encounter each of these items in turn, and the coordinator can stop collecting items after it has received at least one item for each i at a level above l. Each sampled item is equally likely to be for any of the s samples, and equally likely to be at level l or above level l. So, by appealing to the coupon collector's problem, the coordinator only needs to see $O(s\log s)$ sampled items until the level can be terminated, with high

probability. Similar to previous analysis, there are $O(\log W)$ levels with high probability.

To see that the samples are drawn uniformly, we adapt the argument of Theorem 3.3. For a particular value of i, the item drawn as the sample is the e in the time range $[t-w, t]$ with the lowest $b_i(e)$. By interrogating the stored data structure, the coordinator recovers a set of items with 0^l as a prefix of $b_i(e)$, and guarantees that there are no other unexpired items from that epoch with the same prefix. From the instance of $\mathsf{ISWR}(s)$, the coordinator recovers e with 0^j as a prefix of $b_i(e)$, and guarantees that there is no other e from the same epoch with a lower $b_i(e)$. The remainder of the process combines these two sets of items to find the item from $[t-w, t]$ with the smallest $b_i(e)$ is recovered as the ith sample. □

6. CONCLUSION AND OPEN PROBLEMS

In this paper we have generalized classical reservoir sampling algorithms to multiple distributed streams. To minimize communication, we have required new techniques and analysis since those for a single stream rely on information (the current total number of elements) that is inherently hard to maintain in the distributed setting.

At the end of Section 1, we mentioned a number of applications of random sampling. Random sampling indeed solves these problems, but it is unclear if it always gives the best solution. On a single stream, better algorithms are known that either do not use random sampling at all (e.g., the heavy hitter [17] and quantile problem [11]), or use some more sophisticated sampling algorithms (e.g., ϵ-approximations in bounded VC dimensions [5]). While some of these problems have been studied in this distributed streaming setting, but only the deterministic complexity has been understood [22]. It remains open to see how randomization can help reduce communication for these problems in this model.

7. REFERENCES

[1] C. Arackaparambil, J. Brody, and A. Chakrabarti. Functional monitoring without monotonicity. In *ICALP*, 2009.

[2] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, 2002.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[4] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling for sliding windows. In *PODS*, 2009.

[5] B. Chazelle. *The Discrepancy Method*. Cambridge University Press, 2000.

[6] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

[7] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*, 2005.

[8] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In *SODA*, 2008.

[9] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *ICDE*, pages 20–31, 2006.

[10] R. Gemulla and W. Lehner. Sampling time-based sliding windows in bounded space. In *SIGMOD*, pages 379–392, 2008.

[11] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, 2001.

[12] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete and Computational Geometry*, 2:127–151, 1987.

[13] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, A. D. Joseph, M. Jordan, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *IEEE INFOCOM*, 2007.

[14] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, 2006.

[15] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 2nd edition, 1981.

[16] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, 2005.

[17] A. Metwally, D. Agrawal, and A. E. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems*, 2006.

[18] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distribtuted data streams. In *SIGMOD*, 2006.

[19] I. Sharfman, A. Schuster, and D. Keren. Shape sensitive geometric monitoring. In *PODS*, 2008.

[20] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

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

[22] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *PODS*, 2009.