Improved Algorithms for Distributed Entropy Monitoring*

Jiecao Chen
Indiana University Bloomington
jiecchen@umail.iu.edu

Qin Zhang†
Indiana University Bloomington
qzhangcs@indiana.edu

Abstract

Modern data management systems often need to deal with massive, dynamic and inherently distributed data sources. We collect the data using a distributed network, and at the same time try to maintain a global view of the data at a central coordinator using a minimal amount of communication. Such applications have been captured by the distributed monitoring model which has attracted a lot of attention in recent years. In this paper we investigate the monitoring of the entropy functions, which are very useful in network monitoring applications such as detecting distributed denial-of-service attacks. Our results improve the previous best results by Arackaparambil et al. [2]. Our technical contribution also includes implementing the celebrated AMS sampling method (by Alon et al. [1]) in the distributed monitoring model, which could be of independent interest.

1 Introduction

Modern data management systems often need to deal with massive, dynamic, and inherently distributed data sources, such as packets passing through the IP backbone network, loads of machines in content delivery service systems, data collected by large-scale environmental sensor networks, etc. One of the primary goals is to detect important and/or abnormal events that have happened in networks and systems in a timely manner, while incurring a minimal amount of communication overhead. These applications led to the study of *(continuous) distributed monitoring*, which has attracted a lot of attention in the database and network communities in the past decade [3, 7, 8, 12, 13, 15, 16, 22, 23, 27–30]. The model was then formalized by Cormode, Muthukrishnan and Yi [10] in 2008, and since then considerable work has been done in theory, including tracking heavy hitters and quantiles [21, 34], entropy [2], frequency moments [10, 32], and performing random sampling [11, 31]. Some of these problems have also been studied in the sliding window settings [6, 11, 14]. We note that a closely related model, called the *distributed streams* model, has been proposed and studied earlier by Gibbons and Tirthapura [17, 18]. The distributed streams model focuses on one-shot computation, and is thus slightly different from the (continuous) distributed monitoring model considered in this paper.

In this paper we focus on monitoring the entropy functions. Entropy is one of the most important functions in distributed monitoring due to its effectiveness in detecting distributed denial-of-service attacks (the empirical entropy of IP addresses passing through a router may exhibit a noticeable change under an attack), clustering to reveal interesting patterns and performing traffic classifications [24, 25, 33] (different

*Work supported in part by NSF CCF-1525024, IIS-1633215, and IUs Office of the Vice Provost for Research through the Faculty Research Support Program.

†Corresponding author.

values of empirical entropy correspond to different traffic patterns), which are central tasks in network monitoring. Previously the entropy problem has also been studied extensively in the data stream model [4, 5, 19, 20, 25]. Arackaparambil, Brody and Chakrabarti [2] studied the Shannon and Tsallis entropy functions in the distributed *threshold* monitoring setting, where one needs to tell whether the entropy of the joint data stream $\geq \tau$ or $\leq \tau(1-\epsilon)$ for a fixed threshold τ at any time step. They obtained algorithms using $\tilde{O}(k/(\epsilon^3\tau^3))$[1] bits of communication where k is the number of sites. Note that this bound can be arbitrarily large when τ is arbitrarily small. In this paper we design algorithms that can track these two entropy functions continuously using $\tilde{O}(k/\epsilon^2 + \sqrt{k}/\epsilon^3)$ bits of communication. Our results work for all $\tau \geq 0$ simultaneously, and have improved upon the results in [2] by at least a factor of $\min\{\sqrt{k}, 1/\epsilon\}$ (ignoring logarithmic factors), even for a constant τ. Another advantage of our algorithms is that they can be easily extended to the sliding window cases (the approximation error needs to be an additive ϵ when τ is small).

As a key component of our algorithms, we show how to implement the AMS sampling, a method initially proposed by Alon, Matias and Szegedy [1] in the data stream model, in distributed monitoring. Similar as that in the data stream model, this sampling procedure can be used to track general (single-valued) functions in the distributed monitoring model, and should be of independent interest.

We note that the techniques used in our algorithms are very different from that in [2]. The algorithm in [2] monitors the changes of the values of the entropy function over time, and argue that it has to "consume" many items in the input streams for the function value to change by a factor of $\Theta(\epsilon\tau)$ (ϵ is the approximation error and τ is the threshold). We instead adapt the AMS sampling framework to the distributed monitoring model, as we will explain shortly in the technique overview. We also note that using the AMS sampling for monitoring the entropy function has already been conducted in the data stream model [5], but due to the model differences it takes quite some non-trivial efforts to port this idea to the distributed monitoring model.

In the rest of this section, we will first define the distributed monitoring model, and then give an overview of the techniques used in our algorithms.

The Distributed Monitoring Model. In the distributed monitoring model, we have k sites $S_1, \ldots, S_k$ and one central coordinator. Each site observes a stream A_i of items over time. Let $A = (a_1, \ldots, a_m) \in [n]^m$ be the joint global stream (that is, the concatenation of A_i's with items ordered by their arrival times). Each a_ℓ is observed by exactly one of the k sites at time t_ℓ, where $t_1 < t_2 < \ldots < t_m$. Let $A(t)$ be the set of items received by the system until time t, and let f be the function we would like to track. The coordinator is required to report $f(A(t))$ at *any* time step t. There is a two-way communication channel between each site and the coordinator. Our primary goal is to minimize the total bits of communication between sites and the coordinator in the whole process, since the communication cost directly links to the network bandwidth usage and energy consumption.[2] We also want to minimize the space usage and processing time per item at each site. Generally speaking, we want the total communication, space usage and processing time per item to be *sublinear* in terms of input size m and the item universe size n.

We also consider two standard *sliding window* settings, namely, the *sequence-based* sliding window and the *time-based* sliding window. In the sequence-based case, at any time step t_{now} the coordinator is required to report

$$f(A^w(t_{now})) = f(a_{L-w+1}, \ldots, a_L),$$

where w is the length of the sliding window and $L = \max\{\ell \mid t_\ell \leq t_{now}\}$. In other words, the coordinator needs to maintain the value of the function defined on the most recent w items continuously. In the time-based case the coordinator needs to maintain the function on the items that are received in the last t time

[1] $\tilde{O}(\cdot)$ ignores all polylog terms; see Table 2 for details.

[2] It is well-known that in sensor networks, communication is by far the biggest battery drain. [26]

function	type	window	approx	(total) comm.	space (site)	time (site)	ref.
Shannon	threshold	infinite	multi.	$\tilde{O}(k/(\epsilon^3\tau^3))$	$\tilde{O}(\epsilon^{-2})$	–	[2]
Tsallis	threshold	infinite	multi.	$\tilde{O}(k/(\epsilon^3\tau^3))$	$\tilde{O}(\epsilon^{-2})$	–	[2]
Shannon	continuous	infinite	multi.	$\tilde{O}(k/\epsilon^2 + \sqrt{k}/\epsilon^3)$	$\tilde{O}(\epsilon^{-2})$	$\tilde{O}(\epsilon^{-2})$	new
Shannon	continuous	sequence	mixed	$\tilde{O}(k/\epsilon^2 + \sqrt{k}/\epsilon^3)$	$\tilde{O}(\epsilon^{-2})$	$\tilde{O}(\epsilon^{-2})$	new
Shannon	continuous	time	mixed	$\tilde{O}(k/\epsilon^2 + \sqrt{k}/\epsilon^3)$	$\tilde{O}(\epsilon^{-2})$	$\tilde{O}(\epsilon^{-2})$	new
Tsallis	continuous	infinite	multi.	$\tilde{O}(k/\epsilon^2 + \sqrt{k}/\epsilon^3)$	$\tilde{O}(\epsilon^{-2})$	$\tilde{O}(\epsilon^{-2})$	new
Tsallis	continuous	sequence	mixed	$\tilde{O}(k/\epsilon^2 + \sqrt{k}/\epsilon^3)$	$\tilde{O}(\epsilon^{-2})$	$\tilde{O}(\epsilon^{-2})$	new
Tsallis	continuous	time	mixed	$\tilde{O}(k/\epsilon^2 + \sqrt{k}/\epsilon^3)$	$\tilde{O}(\epsilon^{-2})$	$\tilde{O}(\epsilon^{-2})$	new

Table 1: Summary of results. *Sequence* and *time* denote sequence-based sliding window and time-based sliding window respectively. The correctness guarantees for the sliding window cases are different from the infinite-window case: *multi.* stands for $(1+\epsilon, \delta)$-approximation; and *mixed* stands for $(1+\epsilon, \delta)$-approximation when the entropy is larger than 1, and (ϵ, δ)-approximation when the entropy is at most 1. In the threshold monitoring model, τ is the threshold value.

steps, that is, on $A^t = A(t_{now}) \backslash A(t_{now} - t)$. To differentiate we call the full stream case the *infinite window*.

Our results. In this paper we study the following two entropy functions.

- *Shannon entropy (also known as empirical entropy).* For an input sequence A of length m, the Shannon entropy is defined as $H(A) = \sum_{i \in [n]} \frac{m_i}{m} \log \frac{m}{m_i}$ where m_i is the frequency of the ith element.
- *Tsallis entropy.* The q-th $(q > 1)$ Tsallis entropy is defined as $T_q(A) = \frac{1 - \sum_{i \in [n]} (\frac{m_i}{m})^q}{q-1}$. It is known that when $q \to 1$, the q-th Tsallis entropy converges to the Shannon entropy.

We say $\hat{Q}$ is $(1+\epsilon, \delta)$-approximation of Q iff $\mathbf{Pr}[|Q - \hat{Q}| > \epsilon Q] \le \delta$, and $\hat{Q}$ is (ϵ, δ)-approximation of Q iff $\mathbf{Pr}[|Q - \hat{Q}| > \epsilon] \le \delta$. Our results are summarized in Table 1. Note that $\log(1/\delta)$ factors are absorbed in the $\tilde{O}(\cdot)$ notation.

Technique Overview. We first recall the AMS sampling method in the data stream model. Let $A = \{a_1, \ldots, a_m\} \in [n]^m$ be the stream. The AMS sampling consists of three steps: (1) pick $J \in [m]$ uniformly at random; (2) let $R = |\{j : a_j = a_J, J \le j \le m\}|$ be the frequency of the element a_J in the rest of the stream (call it a_J's *tail frequency*); and (3) set $X = f(R) - f(R-1)$. In the rest of the paper we will use $(a_J, R) \overset{\texttt{AMS}}{\sim} A$ to denote the first two steps. Given $(m_1, m_2, \ldots, m_n)$ as the frequency vector of the data stream A, letting $\bar{f}(A) = \frac{1}{m} \sum_{i \in [n]} f(m_i)$, it has been shown that $\mathbf{E}[X] = \bar{f}(A)$ [1].[3] By the standard repeat-average technique (i.e. run multiple independent copies in parallel and take the average of the outcomes), we can use sufficient (possibly polynomial in n, but for entropy this is $\tilde{O}(1/\epsilon^2)$) i.i.d. samples of X to get a $(1+\epsilon)$-approximation of $\bar{f}(A)$.

A key component of our algorithms is to implement $(a_J, R) \overset{\texttt{AMS}}{\sim} A$ in distributed monitoring. Sampling a_J can be done using a random sampling algorithm by Cormode et al. [11]. Counting R seems to be easy; however, in distributed monitoring $\Omega(m)$ bits of communication are needed if we want to keep track of R

[3]To see this, note that $\mathbf{E}[X] = \sum_{j \in [m]} \mathbf{E}[X | a_J = j] \mathbf{Pr}[a_J = j] = \sum_{j \in [m]} (\mathbf{E}[f(R) - f(R-1) | a_J = j] \cdot m_j / m)$, and $\mathbf{E}[f(R) - f(R-1) | a_J = j] = \sum_{k \in [m_j]} \frac{f(k) - f(k-1)}{m_j} = \frac{f(m_j)}{m_j}$.

exactly at any time step. One of our key observations is that a $(1+\epsilon)$-approximation of R should be enough for a big class of functions, and we can use any existing counting algorithms (e.g., the one by Huang et al. [21]) to maintain such an approximation of R. Another subtlety is that the sample a_J will change over time, and for every change we have to restart the counting process. Fortunately, we manage to bound the number of updates of a_J by $O(\log m)$.

To apply the AMS sampling approach to the Shannon entropy functions efficiently, we need to tweak the framework a bit. The main reason is that the AMS sampling method works poorly on an input that has a very frequent element, or equivalently, when the entropy of the input is close to 0. At a high level, our algorithms adapt the techniques developed by Chakrabarti et al. [5] for computing entropy in the data stream model where they track $p_{\max}$ as the empirical probability of the most frequent element $i_{\max}$, and approximate $\bar{f}(A)$ by

$$(1-p_{\max})\bar{f}(A \backslash i_{\max}) + p_{\max} \log \frac{1}{p_{\max}},$$

where for a universe element a, $A \backslash a$ is the substream of A obtained by removing all occurrences of a in A while keeping the orders of the rest of the items. But due to inherent differences between (single) data stream and distributed monitoring, quite a few specific implementations need to be reinvestigated, and the analysis is also different since in distributed monitoring we primarily care about communication instead of space. For example, it is much more complicated to track $(1-p_{\max})$ up to a $(1+\epsilon)$ approximation in distributed monitoring than in the data stream model , for which we need to assemble a set of tools developed in previous work [9, 21, 34].

Notations and conventions. We summarize the main notations in this paper in Table 2. We differentiate *item* and *element*; we use *item* to denote a token in the stream A, and *element* to denote an element from the universe $[n]$. We refer to ϵ as *approximation error*, and δ as *failure probability*.

Roadmap. In Section 2 we show how to implement the AMS sampling in the distributed monitoring model, which will be used in our entropy monitoring algorithms. We present our improved algorithms for monitoring the Shannon entropy function and the Tsallis entropy function in Section 3 and Section 4, respectively. We then conclude the paper in Section 5.

2 AMS Sampling in Distributed Monitoring

In this section we extend the AMS sampling algorithm to the distributed monitoring model. We choose to present this implementation in a general form so that it can be used for tracking both the Shannon entropy and the Tsallis entropy. We will discuss both the infinite window case and the sliding window cases.

Roadmap. We will start by introducing some tools from previous work, and then give the algorithms for the infinite window case, followed by the analysis. We then discuss the sliding window cases.

2.1 Preliminaries

Recall the AMS sampling framework sketched in the introduction. Define $\mathsf{Est}(f, R, \kappa) = \frac{1}{\kappa}\sum_{i \in [\kappa]} X_i$, where $\{X_1, \ldots, X_\kappa\}$ are i.i.d. sampled from the distribution of $X = f(R) - f(R-1)$. The following lemma shows that for a sufficiently large κ, $\mathsf{Est}(f, R, \kappa)$ is a good estimator of $\mathbf{E}[X]$.

k	number of sites
$[n]$	$[n] = \{1, 2, \ldots, n\}$, the universe
$-i$	$-i = [n] \backslash i = \{1, \ldots, i-1, i+1, \ldots, n\}$
$s \in_R S$	the process of sampling s from set S uniformly at random
$\log x, \ln x$	$\log x = \log_2 x$, $\ln x = \log_e x$
A	$A = (a_1, \ldots, a_m) \in [n]^m$ is a sequence of items
$A \backslash z$	subsequence obtained by deleting all the occurrences of z from A
m_i	$m_i = \lvert\{j : a_j = i\}\rvert$ is the frequency of element i in A
p_i	$p_i = \frac{m_i}{m}$, the empirical probability of i
$\vec{p}$	$\vec{p} = (p_1, p_2, \ldots, p_n)$
$H(A) \equiv H(\vec{p})$	$H(A) \equiv H(\vec{p}) = \sum_{i \in [n]} p_i \log p_i^{-1}$ is the Shannon entropy of A
m_{-i}	$m_{-i} = \sum_{j \in [n] \backslash i} m_j$
$\bar{f}(A)$	$\bar{f}(A) = \frac{1}{\lvert A\rvert} \sum_{i \in [n]} f(m_i)$
$H(A), f_m$	$f_m(x) = x \log \frac{m}{x}$ and $H(A) \equiv \bar{f_m}(A)$
$(1+\epsilon, \delta)$-approx.	$\hat{Q}$ is $(1+\epsilon, \delta)$-approximation of Q iff $\mathbf{Pr}[\lvert Q - \hat{Q}\rvert > \epsilon Q] \leq \delta$
$(1+\epsilon)$-approx.	simplified notation for $(1+\epsilon, 0)$-approximation
(ϵ, δ)-approx.	$\hat{Q}$ is (ϵ, δ)-approximation of Q iff $\mathbf{Pr}[\lvert Q - \hat{Q}\rvert > \epsilon] \leq \delta$
ϵ-approx.	simplified notation for $(\epsilon, 0)$-approximation
$\tilde{O}(\cdot)$	$\tilde{O}$ suppresses poly$(\log \frac{1}{\epsilon}, \log \frac{1}{\delta}, \log n, \log m)$
Est(f, R, κ)	defined in Section 2.1
$\lambda_{f,\mathcal{A}}$	see Definition 1
λ	when f and $\mathcal{A}$ are clear from context, λ is short for $\lambda_{f,\mathcal{A}}$

Table 2: List of notations

Lemma 1 ([5]) *Let $a \geq 0, b > 0$ such that $-a \leq X \leq b$, and*

$$\kappa \geq \frac{3(1 + a/\mathbf{E}[X])^2 \epsilon^{-2} \ln(2\delta^{-1})(a+b)}{(a + \mathbf{E}[X])}. \tag{1}$$

If $\mathbf{E}[X] > 0$, then $\mathsf{Est}(f, R, \kappa)$ *gives a $(1+\epsilon, \delta)$-approximation to $\mathbf{E}[X] = \bar{f}(A)$.*

We will also make use of the following tools from the previous work in distributed monitoring.

CountEachSimple. A simple (folklore) algorithm for counting the frequency of a given element in distribution monitoring is the following: Each site S_i maintains a local counter ct_i, initiated to be 1. Every time ct_i increases by a factor of $(1+\epsilon)$, S_i sends a message (say, a signal bit) to the coordinator. It is easy to see that the coordinator can always maintain a $(1+\epsilon)$-approximation of $\sum_i ct_i$, which is the frequency of the element. The total communication cost can be bounded by $O(k \cdot \log_{1+\epsilon} m) = O(k/\epsilon \cdot \log m)$ bits. The space used at each site is $O(\log m)$ bits and the processing time per item is $O(1)$. We denote this algorithm by *CountEachSimple*(e, ϵ), where e is the element whose frequency we want to track.

The pseudocode of *CountEachSimple* is presented in Appendix C.

CountEach. Huang et al. [21] proposed a randomized algorithm *CountEach* with a better performance. We summarize their main result in the following lemma.

Lemma 2 ([21]) *Given an element e,* $\text{CountEach}(e, \epsilon, \delta)$ *maintains a $(1+\epsilon, \delta)$-approximation to e's frequency at the coordinator, using $O\left((k + \frac{\sqrt{k}}{\epsilon}) \log \frac{1}{\delta} \log^2 m\right)$ bits of communication, $O(\log m \log \frac{1}{\delta})$ bits space per site and amortized $O(\log \frac{1}{\delta})$ processing time per item.*

For $V \subseteq [n]$, *CountEach*(V, ϵ, δ) maintains a $(1+\epsilon, \delta)$-approximation of $m_V = \sum_{i \in V} m_i$, the total frequencies of elements in V. Similarly, *CountEachSimple*(V, ϵ) maintains a $(1+\epsilon)$-approximation of m_V.

2.2 The Algorithms

To describe the algorithms, we need to introduce a positive "constant" λ which depends on the property of the function to be tracked. As mentioned that different from the streaming model where we can maintain R exactly, in distributed monitoring we can only maintain an approximation of S's tail frequency R. For a function f, recall that $\bar{f}(A) = \mathbf{E}[X] = \mathbf{E}[f(R) - f(R-1)]$. The observation is that, if $\hat{X} = f(\hat{R}) - f(\hat{R}-1)$ is very close to $X = f(R) - f(R-1)$ when $\hat{R}$ is close to R, then $\mathsf{Est}(f, \hat{R}, \kappa)$ will be a relatively accurate estimation of $\mathsf{Est}(f, R, \kappa)$ (hence $\mathbf{E}[X]$). To be more precise, if $\hat{R} \in \mathbb{Z}^+$ is a $(1+\epsilon)$-approximation to R, we hope $|X - \hat{X}|$ can be bounded by

$$|X - \hat{X}| \leq \lambda \cdot \epsilon \cdot X. \tag{2}$$

Unfortunately, for some functions there is no such λ. For example, let us consider $f(x) = x \log \frac{m}{x}$ (the function for the Shannon entropy) and $A = \{1, 1, \ldots, 1\}$. If (2) holds for some positive λ, we have $\mathbf{E}[|X - \hat{X}|] \leq \lambda \cdot \epsilon \cdot \mathbf{E}[X] = \lambda \cdot \epsilon \cdot \bar{f}(A) = 0$, which is clearly not possible.

To fix above issue, we can get rid of "bad inputs" (we can handle them using other techniques) by putting our discussion of λ under a restricted input class. That is, the constant λ depends on both the function f and the set of possible inputs $\mathcal{A}$. Formally, we introduce $\lambda_{f,\mathcal{A}}$ (the subscript emphasizes that λ depends on both f and $\mathcal{A}$) as following,

Definition 1 ($\lambda_{f,\mathcal{A}}$) Given a function $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ with $f(0) = 0$ and a class of inputs $\mathcal{A}$, we define $\lambda_{f,\mathcal{A}}$ be the smallest λ that satisfies the following:

- $\lambda \geq 1$,
- for any $A \in \mathcal{A}$, let $(S, R) \overset{\texttt{AMS}}{\sim} A$, for any positive number $\epsilon \leq 1/4$ and any $\hat{R}$ that is a $(1+\epsilon)$-approximation of R, we have
$$|X - \hat{X}| \leq \lambda \cdot \epsilon \cdot X, \tag{3}$$
where X and $\hat{X}$ equal $f(R) - f(R-1)$ and $f(\hat{R}) - f(\hat{R}-1)$ respectively.

When f and $\mathcal{A}$ are clear from the context, we often write $\lambda_{f,\mathcal{A}}$ as λ. λ measures the approximation error introduced by using the approximation $\hat{R}$ when estimating $\mathbf{E}[X] = \bar{f}(A)$ under the *worst-case* input $A \in \mathcal{A}$. We will see soon that the efficiency of our AMS sampling algorithm is directly related to the value of λ.

Remark 1 Directly calculating λ based on f and $\mathcal{A}$ may not be easy, but for the purpose of bounding the complexity of the AMS sampling, it suffices to calculate a relatively tight upper bound of $\lambda_{f,\mathcal{A}}$; examples can be found in Section 3.5 and Section 4 when we apply this algorithm framework to entropy functions.

We now show how to maintain a single pair $(S, \hat{R}) \overset{\texttt{AMS}}{\sim} A$ (we use $\hat{R}$ because we can only track R approximately). The algorithms are presented in Algorithm 1 and 2.

Algorithm 1: Receive an item at a site

```
1 intialize $S = \perp, r(S) = +\infty$;
2 foreach $e$ received do
3     sample $r(e) \in_R (0, 1)$;
4     if $r(e) < r(S)$ then send $(e, r(e))$ to the coordinator ;
```

Algorithm 2: Update a sample at the coordinator

```
1 foreach $(e, r(e))$ received do
2     update $S \leftarrow e, r(S) \leftarrow r(e)$;
3     restart $\hat{R} \leftarrow$ CountEachSimple$(S, \frac{\epsilon}{3\lambda})$;
4     broadcast new $(S, r(S))$ to all sites and each site updates their local copy.
```

- *Maintain* S: Similar to that in [11], we randomly associate each incoming item a with a real number [4] $r(a) \in (0, 1)$ as its rank. We maintain S to be the item with the smallest rank in $A(t)$ at any time step t. Each site also keeps a record of $r(S)$, and only sends items with ranks smaller than $r(S)$ to the coordinator. Each time S getting updated, the coordinator broadcasts the new S with its rank $r(S)$ to all the k sites.

- *Maintain* R: Once S is updated, we use *CountEachSimple*$(S, \frac{\epsilon}{3\lambda})$ to keep track of its tail frequency R up to a factor of $(1 + \frac{\epsilon}{3\lambda})$.

To present the final algorithm, we need to calculate κ, the number of copies of $(S, \hat{R}) \overset{\texttt{AMS}}{\sim} A$ we should maintain at the coordinator. Consider a fixed function f and an input class $\mathcal{A}$. Recall that in Lemma 1, a, b and $\mathbf{E}[X]$ all depend on the input stream $A \in \mathcal{A}$ because the distribution of X is determined by the input stream. To minimize the communication cost, we want to keep κ as small as possible while Inequality (1) holds for all input streams in $\mathcal{A}$. Formally, given an input stream $A \in \mathcal{A}$, we define $\pi(A)$ as

$$\begin{aligned} \underset{a,b}{\text{minimize}} \quad & \frac{3(1 + a/\mathbf{E}[X])^2(a + b)}{(a + \mathbf{E}[X])} \\ \text{subject to} \quad & a \geq 0, \\ & b > 0, \\ & -a \leq X \leq b \; (\forall X). \end{aligned} \tag{4}$$

Then κ takes the upper bound of $\epsilon^{-2} \ln(2\delta^{-1})\pi(A)$ over $A \in \mathcal{A}$, that is,

$$\kappa(\epsilon, \delta, \mathcal{A}) = \epsilon^{-2} \ln(2\delta^{-1}) \cdot \sup_{A \in \mathcal{A}} \pi(A). \tag{5}$$

One way to compute $\sup_{A \in \mathcal{A}} \pi(A)$, as we will do in the proof of Lemma 8, is to find specific values for a and b such that under arbitrary stream $A \in \mathcal{A}$, $-a \leq X \leq b$ holds for all X. We further set $E = \inf_{A \in \mathcal{A}} \mathbf{E}[X]$, then an upper bound of $\sup_{A \in \mathcal{A}} \pi(A)$ is given by $O(\frac{(1+a/E)^2(a+b)}{(a+E)})$.

[4] In practice, one can generate a random binary string of, say, $10 \log m$ bits as its rank, and w.h.p. all ranks will be different.

Our algorithm then maintains $\kappa = \kappa(\frac{\epsilon}{2}, \delta, \mathcal{A})$ copies of $(S, \hat{R}) \overset{\texttt{AMS}}{\sim} A$ at the coordinator. At each time step, the coordinator computes $\mathsf{Est}(f, \hat{R}, \kappa)$. We present the main procedure in Algorithm 3.

Algorithm 3: TrackF(ϵ, δ): Track $\bar{f}(A)$ at the coordinator

```
/* $(S, \hat{R}) \overset{\texttt{AMS}}{\sim} A$ are maintained via Algorithm 1, 2    */
1 track $\kappa(\frac{\epsilon}{2}, \delta, \mathcal{A})$ (defined in Equation (5)) copies of $(S, \hat{R}) \overset{\texttt{AMS}}{\sim} A$ in parallel;
2 return the average of all $\left(f(\hat{R}) - f(\hat{R} - 1)\right)$;
```

2.3 The Analysis

We prove the following result in this section.

Theorem 1 *For any function $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ with $f(0) = 0$ and input class $\mathcal{A}$, Algorithm 3 maintains at the coordinator a $(1 + \epsilon, \delta)$-approximation to $\bar{f}(A)$ for any $A \in \mathcal{A}$, using*

$$O\left(k/\epsilon^3 \cdot \lambda \cdot \sup_{A \in \mathcal{A}} \pi(A) \cdot \log \frac{1}{\delta} \log^2 m\right) \tag{6}$$

bits of communication, $O\left(\kappa(\frac{\epsilon}{2}, \delta, \mathcal{A}) \cdot \log m\right)$ bits space per site, and amortized $O\left(\kappa(\frac{\epsilon}{2}, \delta, \mathcal{A})\right)$ time per item, where $\pi(A), \kappa(\frac{\epsilon}{2}, \delta, \mathcal{A})$ are defined in (4) and (5) respectively.

We first show the correctness of Algorithm 3, and then analyze the costs.

Correctness. The following lemma together with the property of $\hat{R}$ gives the correctness of Algorithm 3.

Lemma 3 *For any $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ with $f(0) = 0$ and input class $\mathcal{A}$, set $\kappa = \kappa(\frac{\epsilon}{2}, \delta, \mathcal{A})$. If $\hat{R}$ is a $(1 + \frac{\epsilon}{3\lambda})$-approximation to R, then $\mathsf{Est}(f, \hat{R}, \kappa)$ is a $(1 + \epsilon, \delta)$-approximation to $\bar{f}(A), \forall A \in \mathcal{A}$.*

Proof: By Definition 1, the fact "$\hat{R}$ is a $(1 + \frac{\epsilon}{3\lambda})$-approximation to R" implies $|X - \hat{X}| \le \frac{\epsilon}{3} X$, hence

$$|\mathsf{Est}(f, R, \kappa) - \mathsf{Est}(f, \hat{R}, \kappa)| \le \frac{\epsilon}{3} \mathsf{Est}(f, R, \kappa) \tag{7}$$

By Lemma 1, our choice for κ ensures that $\mathsf{Est}(f, R, \kappa)$ is a $(1 + \epsilon/2, \delta)$-approximation to $\mathbf{E}[X]$, that is, with probability at least $1 - \delta$, we have

$$|\mathsf{Est}(f, R, \kappa) - \mathbf{E}[X]| \le \frac{\epsilon}{2} \mathbf{E}[X]. \tag{8}$$

Combining (7) and (8), we obtain

$$|\mathsf{Est}(f, \hat{R}, \kappa) - \mathbf{E}[X]| \le \left(\left(1 + \frac{\epsilon}{2}\right)\left(1 + \frac{\epsilon}{3}\right) - 1\right) \mathbf{E}[X] \le \epsilon \mathbf{E}[X].$$

We thus conclude that $\mathsf{Est}(f, \hat{R}, \kappa)$ is a $(1 + \epsilon, \delta)$-approximation to $\mathbf{E}[X]$ for any input stream $A \in \mathcal{A}$. □

Costs. By *CountEachSimple*, tracking $\hat{R}$ as a $(1+\epsilon)$-approximation to R for each sample S costs $O(\frac{k}{\epsilon}\log m)$ bits. We show in the following technical lemma (whose proof we deferred to Appendix A) that the total number of updates of S is bounded by $O(\log m)$ with high probability. Thus the total bits of communication to maintain one copy of $\hat{R}$ can be bounded by $O(\frac{k}{\epsilon}\log^2 m)$.

Lemma 4 *Let $U_1, \ldots, U_m$ be random i.i.d samples from $(0,1)$. Let $J_1 = 1$; for $i \geq 2$, let $J_i = 1$ if $U_i < \min\{U_1, \ldots, U_{i-1}\}$ and $J_i = 0$ otherwise. Let $J = \sum_{i\in[m]} J_i$. Then $J_1, J_2, \ldots, J_m$ are independent, and $\mathbf{Pr}[J > 2\log m] < m^{-1/3}$.*

We will ignore the failure probability $m^{-1/3}$ in the rest of the analysis since it is negligible in all cases we consider.

We now bound the total communication cost: we track $\kappa(\frac{\epsilon}{2}, \delta, \mathcal{A})$ (defined in Equation (5)) copies of $(S, \hat{R}) \overset{\texttt{AMS}}{\sim} A$ in parallel; and to maintain each such pair, we may restart *CountEachSimple* for $O(\log m)$ times. Recall that the communication cost of each run of *CountEachSimple* is $O(\frac{k\cdot\lambda}{\epsilon}\log m)$. The total communication cost (6) follows immediately. The space and processing time per item follows by noting the fact that maintaining each copy of $(S, \hat{R}) \overset{\texttt{AMS}}{\sim} A$ needs $O(\log m)$ bits, and each item requires $O(1)$ processing time. We are done with the proof of Theorem 1.

We can in fact use *CountEach* instead of *CountEachSimple* in Algorithm 2 to further reduce the communication cost. The idea is straightforward: we simply replace *CountEachSimple*$(S, \frac{\epsilon}{3\lambda})$ in Algorithm 2 with *CountEach*$(S, \frac{\epsilon}{3\lambda}, \frac{\delta}{2\kappa})$.

Corollary 1 *For any function $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ with $f(0) = 0$ and input class $\mathcal{A}$, there is an algorithm that maintains at the coordinator a $(1+\epsilon, \delta)$-approximation to $\bar{f}(A)$, and it uses*

$$O\left(\left(k/\epsilon^2 + \sqrt{k}/\epsilon^3\right) \cdot \lambda \cdot \sup_{A\in\mathcal{A}} \pi(A) \cdot \log\frac{1}{\delta}\log\frac{\kappa}{\delta}\log^3 m\right) \tag{9}$$

bits of communication, $O(\kappa\log m\log\frac{\kappa}{\delta})$ bits space per site, and amortized $O(\kappa\log\frac{\kappa}{\delta})$ time per item, where $\pi(A)$ and $\kappa = \kappa(\frac{\epsilon}{2}, \frac{\delta}{2}, \mathcal{A})$ are defined in (4) and (5) respectively.

Proof: We track $\kappa(\frac{\epsilon}{2}, \frac{\delta}{2}, \mathcal{A})$ copies of $\hat{R}$ at the coordinator. Each $\hat{R}$ is tracked by *CountEach*$(S, \frac{\epsilon}{3\lambda}, \frac{\delta}{2\kappa})$ so that all κ copies of $\hat{R}$ are still $(1+\frac{\epsilon}{3\lambda})$-approximation to R with probability at least $(1-\frac{\delta}{2})$. Following the same arguments as that in Lemma 3, $\mathsf{Est}(f, \hat{R}, \kappa)$ will be a $(1+\epsilon, \delta)$-approximation to $\mathbf{E}[X]$.

For the communication cost, recall that the communication cost of each *CountEach*$(S, \frac{\epsilon}{3\lambda}, \frac{\delta}{2\kappa})$ is $O((k + \frac{\sqrt{k}\lambda}{\epsilon}) \cdot \log\frac{\kappa}{\delta}\log^2 m) = O\left((k + \frac{\sqrt{k}}{\epsilon}) \cdot \lambda\log\frac{\kappa}{\delta}\log^2 m\right)$ bits (Lemma 2). Since we run κ (defined in (5)) copies of *CountEach* and each may be restarted for $O(\log m)$ times, the total communication cost is bounded by (9). Each *CountEach*$(S, \frac{\epsilon}{3\lambda})$ uses $O(\log m\log\frac{\kappa}{\delta})$ space per site and $O(\log\frac{\kappa}{\delta})$ processing time per item. We get the space and time costs immediately. □

2.4 Sequence-Based Sliding Window

In the sequence-based sliding window case, we are only interested in the last w items received by the system, denoted by $A^w(t) = \{a_j \mid j > t - w\}$.

It is easy to extend the AMS sampling step to the sliding window case. Cormode et al. [11] gave an algorithm that maintains s random samples at the coordinator in the sequence-based sliding window setting.

This algorithm can be directly used in our case by setting $s = 1$. Similar as before, when the sample S is updated, we start to track its tailing frequency R using *CountEach*. The algorithm is depicted in Algorithm 4.

Algorithm 4: TrackF-SW(ϵ, δ): Track $\bar{f}(A^w)$ in sequence-based sliding window setting

```
1 $\kappa \leftarrow \kappa(\frac{\epsilon}{2}, \frac{\delta}{2}, \mathcal{A})$;
2 use the sequence-based sliding window sampling algorithm from [11] to maintain $\kappa$ independent
  samples;
3 each sample $S$ initiates a CountEach$(S, \frac{\epsilon}{3\lambda}, \frac{\delta}{2\kappa})$ to track a $\hat{R}$. Whenever $S$ is updated, restart
  CountEach;
4 return the average of all $\left(f(\hat{R}) - f(\hat{R} - 1)\right)$ ;
```

Theorem 2 *For any function $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ with $f(0) = 0$ and input class $\mathcal{A}$, let π be defined as in (4) but with A being replaced with A^w. Let $\kappa = \kappa(\frac{\epsilon}{2}, \frac{\delta}{2}, \mathcal{A})$. There is an algorithm for the sequence-based sliding window (with window size w) that maintains at the coordinator a $(1+\epsilon, \delta)$-approximation to $\overline{f}(A^w)$, using*

$$O\left(\left(k/\epsilon^2 + \sqrt{k}/\epsilon^3\right) \cdot \lambda \cdot \sup_{A \in \mathcal{A}} \pi(A^w) \cdot \log\frac{1}{\delta} \log\frac{\kappa}{\delta} \log^3 m\right)$$

bits of communication, $O(\kappa \log m \log \frac{\kappa}{\delta})$ bits space per site, and amortized $O(\kappa \log \frac{\kappa}{\delta})$ time per item.

Proof: In [11] it is shown that $O(k \log w \log m) = O(k \log^2 m)$ bits of communication is sufficient to maintain a random sample in A^w, and each site uses $O(\log m)$ bits space and $O(1)$ processing time per item. The rest of the proof is exactly the same as Corollary 1. □

2.5 Time-Based Sliding Window

In the time-based sliding window case, we are only interested in the items received in the last t time steps, denoted by A^t.

The algorithm of tracking $\bar{f}(A^t)$ is essentially the same as that in the sequence-based sliding window case (Algorithm 4), except that in Line 2 of Algorithm 4, we use the time-based sampling algorithm from [11] instead of the sequence-based sampling algorithm. We summarize the result in the following theorem. Note that compared with Theorem 2, the only difference is the extra $\log m$ in the space per site, which is due to the extra $\log m$ factor in the sampling algorithm for the time-based sliding window in [11].

Theorem 3 *For any function $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ with $f(0) = 0$ and input class $\mathcal{A}$, let π be defined as in (4) but with A being replaced with A^t. Let $\kappa = \kappa(\frac{\epsilon}{2}, \frac{\delta}{2}, \mathcal{A})$. There is an algorithm for the time-based sliding window (with window size t) that maintains at the coordinator a $(1 + \epsilon, \delta)$-approximation to $\bar{f}(A^t)$, using*

$$O\left(\left(k/\epsilon^2 + \sqrt{k}/\epsilon^3\right) \cdot \lambda \cdot \sup_{A \in \mathcal{A}} \pi(A^t) \cdot \log\frac{1}{\delta} \log\frac{\kappa}{\delta} \log^3 m\right)$$

bits of communication, $O(\kappa \log^2 m \log \frac{\kappa}{\delta})$ bits space per site, and amortized $O(\kappa \log \frac{\kappa}{\delta})$ time per item.

3 Shannon Entropy

In the Shannon entropy function we have $f(x) = x \log \frac{m}{x}$ $(x > 0)$ and $f(0) = 0$, where $m = |A(t)|$. Let f_m denote this function. In this section, we will show that for arbitrary input $A \in [n]^m$, we can track $\bar{f_m}(A) = \sum_{i \in [n]} \frac{m_i}{m} \log \frac{m}{m_i}$ efficiently, using only $\tilde{O}\left(k/\epsilon^2 + \sqrt{k}/\epsilon^3\right)$ bits of communication. We also obtain similar results for sliding window cases.

To do this, we first show that when only considering a restricted input class $\mathcal{A}'$, $\bar{f_m}(A)$ $(A \in \mathcal{A}')$ can be tracked efficiently by directly applying the AMS framework presented in previous section. We then discuss how to track $\bar{f_m}(A)$ under arbitrary input $A \in [n]^m$.

For technical reasons, we assume $1/m \leq \delta, \epsilon \leq 1/20$ throughout this section. As mentioned, in distributed monitoring we can only maintain a $(1 + \epsilon)$-approximation of m at the coordinator using $o(m)$ bits of communication, but for the sake of simplifying the presentation, we assume that m can be maintained at the coordinator *exactly without any cost*. Appendix B explains why we can make such an assumption. The same assumption is also applied to the analysis of the Tsallis Entropy in Section 4.

Roadmap. We will again start by introducing some tools from previous work. We then define the restricted input class $\mathcal{A}'$, and give some intuition on how to track general inputs. We next give the algorithm for the infinite window case, followed by the analysis. Finally we discuss the sliding window cases.

3.1 Preliminaries

To present our algorithm for the Shannon entropy we need a few more tools from previous work.

CountAll. Yi and Zhang [34] gave a deterministic algorithm, denoted by *CountAll*(ϵ), that can be used to track the empirical probabilities of all universe elements up to an additive approximation error ϵ in distributed monitoring. We summarize their main result below.

Lemma 5 ([34]) *For any* $0 < \epsilon \leq 1$, CountAll(ϵ) *uses* $O(\frac{k}{\epsilon} \log^2 m)$ *bits of communication, such that for any element* $i \in [n]$*, it maintains at the coordinator an estimation* $\hat{p}_i$ *such that* $|\hat{p}_i - p_i| < \epsilon$*. Each site uses* $O(\frac{1}{\epsilon} \log m)$ *bits space and amortized* $O(1)$ *time per item.*

CountMin. We will also need the CountMin sketch introduced by Cormode and Muthukrishnan [9] in the streaming model. We summarize its property below.

Lemma 6 ([9]) *The* CountMin(ϵ, δ) *sketch uses* $O(\frac{1}{\epsilon} \log m \log \frac{1}{\delta})$ *bits of space in the streaming model, such that for any given element* $i \in [n]$*, it gives an estimation* $\hat{m}_i$ *of* i*'s frequency* m_i *such that* $\mathbf{Pr}[m_i \leq \hat{m}_i \leq m_i + \epsilon m_{-i}] \geq 1 - \delta$*. The processing time for each item is* $O(\log \frac{1}{\delta})$*.*

3.2 Tracking f_m Under A Restricted Class $\mathcal{A}'$

We have briefly mentioned (before Definition 1) that if we consider all possible inputs, $f_m = x \log \frac{m}{x}$ cannot be tracked efficiently by directly using our AMS sampling framework because the corresponding λ does not exist. However, if we consider another input class

$$\mathcal{A}' = \{A \in [n]^{m'} : 0 < m' \leq m, \forall i \in [n], m_i \leq 0.7m\},$$

(in other words, we consider streams with length no more than m and the frequency of each element is bounded by $0.7m$), then we can upper bound $\lambda_{f_m, \mathcal{A}'}$ by a constant.

The following two lemmas show that under input class $\mathcal{A}'$, f_m can be tracked efficiently using the AMS framework in Section 2.

Lemma 7 *Let f_m and the input class $\mathcal{A}'$ be defined above. We have $\lambda_{f_m,\mathcal{A}'} \le 10$ and $\inf_{A' \in \mathcal{A}'} \bar{f_m}(A') \ge 0.5$.*

Proof: Let $r, \hat{r} \in \mathbb{Z}^+$, where $\hat{r}$ is a $(1+\epsilon)$-approximation to r. Let $X = X(r) = f_m(r) - f_m(r-1)$ and $\hat{X} = X(\hat{r})$. Taking the derivative, $X'(r) = f'_m(r) - f'_m(r-1) = -\log\left(1 + \frac{1}{r-1}\right) < 0$, and thus $\inf X = f_m(0.7m) - f_m(0.7m-1) \overset{m \gg 1}{\approx} \log 0.7^{-1} > 0.5$.

When $r \ge 2$, we have

$$|X(r) - X(\hat{r})| \le \epsilon r \log\left(1 + \frac{1}{(1-\epsilon)r - 1}\right) \le 5\epsilon;$$

and when $r = 1$, we have $\hat{r} = r$ hence $X = \hat{X}$. Therefore $\left|X - \hat{X}\right| \le 5\epsilon \le 10 \cdot \epsilon \cdot X$ (as $\inf X > 0.5$). Consequently we can set $\lambda = 10$, and thus $\lambda_{f_m,\mathcal{A}'} = \inf\{\lambda\} \le 10$.

Next, given any $A' \in \mathcal{A}'$, we have $m_i < 0.7m$ for all $i \in [n]$, and thus $\bar{f_m}(A') = \frac{1}{|A'|}\sum_{i\in[n]} f_m(m_i) > \log 0.7^{-1} > 0.5$. □

Lemma 8 *Let f_m and $\mathcal{A}'$ be defined above. Algorithm 3 (with* CountEachSimple *in Algorithm 1 and Algorithm 2 replaced by* CountEach*) maintains a $(1+\epsilon,\delta)$-approximation to $\bar{f_m}(A)$ for any $A \in \mathcal{A}'$ at the coordinator, using $O\left((k/\epsilon^2 + \sqrt{k}/\epsilon^3) \cdot \log\frac{1}{\delta}\log^4 m\right)$ bits of communication, $O(\epsilon^{-2} \cdot \log^3 m \log\frac{1}{\delta})$ bits space per site, and amortized $O(\epsilon^{-2} \cdot \log\frac{1}{\delta}\log^2 m)$ time per item.*

Proof: This lemma is a direct result of Corollary 1. The main task to derive the stated is to bound $\sup_{A\in\mathcal{A}'} \pi(A)$. Recall that $X(R) = f_m(R) - f_m(R-1)$, as $X'(R) < 0$ we have $0.5 < X(0.7m) \le X \le X(1) = \log m$. To give an upper bound of κ, it suffices to use $a = 0, b = \log m$, and set $E = \inf_{A\in\mathcal{A}'} \bar{f_m}(A) \ge 0.5$, which gives an upper bound

$$\sup_{A\in\mathcal{A}'} \pi(A) \le \frac{3(1+a/E)^2(a+b)}{(a+E)} = O(\log m),$$

or $\kappa(\frac{\epsilon}{2}, \frac{\delta}{2}, \mathcal{A}') = O(\epsilon^{-2}\log m \log \delta^{-1})$.

Thus we maintain $\Theta(\epsilon^{-2}\log m\log\delta^{-1})$ copies of estimators at the coordinator. The lemma follows by applying Corollary 1 with the values $\lambda_{f_m,\mathcal{A}'}$ and $\sup_{A\in\mathcal{A}'}\pi(A)$ chosen above (and note $O(\log\frac{\kappa}{\delta}) = O(\log m)$). □

3.3 Intuition on Tracking f_m under $\mathcal{A} = [n]^m$

To track f_m under input class $\mathcal{A} = [n]^m$, a key observation made by Chakrabarti et al. [5] is that we can use the following expression to compute the entropy of A when the stream $A \in \mathcal{A}$ has a frequent element z (say, $p_z \ge 0.6$):

$$\begin{aligned} H(A) &= \frac{1}{m}\sum_{i=1}^{n} f_m(m_i) \\ &= (1-p_z)\mathbf{E}[X'] + p_z\log(1/p_z), \end{aligned} \tag{10}$$

where $X' = f_m(R') - f_m(R'-1)$, and $(S', R') \overset{\texttt{AMS}}{\sim} A\backslash z$. Note that $A\backslash z \in \mathcal{A}'$. Thus by Lemma 8 we can track $\mathbf{E}[X'] = \bar{f}_m(A\backslash z)$ efficiently.

We try to implement this idea in distributed monitoring. The remaining tasks are: (1) keep track of the pair (S', R') (thus X'); and (2) keep track of $(1 - p_z)$ and p_z. Compared with the streaming model [5], both tasks in distributed monitoring require some new ingredients in algorithms and analysis, which we present in Section 3.4 and Section 3.5, respectively.

3.4 The Algorithms

We first show how to maintain the pair (S', R'), p_z and $1 - p_z$ approximately.

Maintain (S', R'). As observed in [5], directly sampling (S', R') is not easy. The idea in [5] is to maintain $(S_0, R_0) \overset{\texttt{AMS}}{\sim} A$ and $(S_1, R_1) \overset{\texttt{AMS}}{\sim} A\backslash S_0$. We also keep track of the item z with $p_z \geq 0.6$, if exists. Now we can construct (S', R') as follows: if $z \neq S_0$, then $(S', R') \leftarrow (S_0, R_0)$; otherwise $(S', R') \leftarrow (S_1, R_1)$. The proof of the fact that S' is a random sample from $A\backslash z$ can be found in [5], Lemma 2.4. Algorithm 5 and 6 show how to maintain S_0, S_1. Algorithm 7 (that is, *TrackR*(ϵ, δ_1), where δ_1 will be set to $\delta/4\kappa$ in Algorithm 9) shows how to maintain $\left(1 + \frac{\epsilon}{\lambda_{f_m,\mathcal{A}'}}, \delta_1\right)$-approximations to R_0 and R_1, and consequently a $\left(1 + \frac{\epsilon}{\lambda_{f_m,\mathcal{A}'}}, \delta_1\right)$-approximation to R', which guarantees that $|X' - \hat{X}'| \leq \epsilon \cdot X'$ holds with probability at least $(1 - \delta_1)$.

Algorithm 5: Receive an item at a site (for the Shannon entropy)

```
1  initialize $S_0 = S_1 = \perp, r(S_0) = r(S_1) = +\infty$;
2  foreach $e$ received do
3      sample $r(e) \in_R (0,1)$;
4      if $e = S_0$ then
5          if $r(e) < r(S_0)$ then send the coordinator "update $(S_0, r(S_0))$ with $(e, r(e))$" ;
6      else if $e \neq S_0$ then
7          if $r(e) < r(S_0)$ then
8              send the coordinator "update $(S_1, r(S_1))$ with $(S_0, r(S_0))$";
9              send the coordinator "update $(S_0, r(S_0))$ with $(e, r(e))$";
10         else if $r(e) < r(S_1)$ then send the coordinator "update $(S_1, r(S_1))$ with $(e, r(e))$" ;
```

Algorithm 6: Update samples at the coordinator (for the Shannon entropy)

```
1  foreach message msg received do
2      execute msg: update $(S_0, r(S_0))$ and/or $(S_1, r(S_1))$ based on msg;
3      broadcast msg to all sites and request each site to execute the msg;
```

Algorithm 7: ***TrackR***(ϵ, δ_1): Maintain R_0 and R_1 at the coordinator

```
1  initialize $S_0 = S_1 = \perp$, and $r(S_0) = r(S_1) = +\infty$;
2  set $\epsilon_1 \leftarrow \frac{\epsilon}{10}$;
3  if $S_0$ is updated by the same element then restart CountEach$(S_0, \epsilon_1, \delta_1)$ ;
4  else if $S_0$ is updated by a different element then restart CountEach$(S_0, \epsilon_1, \delta_1)$ ;
5  else if $S_1$ is updated then
6      if $S_1$ is updated by $S_0$ then
7          replace the whole data structure of CountEach$(S_1, \epsilon_1, \delta_1)$ with CountEach$(S_0, \epsilon_1, \delta_1)$;
8      else restart CountEach$(S_1, \epsilon_1, \delta_1)$ ;
```

Maintain p_z and $1 - p_z$. It is easy to use *CountAll* to maintain p_z up to an additive approximation error ϵ, which is also a $(1 + O(\epsilon))$-approximation of p_z if $p_z \geq 0.6$. However, to maintain a $(1 + \epsilon)$-relative approximation error of $(1 - p_z)$ is non-trivial when $(1 - p_z)$ is very close to 0. We make use of *CountAll*, *CountEachSimple* and *CountMin* to construct an algorithm *TrackProb*(ϵ, δ), which maintains a $(1 + \epsilon, \delta)$-approximation of $(1 - p_z)$ at the coordinator when $p_z > 0.6$. We describe *TrackProb* in Algorithm 8.

Algorithm 8: ***TrackProb***(ϵ, δ): Approximate the empirical probability of a frequent element

```
1  initialize $z \leftarrow \perp$; $ct \leftarrow 0$; $\forall i \in [k], ct_i \leftarrow 0$;
   /* run the following processes in parallel:                                */
2  run CountAll$(0.01)$ ;
3  run $\gamma \leftarrow$ CountEachSimple$(-z, \epsilon/4)$ ;
4  run $\hat{m} \leftarrow$ CountEachSimple$([n], \epsilon/4)$;
5  the coordinator maintains a counter $ct$ that counts the number of items received by all sites up to the
   last update of $z$;
6  each site maintains a local CountMin$(\epsilon/4, \delta)$ sketch;
7  each site $S_i$ maintains a counter $ct_i$ that counts the number of items received at $S_i$;
   /* monitored by CountAll$(0.01)$                                            */
8  if CountAll identifies a new frequent element $e$ with $\hat{p}_e \geq 0.59$ then
9      $z \leftarrow e$. Broadcast $z$ to all sites;
10     restart $\gamma \leftarrow$ CountEachSimple$(-z, \epsilon/4)$;
11     each site $S_i$ sends its local CountMin sketch and local counter $ct_i$ to the coordinator;
12     the coordinator merges $k$ local CountMin sketches to a global CountMin, and sets $ct = \sum_{i \in [k]} ct_i$;
13 return $z$, $\hat{p}_{-z} \leftarrow \frac{ct - CountMin[z] + \gamma}{\hat{m}}$; $\hat{p}_z \leftarrow 1 - \hat{p}_{-z}$.
```

Putting Things Together. Let $(S_0, \hat{R}_0)$ and $(S_1, \hat{R}_1)$ be samples and their associated counts maintained by Algorithm 5, 6, and 7. The final algorithm for tracking $H(A)$ is depicted in Algorithm 9.

Algorithm 9: ***TrackEntropy***(ϵ, δ): Approximate the Shannon entropy

```
1 $\kappa \leftarrow 480\epsilon^{-2}\ln(4\delta^{-1})(2+\log m)$;
  /* maintain $z$, $\hat{p}_z$ and $\hat{p}_{-z}$ */
2 run TrackProb$(\epsilon/4, \delta/2)$;
  /* get $\kappa$ independent copies of $(S_0, \hat{R}_0, S_1, \hat{R}_1)$ */
3 run $\kappa$ copies of Algorithm 5, 6, and TrackR$(\epsilon/6, \delta/(4\kappa))$ in parallel;
4 if $\hat{p}_z > 0.65$ then
      /* For each copy of $(S_0, \hat{R}_0, S_1, \hat{R}_1)$, construct $\hat{R}'$ */
5     if $S_0 = z$ then $\hat{R}' \leftarrow \hat{R}_1$ ;
6     else $\hat{R}' \leftarrow \hat{R}_0$ ;
      /* $\mathsf{Est}(f_m, \hat{R}', \kappa)$ gives the average of $\kappa$ copies of $f_m(\hat{R}') - f_m(\hat{R}'-1)$ */
7     return $(1-\hat{p}_z)\mathsf{Est}(f_m, \hat{R}', \kappa) + \hat{p}_z \log(1/\hat{p}_z)$;
8 else
9     return $\mathsf{Est}(f_m, \hat{R}_0, \kappa)$
```

3.5 The Analysis

We prove the following result in this section.

Theorem 4 TrackEntropy(ϵ, δ) *maintains at the coordinator a* $(1+\epsilon, \delta)$*-approximation to the Shannon entropy, using* $O\left((k/\epsilon^2 + \sqrt{k}/\epsilon^3) \cdot \log\frac{1}{\delta}\log^5 m\right)$ *bits of communication,* $O(\epsilon^{-2} \cdot \log\frac{1}{\delta}\log^3 m)$ *bits space per site and amortized* $O(\epsilon^{-2} \cdot \log\frac{1}{\delta}\log^2 m)$ *time per item.*

We first show the correctness of *TrackEntropy*, and then analyze its costs.

Correctness. We establish the correctness by the following two lemmas. The first lemma shows that if there is a frequent element z with empirical probability $p_z \geq 0.6$, then Algorithm 8 properly maintains $1 - p_z$. The second lemma shows the correctness of Algorithm 9 for any input $A \in [n]^m$.

Lemma 9 $\hat{p}_{-z}$ *(see Line* 13 *of Algorithm 8) is a* $(1+\epsilon, \delta)$*-approximation to* $(1 - p_z)$.

Proof: Let z ($p_z \geq 0.6$) be the candidate frequent element if exists. Let $t(z)$ be the time step of the most recent update of z. At any time step, let A^0 be the substream consisting of all items received on or before $t(z)$, $ct = |A^0|$, and let A^1 be the rest of the joint stream A. Let m_z^0 and m_{-z}^0 be the frequency of element z in A^0 and the sum of frequencies of elements other than z in A^0, respectively. Similarly, let m_z^1 and m_{-z}^1 be defined for A^1.

Algorithm 8 computes $\hat{m}_{-z}^1$ as an approximation to m_{-z}^1 by *CountEachSimple*$(-z, \epsilon/4)$, and $\hat{m}$ as an approximation to m by *CountEachSimple*$([n], \epsilon/4)$, both at the coordinator. The coordinator can also extract from the global *CountMin* sketch an $\hat{m}_z^0$, which approximates m_z^0 up to an additive approximation error $\frac{\epsilon}{4}m_{-z}^0$ with probability $(1-\delta)$. At Line 13 of Algorithm 8, at any time step, the coordinator can compute

$$1 - \hat{p}_z = \hat{p}_{-z} = \frac{ct - \hat{m}_z^0 + \hat{m}_{-z}^1}{\hat{m}},$$

where $\hat{m}_z^0$ is an $(\frac{\epsilon}{4}m_{-z}^0, \delta)$-approximation of m_z^0, and $\hat{m}_{-z}^1$ and $\hat{m}$ are $(1+\epsilon/4)$-approximation of m_{-z}^1 and m, respectively.

The lemma follows by further noting that $ct = m_z^0 + m_{-z}^0$. $\square$

Lemma 10 TrackEntropy(ϵ, δ) *(Algorithm 9) correctly maintains at the coordinator a* $(1+\epsilon, \delta)$*-approximation to the empirical entropy* $H(A)$*.*

Proof: Similar to [5] Theorem 2.5, we divide our proof into two cases.

Case 1: there does not exist a z with $\hat{p}_z \leq 0.65$. We reach line 9 of Algorithm 9. The assumption that $\epsilon < 1/20$ implies $p_z < 0.7$, and thus the input stream $A \in \mathcal{A}'$. It is easy to verify that our choices of parameters satisfy the premise of Lemma 3, thus the correctness.

Case 2: there is a z with $\hat{p}_z > 0.65$. We reach line 7 of Algorithm 9. In this case we use Equation (10). The assumption that $\epsilon < 1/20$ implies $p_z > 0.6$, thus $A \backslash z \in \mathcal{A}'$. At line 3, we run *TrackR*$(\epsilon/6, \frac{\delta}{4\kappa})$ so that with probability $1 - \frac{\delta}{4}$, the κ copies X's satisfy $|X' - \hat{X}'| \leq \frac{\epsilon}{6}X'$ simultaneously by a union bound. One can verify based on (5) that our choice for κ is large enough to ensure that $\mathsf{Est}(f_m, R', \kappa)$ is a $(1+\epsilon/4, \delta/4)$-approximation to $\mathbf{E}[X']$. Applying the same argument as in Lemma 3, we have $\mathsf{Est}(f_m, \hat{R}', \kappa)$ as a $(1+\epsilon/2, \delta/2)$-approximation to $\mathbf{E}[X']$.

At line 2, *TrackProb*$(\epsilon/4, \delta/2)$ gives $(1-\hat{p}_z)$ as a $(1+\epsilon/4, \delta/2)$-approximation to $(1-p_z)$ (by Lemma 9). Further noting that when $(1-\hat{p}_z)$ is a $(1+\epsilon/4)$-approximation to $(1-p_z)$, we have

$$\begin{aligned} & \frac{|\hat{p}_z \log(1/\hat{p}_z) - p_z \log(1/p_z)|}{p_z \log(1/p_z)} \\ \leq\ & \frac{|\hat{p}_z - p_z|}{p_z \log(1/p_z)} \max_{p \in [\frac{1}{2}, 1]} \left| \frac{d(p \log 1/p)}{dp} \right| \\ \leq\ & \frac{\frac{\epsilon}{4}(1-p_z)}{p_z \log(1/p_z)} \log e \\ \leq\ & \epsilon. \end{aligned}$$

Thus $(1-\hat{p}_z)\mathsf{Est}(f_m, \hat{R}', \kappa) + \hat{p}_z \log \frac{1}{\hat{p}_z}$ is a $(1+\epsilon, \delta)$-approximation to $H(A) = (1-p_z)\mathbf{E}[X'] + p_z \log \frac{1}{p_z}$. $\square$

Communication Cost. We shall consider the extra cost introduced by the following adaptations to the general AMS Sampling framework described in Section 2: (1) we need to maintain S_0 and S_1 rather than to simply maintain S; and (2) we have a new subroutine *TrackProb*. It turns out that (1) will only introduce an additional multiplicative factor of $\log m$ to the communication, and (2) is not the dominating cost.

We first bound the total number of times *CountEach*(S_0) and *CountEach*(S_1) being restarted.

Lemma 11 *Let* C_0 *and* C_1 *be the frequency of* CountEach(S_0) *and* CountEach(S_1) *being restarted in Algorithm 7, respectively. Then* $(C_0 + C_1)$ *is bounded by* $O(\log^2 m)$ *with probability at least* $1 - m^{-1/6}$.

Proof: by Lemma 4, C_0 is bounded by $O(\log m)$ with probability at least $1 - m^{-1/3}$. Now let us focus on C_1. Suppose $n_1 < n_2 < \ldots < n_{C_0}$ are the global indices of items that update S_0. Let $b_i = r(a_{n_i})$, we have $b_1 > b_2 > \ldots > b_{C_0}$. Let A_i be the substream of $(a_{n_i}, a_{n_i+1}, \ldots a_{n_{i+1}-1})$ obtained by collecting all items that will be compared with $r(S_1)$; thus $|A_i| \leq m$ and each item in A_i is associated with a rank uniformly sampled from $(b_i, 1)$. For a fixed C_0, by Lemma 4 and a union bound we have that $C_1 < O(C_0 \log m)$ with probability at least $1 - \frac{C_0}{m^{1/3}}$. Also recall that $C_0 < 2\log m$ with probability $1 - m^{-1/3}$. Thus $C_1 = O(\log^2 m)$ with probability at least $1 - \frac{2\log m}{m^{1/3}}$. □

We now bound the total communication cost.

Lemma 12 TrackProb(ϵ, δ) *uses* $O(\frac{k}{\epsilon}\log\frac{1}{\delta}\log^3 m)$ *bits of communication.*

Proof: We show that z will be updated by at most $O(\log m)$ times. Suppose at some time step m_0 items have been processed, and $z = a$ is the frequent element. By definition, the frequency of a must satisfy $m_a > 0.58m_0$. We continue to process the incoming items, and when z is updated by another element at the moment the m_1-th item being processed, we must have $m_a < 0.42m_1$. We thus have $\frac{m_1}{m_0} \geq \frac{0.58}{0.42} = 1.38 > 1$, which means that every time z gets updated, the total number of items has increased by a factor of at least 1.38 since the last update of z. Thus the number of updates of z is bounded by $O(\log m)$.

We list the communication costs of all subroutines in *TrackProb*.

(1) *CountAll*(0.01) costs $O(k\log^2 m)$ bits;

(2) *CountEachSimple*$([n], \epsilon/4)$ costs $O(\frac{k}{\epsilon}\log m)$ bits;

(3) *CountEachSimple*$(-z, \epsilon/4)$ costs $O(\frac{k}{\epsilon}\log m)$ bits;

(4) sending k sketches of *CountMin*$(\epsilon/4, \delta)$ to the coordinator costs $O(\frac{k}{\epsilon}\log\frac{1}{\delta}\log m)$ bits;

(5) sending k local counters to the coordinator costs $O(k\log m)$ bits.

Among them, (3), (4), (5) need to be counted by $O(\log m)$ times, and thus the total communication cost is bounded by $O(\frac{k}{\epsilon}\log\frac{1}{\delta}\log^3 m)$. □

Combining Lemma 2, Lemma 10, Lemma 11 and Lemma 12, we now prove Theorem 4,

Proof: We have already showed the correctness and the communication cost. The only things left are the space and processing time per item. The processing time and space usage are dominated by those used to track $(S_0, \hat{R}_0, S_1, \hat{R}_1)$'s. So the bounds given in Lemma 8 also hold.

□

3.6 Sliding Windows

In Section 2.4 we have extended our general AMS sampling algorithm to the sequence-based sliding window case. We can apply that scheme directly to the Shannon entropy. However, the communication cost is high when the Shannon entropy of the stream is small. On the other hand, it is unclear if we can extend the technique of removing the frequent element to the sliding window case: it seems hard to maintain $(S_0^w, R_0^w) \overset{\texttt{AMS}}{\sim} A^w$ and $(S_1^w, R_1^w) \overset{\texttt{AMS}}{\sim} A^w \backslash S_0^w$ simultaneously in the sliding window using poly$(k, 1/\epsilon, \log w)$ communication, poly$(1/\epsilon, \log w)$ space per site and poly$(1/\epsilon, \log w)$ processing time per item.

By slightly adapting the idea in Section 2.4, we have the following result that may be good enough for most practical applications.

Theorem 5 *There is an algorithm that maintains $\hat{H}(A^w)$ at the coordinator as an approximation to the Shannon entropy $H(A^w)$ in the sequence-based sliding window case such that $\hat{H}(A^w)$ is a $(1+\epsilon,\delta)$-approximation to $H(A^w)$ when $H(A^w) > 1$ and an (ϵ,δ)-approximation when $H(A^w) \le 1$. The algorithm uses $O\left((k/\epsilon^2 + \sqrt{k}/\epsilon^3)\cdot \frac{1}{\delta}\log^4 m\right)$ bits of communication, $O(\epsilon^{-2}\cdot \log\frac{1}{\delta}\log^3 m)$ bits space per site and amortized $O(\epsilon^{-2}\cdot\log\frac{1}{\delta}\log^2 m)$ time per item.*

Proof: Instead of setting κ (the number of sample copies we run in parallel) as Equation (5), we simply set $\kappa = \Theta(\epsilon^{-2}\log w \log \delta^{-1})$, thus the space and time usage for each site. The correctness is easy to see: since we are allowed to have an additive approximation error ϵ (rather than $\epsilon\mathbf{E}[X]$) when $\mathbf{E}[X] \le 1$, we can replace ϵ by $\frac{\epsilon}{\mathbf{E}[X]}$ in Inequality (1) to cancel $\mathbf{E}[X]$. For the communication cost, we just replace the value of κ in Section 2.4 (defined by Equation (5)) with $\Theta(\epsilon^{-2}\log w\log\delta^{-1})$. $\square$

With the same argument we have a result for the time-based sliding window case where the window size is t.

Theorem 6 *There is an algorithm that maintains $\hat{H}(A^t)$ at the coordinator as an approximation to the Shannon entropy $H(A^t)$ in the time-based sliding window setting such that $\hat{H}(A^t)$ is a $(1+\epsilon,\delta)$-approximation to $H(A^t)$ when $H(A^t) > 1$ and an (ϵ,δ)-approximation when $H(A^t) \le 1$. The algorithm uses $O\left((k/\epsilon^2 + \sqrt{k}/\epsilon^3)\cdot \frac{1}{\delta}\log^4 m\right)$ bits of communication, $O(\epsilon^{-2}\cdot\log\frac{1}{\delta}\log^4 m)$ bits space per site and amortized $O(\epsilon^{-2}\cdot\log\frac{1}{\delta}\log^2 m)$ time per item.*

4 Tsallis Entropy

Recall that $\vec{p} = (p_1, p_2, \ldots, p_n) = (\frac{m_1}{m}, \frac{m_2}{m}, \ldots, \frac{m_n}{m})$ is the vector of empirical probabilities. The q-th Tsallis entropy of a stream A is defined as

$$T_q(\vec{p}) = \frac{1 - \sum_{i\in[n]} p_i^q}{q-1}.$$

It is well-known that when $q \to 1$, T_q converges to the Shannon entropy. In this section, we give an algorithm that continuously maintains a $(1+\epsilon,\delta)$-approximation to $T_q(\vec{p})$ for any constant $q > 1$.

Similar to the analysis for the Shannon entropy, we again assume that we can track the exact value of m at the coordinator without counting its communication cost. To apply the general AMS sampling scheme, we use $g_m(x) = x - m(\frac{x}{m})^q$, hence

$$\begin{aligned}\overline{g_m}(A) &= \frac{1}{m}\sum_{i\in[n]}\left[m_i - m\left(\frac{m_i}{m}\right)^q\right] \\ &= 1 - \sum_{i\in[n]} p_i^q = (q-1)T_q(\vec{p}).\end{aligned}$$

Let Z consist of elements in the stream A such that each $z \in Z$ has $m_z \ge 0.3m$. Thus $|Z| \le 4$. Consider the following two cases:

- $Z = \emptyset$. In this case $\overline{g_m}(A) \ge 1 - (\frac{1}{3})^{q-1}$.

- $Z \neq \emptyset$. In this case

$$\overline{g_m}(A) = \left(1 - \sum_{z \in Z} p_z\right) \overline{g_m}(A \backslash Z) + \frac{1}{m} \sum_{z \in Z} g_m(m_z)$$

  with $\overline{g_m}(A \backslash Z) \geq 1 - (\frac{1}{3})^{q-1}$.

Thus we can use the same technique as that for the Shannon entropy. That is, we can track the frequency of each element $z \in Z$ separately (at most 4 of them), and simultaneously remove all occurrences of elements in Z from A and apply the AMS sampling scheme to the substream $A \backslash Z$.

We only need to consider the input class $\mathcal{A}' = \{A \in [n]^{m'} : 0 < m' \leq m, \forall i \in [n], m_i \leq 0.3m\}$. The algorithm is essentially the same as the one for the Shannon entropy in Section 3; we thus omit its description.

Theorem 7 *There is an algorithm that maintains at the coordinator a $(1+\epsilon, \delta)$-approximation to $T_q(A)$ for any constant $q > 1$, using $O\left((k/\epsilon^2 + \sqrt{k}/\epsilon^3) \cdot \log \frac{1}{\delta} \log^4 m\right)$ bits of communication. Each site uses $O(\epsilon^{-2} \cdot \log \frac{1}{\delta} \log^2 m)$ and amortized $O(\epsilon^{-2} \cdot \log \frac{1}{\delta} \log m)$ time per item.*

Proof: The algorithm and correctness proof are essentially the same as that for the Shannon entropy in Section 3. We thus only analyze the costs.

Let us first bound the corresponding $\lambda_{g_m, \mathcal{A}'}, \sup_{A \in \mathcal{A}'} \pi(A)$ for g_m under the input class $\mathcal{A}'$:

$$E = \inf_{A' \in \mathcal{A}'} \overline{g_m}(A') = 1 - \left(\frac{1}{3}\right)^{q-1} = \Theta(1).$$

Let $h(x) = g_m(x) - g_m(x-1)$, $h'(x) < 0$. As $q > 1$,

$$\begin{aligned} |h(m)| &= g_m(m-1) \\ &= m - 1 - m\left(1 - \frac{1}{m}\right)^q \\ &= q - 1 + O(1/m). \end{aligned}$$

We thus set $a = q$. On the other hand, $h(1) \leq 1$, we thus set $b = 1$. Now $\sup_{A \in \mathcal{A}'} \pi(A) = O\left(\frac{(1+a/E)^2(a+b)}{a+E}\right) = O(1)$ (recall that q is constant).

Next, note that

$$h(x) = -m^{1-q}(x^q - (x-1)^q) + 1 \approx 1 - q\left(\frac{x}{m}\right)^{q-1}, \text{ and}$$

$$\begin{aligned} h(x) - h((1+\epsilon)x) &\approx -m^{1-q}(qx^{q-1} - q(1+\epsilon)^{q-1}x^{q-1}) \\ &\approx \epsilon(q-1)\left(\frac{x}{m}\right)^{q-1}. \end{aligned}$$

Also note if $x < 0.3m$, then $\left(\frac{x}{m}\right)^{q-1} \leq \left(\frac{1}{3}\right)^{q-1}$ and $h(x) > h(0.3m) = \Omega(1)$. Therefore for $q > 1$ we can find a large enough constant λ to make Equation (3) hold.

For the communication cost, simply plugging $\lambda_{g_m, \mathcal{A}'} = O(1)$ and $\sup_{A \in \mathcal{A}'} \pi(A) = O(1)$ to Equation (9) yields our statement. Note that we have $\kappa = \Theta(\epsilon^{-2} \log \delta^{-1})$, hence imply the space usage and the processing time per item (using Corollary 1). □

We omit the discussion on sliding windows since it is essentially the same as that in the Shannon entropy. The results are presented in Table 1.

5 Concluding Remarks

In this paper we have given improved algorithms for tracking the Shannon entropy function and the Tsallis entropy function in the distributed monitoring model. A couple of problems remain open. First, we do not know if our upper bound is tight. In [32] a lower bound of $\Omega(k/\epsilon^2)$ is given for the case when we have item deletions. It is not clear if the same lower bound will hold for the insertion-only case. Second, in the sliding window case, can we keep the approximation error to be multiplicative even when the entropy is small, or do strong lower bounds exist? The third, probably most interesting, question is that whether we can apply the AMS sampling framework to track other functions with improved performance in the distributed monitoring model? Candidate functions include Renyi entropy, f-divergence, mutual information, etc. Finally, it would be interesting to implement and test the proposed algorithms on real-world datasets, and compare them with related work competitors.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] C. Arackaparambil, J. Brody, and A. Chakrabarti. Functional monitoring without monotonicity. In *ICALP (1)*, pages 95–106, 2009.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, pages 28–39, 2003.

[4] A. Chakrabarti, K. D. Ba, and S. Muthukrishnan. Estimating entropy and entropy norm on data streams. *Internet Mathematics*, 3(1):63–78, 2006.

[5] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for estimating the entropy of a stream. *ACM Transactions on Algorithms*, 6(3), 2010.

[6] H.-L. Chan, T. W. Lam, L.-K. Lee, and H.-F. Ting. Continuous monitoring of distributed data streams over a time-based sliding window. *Algorithmica*, 62(3-4):1088–1111, 2012.

[7] G. Cormode and M. N. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24, 2005.

[8] G. Cormode, M. N. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*, pages 25–36, 2005.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[10] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In *SODA*, pages 1076–1085, 2008.

[11] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Continuous sampling from distributed streams. *J. ACM*, 59(2):10, 2012.

[12] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *ICDE*, page 57, 2006.

[13] G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *ICDE*, pages 1036–1045, 2007.

[14] G. Cormode and K. Yi. Tracking distributed aggregates over time-based sliding windows. In *SSDBM*, pages 416–430, 2012.

[15] M. Dilman and D. Raz. Efficient reactive monitoring. In *INFOCOM*, pages 1012–1019, 2001.

[16] S. Ganguly, M. N. Garofalakis, and R. Rastogi. Tracking set-expression cardinalities over continuous update streams. *VLDB J.*, 13(4):354–369, 2004.

[17] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *SPAA*, pages 281–291, 2001.

[18] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, pages 63–72, 2002.

[19] S. Guha, A. McGregor, and S. Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *SODA*, pages 733–742, 2006.

[20] N. J. A. Harvey, J. Nelson, and K. Onak. Sketching and streaming entropy via approximation theory. In *FOCS*, pages 489–498, 2008.

[21] Z. Huang, K. Yi, and Q. Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *PODS*, pages 295–306, 2012.

[22] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *Proceedings of HotNets-III*, 2004.

[23] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, pages 289–300, 2006.

[24] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *SIGCOMM*, pages 217–228, 2005.

[25] A. Lall, V. Sekar, M. Ogihara, J. J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *SIGMETRICS/Performance*, pages 145–156, 2006.

[26] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, 2003.

[27] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, pages 767–778, 2005.

[28] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, pages 563–574, 2003.

[29] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *SIGMOD*, pages 301–312, 2006.

[30] I. Sharfman, A. Schuster, and D. Keren. Shape sensitive geometric monitoring. In *PODS*, pages 301–310, 2008.

[31] S. Tirthapura and D. P. Woodruff. Optimal random sampling from distributed streams revisited. In *DISC*, pages 283–297, 2011.

[32] D. P. Woodruff and Q. Zhang. Tight bounds for distributed functional monitoring. In *STOC*, pages 941–960, 2012.

[33] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *SIGCOMM*, pages 169–180, 2005.

[34] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *PODS*, pages 167–174, 2009.

A Proof of Lemma 4

Proof: We can assume that $U_1, \ldots, U_m$ are distinct, since the event that $U_i = U_j$ for some $i \neq j$ has measure 0. Let $T_i = \min\{U_1, \ldots U_m\}$. Let OD_i denote the order of $U_1, \ldots, U_i$. Let $\Sigma_i = \{\text{Permutation of } [i]\}$. It is clear that for any $\sigma \in \Sigma_i$, $\mathbf{Pr}[\mathrm{OD}_i = \sigma \mid T_i > t] = \mathbf{Pr}[\mathrm{OD}_i = \sigma] = \frac{1}{i!}$. Since the order of $U_1, \ldots, U_i$ does not depend on the minimal value in that sequence, we have

$$\begin{aligned}\mathbf{Pr}[\mathrm{OD}_i = \sigma, T_i > t] &= \mathbf{Pr}[\mathrm{OD}_i = \sigma \mid T_i > t] \cdot \mathbf{Pr}[T_i > t] \\ &= \mathbf{Pr}[\mathrm{OD}_i = \sigma] \cdot \mathbf{Pr}[T_i > t].\end{aligned}$$

Therefore, the events $\{\mathrm{OD}_i = \sigma\}$ and $\{T_i > t\}$ are independent.

For any given $\sigma \in \Sigma_{i-1}$ and $z \in \{0, 1\}$:

$$\begin{aligned}
& \mathbf{Pr}[J_i = z \mid \mathrm{OD}_{i-1} = \sigma] \\
= & \lim_{t \to 0} \mathbf{Pr}[J_i = z, T_{i-1} > t \mid \mathrm{OD}_{i-1} = \sigma] \\
= & \lim_{t \to 0} \frac{\mathbf{Pr}[J_i = z \mid T_{i-1} > t, \mathrm{OD}_{i-1} = \sigma]}{\mathbf{Pr}[\mathrm{OD}_{i-1} = \sigma] / \mathbf{Pr}[T_{i-1} > t, \mathrm{OD}_{i-1} = \sigma]} \qquad (11) \\
= & \lim_{t \to 0} \frac{\mathbf{Pr}[J_i = z \mid T_{i-1} > t] \cdot \mathbf{Pr}[T_{i-1} > t]}{\mathbf{Pr}[\mathrm{OD}_{i-1} = \sigma] / \mathbf{Pr}[\mathrm{OD}_{i-1} = \sigma]} \qquad (12) \\
= & \lim_{t \to 0} \mathbf{Pr}[J_i = z, T_{i-1} > t] \\
= & \mathbf{Pr}[J_i = z],
\end{aligned}$$

where (11) to (12) holds because the events $\{J_i = z\}$ and $\{\mathrm{OD}_{i-1} = \sigma\}$ are conditionally independent given $\{T_{i-1} > t\}$, and the events $\{\mathrm{OD}_i = \sigma\}$ and $\{T_i > t\}$ are independent.

Therefore, J_i and OD_{i-1} are independent. Consequently, J_i is independent of $J_1, \ldots, J_{i-1}$, since the latter sequence is fully determined by OD_{i-1}.

$$\begin{aligned}\mathbf{Pr}[J_i = 1] &= \mathbf{Pr}[U_i < \min\{U_1, \ldots, U_{i-1}\}] \\ &= \int_0^1 (1-x)^{i-1} dx = \frac{1}{i}.\end{aligned}$$

Thus $\mathbf{E}[J_i] = \mathbf{Pr}[J_i = 1] = \frac{1}{i}$. By the linearity of expectation, $\mathbf{E}[J] = \sum_{i \in [m]} \frac{1}{i} \approx \log m$.

Since $J_1, \ldots, J_m$ are independent, $\mathbf{Pr}[J > 2 \log m] < m^{-1/3}$ follows from a Chernoff Bound. $\square$

B The Assumption of Tracking m Exactly

We explain here why it suffices to assume that m can be maintained at the coordinator *exactly without any cost*. First, note that we can always use *CountEachSimple* to maintain a $(1+\epsilon^2)$-approximation of m using $O(\frac{k}{\epsilon^2}\log m)$ bits of communication, which will be dominated by the cost of other parts of the algorithm for tracking the Shannon entropy. Second, the additional error introduced for the Shannon entropy by the $\epsilon^2 m$ additive error of m is negligible: let $g_x(m) = f_m(x) - f_m(x-1) = x\log\frac{m}{x} - (x-1)\log\frac{m}{x-1}$, and recall (in the proof of Lemma 7) that $X > 0.5$ under any $A \in \mathcal{A}'$. It is easy to verify that

$$\left|g_x((1\pm\epsilon^2)m) - g_x(m)\right| = O(\epsilon^2) \le O(\epsilon^2)X,$$

which is negligible compared with $|X - \hat{X}| \le O(\epsilon)X$ (the error introduced by using $\hat{R}$ to approximate R). Similar arguments also apply to X', and to the analysis of the Tsallis Entropy.

C Pseudocode for *CountEachSimple*

Algorithm 10, 11 describe how we can maintain a $(1+\epsilon)$-approximation to the frequency of element e.

Algorithm 10: Receive an item e at a site

```
1  initialize $c \leftarrow 1, ct \leftarrow 0$;
2  foreach $e$ received do
3      $ct \leftarrow ct + 1$;
4      if $ct = 1$ then
5          send a bit to the coordinator;
6      if $ct > (1+\epsilon)c$ then
7          $c \leftarrow ct$;
8          send a bit to the coordinator;
```

Algorithm 11: ***CountEachSimple***(e, ϵ) maintains c as the count at the coordinator

```
1  initialize $ct_i \leftarrow 0$ for all $i \in [k]$;
2  initialize $c \leftarrow 0$;
   /* maintain c as the count                                    */
3  while True do
4      if received a bit from site $i$ then
5          if $ct_i = 0$ then
6              $c \leftarrow c + 1$;
7              $ct_i \leftarrow 1$;
8          else
9              $c \leftarrow c + \epsilon \cdot ct_i$;
10             $ct_i \leftarrow (1+\epsilon)ct_i$;
```