Anarchists, Unite: Practical Entropy Approximation for Distributed Streams

Moshe Gabel
Technion – Israel Institute of Technology
Haifa, Israel 32000
mgabel@cs.technion.ac.il

Daniel Keren
Haifa University
Haifa, Israel 31905
dkeren@cs.haifa.ac.il

Assaf Schuster
Technion – Israel Institute of Technology
Haifa, Israel 32000
assaf@cs.technion.ac.il

ABSTRACT

Entropy is a fundamental property of data and a key metric in many scientific and engineering fields. Entropy estimation has been extensively studied, but almost always under the assumption that there is a single data stream, seen in its entirety by one node running the estimation algorithm. Multiple distributed data sources are becoming increasingly common, however, with applications in signal processing, computer science, medicine, physics, and more. Centralizing all data can be infeasible, for example in networks of battery or bandwidth limited sensors, so entropy estimation in distributed streams requires new, communication-efficient approaches.

We propose a practical communication-efficient algorithm for continuously approximating the entropy of distributed streams, with deterministic, user-defined error bounds. Unlike previous streaming methods, it supports deletions and variable-sized time-based sliding windows, while still avoiding communication when possible. Moreover, it optionally incorporates a state-of-the-art entropy sketch, allowing for both bandwidth reduction and monitoring very high dimensional problems. Finally, it provides the approximation to all nodes, rather than to a centralized location, which is important in settings such as wireless sensor networks.

Evaluation on several public datasets from real application domains shows that our adaptive algorithm can often reduce the number of messages by two orders of magnitude, compared to centralizing all data in one node.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**;
• **Mathematics of computing** → *Probability and statistics*;
• **Networks** → Network monitoring;

KEYWORDS

Distributed streams, entropy estimation, data mining

ACM Reference format:
Moshe Gabel, Daniel Keren, and Assaf Schuster. 2017. Anarchists, Unite: Practical Entropy Approximation for Distributed Streams. In *Proceedings of KDD '17, Halifax, NS, Canada, August 13-17, 2017,* 10 pages.
https://doi.org/10.1145/3097983.3098092

KDD '17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098092

1 INTRODUCTION

The *Shannon entropy* of a discrete random variable X taking values $\{x_1, \ldots, x_k\}$ is

$$\mathrm{H}(X) = -\sum_{i=1}^{k} \Pr[x_i] \ln \Pr[x_i] \quad .$$

Entropy[1], often described as a measure of disorder or information content, is widely used across many scientific and engineering disciplines, with diverse applications such as network monitoring [40, 50], time series analysis [6, 44], medicine [2, 17, 31], neuroscience [39], signal processing [42], and anomaly detection [7].

Recent years have seen an explosion in the number of devices and sensors, and with it new applications of entropy: network attack detection [3, 22], EEG monitoring [9, 31], air quality monitoring [12], physical activity detection using wearable sensors [18], sensor fusion in wireless sensor networks [46], and many more. Though new streaming approaches [13, 27] help tackle the growth in both rate and volume of incoming data, these still assume that data can be centralized – i.e., a single stream.

In the increasingly relevant setting of distributed streams, however, multiple data sources give rise to new challenges. First, entropy is highly non-linear, making accurate entropy estimation from distributed data a hard problem. Second, when input sources are distant or battery-constrained wireless devices, transmitting even small messages can be very costly in terms of battery power [26, 46] or increased latency [14]. Hence the need for *communication-efficient* approaches: algorithms that avoid sending messages.

Existing approaches to distributed entropy estimation can be impractical in these settings. *Sketching* [13, 27] reduces the size of data updates but each such update must still be communicated – impractical when the input rate is high [3] or sensors are battery powered. *Insertion-only algorithms*[2] [4, 14] can reduce the number of messages, but they do not support deletion of previously observed inputs, and so cannot estimate entropy in the recent (sliding) window. *Sampling* approaches [14] limit the number of sent messages per window, but updates are still sent in each window to maintain the sample even when entropy is unchanged. They also require large samples to accurately approximate entropy [27, 39, 41]. *Periodic sampling* also introduces delays, and the update period must be tuned to balance latency, error, and communication [14]. Finally, the probabilistic bounds of existing approaches could be unsuitable for some tasks, for instance seizure detection [31].

[1] Entropy is an overloaded term. Unless noted, we use "entropy" as shorthand for the well-known maximum likelihood estimator for Shannon entropy, detailed in Section 3.
[2] Also called the *cash register* streaming model[37]: event counts can only increase.

In short, existing approaches offer either reduced communication but no support for sliding windows, or continuous communication and probabilistic error bounds.

This Work

We propose CIDER (Communication-effIcient Distributed Entropy estimatoR), a new deterministic distributed entropy approximation algorithm that reduces communication and supports sliding windows, without losing the advantages of previous approaches. Our goal is a practical "turn-key" replacement for many kinds of distributed entropy monitoring applications that currently use centralization. What makes CIDER practical?

- It is communication-efficient: messages are sent only when entropy changes; if the changes are sufficiently small and do not violate the approximation bounds, CIDER can still avoid communication.
- It supports event deletions[3], meaning that it can monitor entropy in a sliding window of fixed or variable size, including full support for time-based windows.
- Approximation bounds are deterministic and user-defined – users set error bounds directly.
- For bandwidth-limited or very high-dimensional problems, it incorporates a state-of-the-art entropy sketch [13] to reduce message size and avoid holding explicit counters in memory.
- It provides the current entropy estimate in all nodes, rather than in just a single coordinator node. This can be crucial in settings such as wireless sensor networks [46].

In summary, CIDER is the first communication-efficient distributed entropy approximation with deterministic error bounds that fully supports deletions (specifically the strict turnstile model), fixed-size windows, and time-based windows.

We evaluate CIDER on real-world datasets representing real applications: network monitoring, load balancing, and air quality monitoring. Our results show that CIDER reduces communication by up to two orders of magnitude (compared to centralizing the data) and scales well with the number of nodes.

2 RELATED WORK

Existing work on estimating the entropy of distributed streams can be roughly divided into four types.

Sketching: Sketching approaches reduce the size of messages rather than the number of messages sent, and provide probabilistic bounds on the error. Bhuvanagiri and Ganguly [8] and Chakrabarti et al. [11] propose entropy sketches for single streams. Zhao et al. [50] approximate $x \ln x$ as a linear combination $c(x^{1+\alpha} - x^{1-\alpha})$, estimated in turn using Indyk's L_p norm sketch [29]; the resulting sketch is used to estimate the entropy of origin-destination network flows in a datacenter. Harvey et al. [27] use frequency moment sketches to estimate the Tsallis entropy, which is then used to approximate the Shannon entropy. Clifford and Cosma [13] propose a simple unbiased sketch based on random linear projections drawn from a maximally-skewed stable distribution; it requires no tuning and has near-optimal space complexity. They also provide a detailed review of entropy sketching. CIDER provides deterministic bounds, and aims to reduce the number of messages (communication-efficiency) rather than their size. It optionally incorporates a recent entropy sketch [13] to achieve similar reductions in message size (and memory requirements), at the cost of making the bounds probabilistic.

Insertion-only algorithms: Arackaparambil et al. [4] describe a round-based probabilistic approximation for distributed streams in the cash-register model. After each round, nodes update a coordinator if enough new items have arrived during the round, as determined by a distributed counting (F_1 moment) estimator; otherwise, no messages are sent. They also use an entropy sketch [27] to reduce message size.

Sampling: Sampling from distributed streams is well-studied, and Cormode provides a review [15]. Sampling requires continuous communication even if entropy does not change: some communication is incurred per window in order to maintain the sliding window [14, 15]. Moreover, sampling provides a probabilistic bound, and requires a large sample to control both variance and bias [27, 39, 41], yet communication grows approximately linearly with sample size [14].

Theoretical bounds: Communication complexity results use adversarial approaches to prove lower bounds on the number of bits or the number of messages needed to monitor entropy. Woodruff and Zhang [48] and Arackaparambil et al. [4] give such lower bounds, deterministic and probabilistic, with and without deletions. Arackaparambil et al. also use an adversarial construction to show that no nontrivial savings in communication is possible [4]. Indeed, CIDER does not and cannot guarantee communication reduction for all data. Yet we see that on several real datasets communication reduction is not only possible but substantial (Section 5). We observe that much of the existing work on distributed entropy approximation is highly theoretical, and focuses on the worst case; there appears to be little empirical evaluation on real data (with a few notable exceptions, e.g., [3, 50]).

Finally, some of the above also propose approximations for generalizations of entropy; we briefly discuss those in Section 4.7.

3 PROBLEM DEFINITION AND NOTATION

Let X be a discrete random variable taking values $\{x_1, \ldots, x_k\}$. Then its *Shannon entropy* is

$$\mathrm{H}(X) = -\sum_{i=1}^{k} \Pr[x_i] \ln \Pr[x_i]$$

(we use the natural logarithm, but other bases are also used). Since $\Pr[X_i]$ is generally unknown, a widely-used estimator is the maximum likelihood estimator [41]. given b_i observations of each X_i, and denoting by $n = \sum_{i=1}^{k} b_i$ the total number of observations, the entropy estimate is

$$\mathrm{H}_{\mathrm{MLE}}(P) = -\sum_{i=1}^{k} p_i \ln p_i \quad ,$$

where $p_i = b_i/n$ and P is the vector $[p_1, p_2, \ldots, p_k]$. For a continuous X, the widely used *histogram estimator* is computed in the same way: divide the range of X to k bins and define $\{b_i\}_{i=1}^{k}$ to be the

[3] The *strict turnstile* streaming model: counts go up and down, but are never negative.

number of observations in each respective bin, $n = \sum_{i=1}^{k} b_i$, and $p_i = b_i/n$ (we also define $p_i \ln p_i = 0$ if $p_i = 0$).

In the continuous distributed monitoring model [14], there are m nodes, each observing an infinite stream of events $\{(t, i)\}$, where t is the arrival time, and $i \in \{1 \dots k\}$ is observed event: X_i or a value in bin i. Each node j maintains a sliding window holding the last n^j observations seen in the window: in *fixed-size* windows the number of observations n^j is fixed, while in *time-based* windows this number changes dynamically and is determined by the number of observations seen during the window interval. For example, n^j for a five-minute window is the number of observations that arrived in the last five minutes, which can be two or two million.

Let the *local count* b_i^j be the number of observations of type i (X_i or bin i) in the local sliding window of node j, with *local size* $n^j = \sum_{j=1}^{m} b^j$. Note that b_i^j's can increase or decrease as observations enter or exit the window, but can never be negative[4]. We also define the *local probability vector* of node j, $P^j = [p_1^j, p_2^j, \dots, p_k^j]$, where $p_i^j = b_i^j/n^j$. The global sliding window is the union of all local sliding windows. It contains $n = \sum_{j=1}^{m} n^j$ observations, with *global counts* $b_i = \sum_{j=1}^{m} b_i^j$. We denote by P the *global probability vector* $P = [p_1, \dots, p_k]$ where $p_k = b_i/n$.

We aim to maintain an arbitrarily accurate approximation of $\mathrm{H_{MLE}}(P) = -\sum_{i=1}^{k} p_i \ln p_i$. At any time, all nodes must provide h_0: an approximate value of $H_{\mathrm{MLE}}(P)$, such that $LB(h_0) < H_{\mathrm{MLE}}(P) < UB(h_0)$, where $LB(h_0)$ and $UB(h_0)$ are user-defined error bound functions set by the user, and h_0 is identical at all nodes.

This formulation can express any additive or multiplicative approximation, with relative or absolute error, by appropriately defining the functions LB and UB. For example, to maintain a (1 ± 0.2) approximation, we define $LB(h_0) = h_0 \times 0.8$ and $UB(h_0) = h_0 \times 1.2$. We do require that $LB(h_0) < H_{\mathrm{MLE}}(P) < UB(h_0)$.

4 THE CIDER ESTIMATOR

Our basic strategy is to convert the entropy approximation problem to a threshold monitoring problem of the form $\mathrm{H_{MLE}}(P) < T_L$ or $\mathrm{H_{MLE}}(P) > T_U$. Nodes initially forward their local probabilities to a *coordinator* node, which computes the current $h_0 = \mathrm{H_{MLE}}(P)$, and thresholds $T_L = LB(h_0)$, $T_H = UB(h_0)$; we call this procedure *synchronization*. We then proceed with the following basic algorithm, which guarantees that h_0 is close to the current value $\mathrm{H_{MLE}}(P)$:

(1) As long as $T_L \le \mathrm{H_{MLE}}(P) \le T_U$: output approximation h_0 (Alg. 1).
(2) Otherwise, *sync* and continue with the new h_0, T_L, and T_H (Alg. 2 or Alg. 3 from Section 4.6).

By using h_0 as the estimate and continuously monitoring for threshold crossings, we provide an accurate approximation to $\mathrm{H_{MLE}}(P)$. Since h_0 is constant, the estimate is identical at all nodes. For example, if h_0 is currently 3.5 and we aim to maintain a (1 ± 0.2) approximation, we set $T_L = h_0 \times 0.8 = 2.8$ and $T_U = h_0 \times 1.2 = 4.2$. Now suppose that we later detect a threshold violation. After synchronizing, we see that $\mathrm{H_{MLE}}(P)$ is now 4.3. We therefore update $h_0 = 4.3$, $T_L = 3.44$, $T_U = 5.16$, and resume monitoring.

[4]Hence this is the strict turnstile streaming model [37].

Algorithm 1: Node j when events enter or exit the window.

1 increase (or decrease) counts n^j, b_i^j
2 update local vector: $P^j \leftarrow \frac{B}{n^j}$
3 re-weight slack: $S^j \leftarrow \frac{n_0^j}{n^j} S_0^j$
4 **if** $P^j - S^j \notin C_L$ **or** $P^j - S^j \notin C_U$ **then**
5 report safe zone violation and P^j, n^j to coordinator
6 wait for response with updated h_0, C_L, C_H, S_0^j, and/or n_0^j

Algorithm 2: Coordinator violation resolution (eager sync).

1 poll nodes for their P^j, n_j
2 compute updated h_0, C_L, C_H, S_0^j, n_0^j and send to nodes

We have reduced the problem to a simpler one: given h_0 and thresholds $T_L = LB(h_0)$, $T_H = UB(h_0)$, we need to detect whenever the global entropy $\mathrm{H_{MLE}}(P)$ crosses below T_L or above T_U. *Geometric monitoring* [32, 33] is a recent framework for communication-efficient monitoring of such threshold queries. In geometric monitoring, we convert the thresholds to constraints on the global probabilities, and decompose these global constraints to local constraints on the individual input streams that can be tested independently at each node. Nodes only communicate when local constraints are violated, in which case the coordinator can resolve the violation, for example by synchronizing as above. Geometric monitoring and its variants [24, 25, 35, 36, 43, 45] have previously been applied for datacenter monitoring [20], least-squares regression [21], privacy-preserving data mining [19], graph mining [49], linear classifiers [30], and more. See [34] for a recent review.

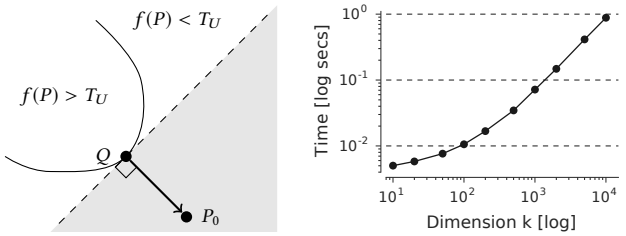
Sections 4.1 and 4.2 describe geometric monitoring (GM) and apply it to the entropy problem. Entropy monitoring, however, poses a unique challenge when using GM, which we address in Section 4.3. We also provide an entropy sketch version for high dimensional problems to reduce both bandwidth and memory (Section 4.4). To support estimating entropy of time-based windows, Section 4.5 extends the GM formulation to this setting using a new slack re-weighting scheme. Section 4.6 describes an alternative to sync on all violation, extends it to the new time-based formulation, and proposes a new balancing strategy. Finally, we briefly discuss estimating generalizations of entropy (Section 4.7).

The resulting CIDER estimator is detailed in Alg. 1, 2, and 3.

4.1 Basic Geometric Monitoring

Let f be the function we wish to monitor for threshold crossings. In our case, $f(P) = \mathrm{H_{MLE}}(P) = -\sum_{i=1}^{k} p_i \ln p_i$. Let P_0 be the *reference point*, the value of P during last sync, and similarly P_0^j denotes the value of P^j during last sync. Finally, define weights $\alpha^j = \frac{n^j}{n}$ and the weights during last sync $\alpha_0^j = \frac{n_0^j}{n_0}$. We initially assume that each sliding window has a fixed, constant size: $n^j = n_0^j$, $n = n_0$ and $\alpha_0^j = \alpha^j$. Section 4.5 extends the formulation to time-based sliding windows where n^j's change over time and global n is unknown.

Applying geometric monitoring can be distilled to two steps. In the first, we show how to compute $f(P)$ from the weighted mean



(a) Tangent half-plane at Q. **(b) Time to find Q with $f = \mathrm{H_{MLE}}$.**

Figure 1: Safe zone for upper threshold. (a) shows the tangent plane at Q. The shaded area is the convex safe zone C_U. (b) Optimization time to find closest point Q.

of local vectors P^j at each node. In our case, the global probability vector $P = \sum_{j=1}^{m} \alpha^j p^j$ since $p_i = \frac{1}{n}\sum_j b_i^j = \frac{1}{n}\sum_j n^j p_i^j = \sum_j \alpha_i^j p_i^j$.

In the second step we convert each threshold constraint to a convex set of probability vectors called the *safe zone*, such that if local probabilities lie inside the safe zone, the constraint is satisfied. Setting convex constraints in the function domain is the key insight in geometric monitoring [34, 47]: if all local vectors lie inside a convex set, then their weighted mean (i.e., the global vector) must also lie inside it, and since entropy is a function of this weighted mean, it follows that the threshold constraint is satisfied. Formally, for any convex safe zone C: $\forall j : P^j \in C \implies P = \sum_j \alpha_j P^j \in C$. Local constraint violations are possible even when the global aggregate vector is still inside the safe zone; these spurious local violations may incur extra synchronizations. However, the reverse is impossible: if $f(P)$ crosses the threshold at any point, at least one local $f(P^j)$ also crosses it. Since there can be no missed violations of the global thresholds, our approximation bounds are guaranteed.

The following lemma shows how to derive convex safe zones for the two thresholds for any concave function.

Lemma 4.1 (Safe Zones for Concave Approximations). *Let P^j and $P = \sum_{j=1}^{m} \alpha^j p^j$ be the current local and global vectors, P_0^j and P_0 their values during the last sync, and assume that $T_L \le f(P_0) \le T_U$. Then for a concave function $f(P)$:*

Lower Threshold: *The optimal convex safe zone for the lower threshold is $C_L = \{P^j : f(P^j) \ge T_L\}$, and $\forall j : P^j \in C_L \implies f(P) \ge T_L$.*

Upper Threshold: *The optimal convex safe zone for the upper threshold is $C_U = \{P^j : \langle P_0 - Q, P^j - Q\rangle \ge 0\}$, where $\langle\rangle$ denotes the inner product and Q is the point on the surface $f(P) = T_U$ closest to P_0. It guarantees: $\forall j : P^j \in C_H \implies f(P) \le T_U$.*

Closest Point: *Q can be found via convex optimization or with a closed-form solution.*

Proof. Lower Threshold: We aim to find a convex local constraint C_L such that $\forall j : P^j \in C_L \implies f(P) \ge T_L$. Since f is concave, the set $f(P) \ge T_L$ is convex, so we can simply use $f(P^j) \ge T_L$ as our local constraint:

$$C_L = \{P^j : f(P^j) \ge T_L\} \quad .$$

C_L is optimal as it contains all points that satisfy the constraint. Since C_L is convex, $\forall j : P^j \in C_L \implies P \in C_L \implies f(P) \ge T_L$.

Upper Threshold: We aim to find a convex local constraint C_U such that $\forall j : P^j \in C_U \implies f(P) \le T_U$. Since f is concave, the set $f(P) \le T_U$ is not convex, but its complement $f(P) > T_U$ is convex. Let Q be the closest point to P_0 on the surface $f(P) = T_U$. $P_0 - Q$ is perpendicular to this convex surface; thus it is the normal to the tangent plane to f at Q (its direction is equal to the gradient at Q). This tangent plane is the boundary of the optimal convex subset [36, Theorem 3] so the local constraint is:

$$C_U = \{P^j : \langle P_0 - Q, P^j - Q\rangle \ge 0\} \quad .$$

Since $C_U \subset \{P^j : f(P) \le T_U\}$ and is convex, it follows that $\forall j : P^j \in C_U \implies f(P) \le T_U$.

Closest Point: Q is the closest point on the surface $f(P) = T_U$ to the reference point P_0. Finding the closest point on a convex surface or set is a well-studied problem, and for many functions very efficient and even closed-form solutions are known. Here we limit ourselves to showing that this is a convex optimization problem:

$$Q = \mathrm{argmin}_P \|P - P_0\|^2 \quad \text{s.t.} \quad T_U - f(P) \le 0$$

Since f is concave, the above constraint $f(P) \ge T_U$ is convex and guarantees the minimizer Q will have $f(Q) = T_U$. Both objective and constraint are convex, and therefore this is always a convex minimization problem[5]. □

Convex Functions. The equivalent lemma for convex f is symmetric: the upper safe zone becomes $C_U = \{P^j : f(P^j) \le T_U\}$, while the lower safe zone is $C_L = \{P^j : \langle P_0 - Q, P^j - Q\rangle \ge 0\}$, where Q is the closest point to P_0 on the surface $f(P) = T_L$.

Entropy Approximation. We can finally apply Lemma 4.1 to derive local safe zones for $f = \mathrm{H_{MLE}}$. Recall that the entropy function $f(P) = -\sum_{i=1}^{k} p_i \ln p_i$ is a concave function in P. We also need to add a second constraint to the optimization problem, $sum_{i=1}^{k} p_i = 1$, to ensure that the probabilities in Q sum to 1. This is a convex constraint, and so the problem remains a convex minimization problem. Figure 1a demonstrates the shape of the lower safe zone. Figure 1b shows the time it takes to find Q with different values of k on an Intel Core i7-4500U CPU running at 1.80GHz, using the CVXPY convex optimization package [16].

Binary Search For Tangent Plane. When k is too high or if the coordinator is processing-limited, we replace the convex optimization procedure with a heuristic: denote by $P_{\max}$ the point with highest entropy $[1/k, \ldots, 1/k]$, and use the tangent plane at $\tilde{Q}$, defined as the point where the surface $f(P) = T_U$ intersects with the segment $\overline{P_0 P_{\max}}$. Since f is concave and $P_{\max}$ is a global maximum, f's value on the segment is monotonic, meaning we can efficiently find $\tilde{Q}$ to arbitrary precision using binary search. The resulting safe zone is not optimal, but in practice we find it performs well.

4.2 Additive Slack and Drift

Consider the following example with 2 nodes: $P_0^1 = [1/2, 1/2]$, $P_0^2 = [1, 0]$, and $P_0 = (P_0^1 + P_0^2)/2 = [3/4, 1/4]$. Node 1 has maximum possible entropy $f(P_0^1) = \ln 2$ while node 2 has minimal entropy $f(P_0^2) = 0$. Any non-trivial safe zones we compute from $f(P_0) \approxeq 0.562$ would be immediately violated. In other words, if P_0^j is too far from P_0, synchronization may not resolve local violations. Moreover, it

[5] For convex f, the first constraint becomes $f(P) - T_L \le 0$, which is still convex.

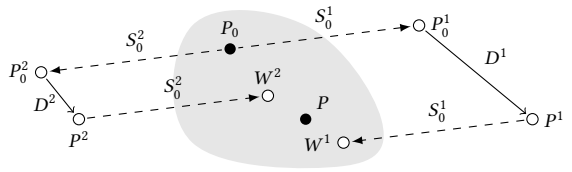


Figure 2: Global, local, drift, and slack vectors with two nodes. The local vectors are outside the safe zone (shaded gray), while W^1, W^2 and $P = \frac{W^1+W^2}{2}$ are inside the safe zone.

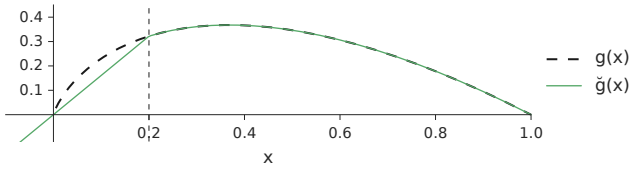


Figure 3: $g(x) = -x \ln x$ and its concave "extension" to the negative domain, $\check{g}(x)$. x_0 is set to 0.2 for illustration.

becomes more likely that $P \in C$ while $P^j \notin C$. We use *drift*, a form of *additive slack*, to address these problems.

Additive Slack. Recall that the global vector is the weighted mean of the local vectors, $P = \sum_j \alpha^j P^j$, that $\alpha_0^j = \alpha^j$, and that the safe zone C is convex. Let S_0^j be node j's *slack* vector, allocated by the coordinator during synchronization such that $\sum_{j=1}^m \alpha^j S_0^j = 0_k$, where 0_k is the zero vector of length k. Each node now checks if $W^j = P^j - S_0^j \in C$ rather than $P^j \in C$. Since $\sum_j \alpha^j W^j = \sum_j \alpha^j P^j - 0_k = P$, and from convexity of the safe zones, it follows that if $W^j \in C_U$ and $W^j \in C_L$ for all nodes, then $T_L \leq f(P) \leq T_U$. We can avoid unnecessary synchronization by choosing S_0^j such that $f(W^j)$ is closer to $f(P)$ than to $f(P^j)$.

Drift. The *drift* vector $D^j = P^j - P_0^j$ is the change in the local vector of node j since the last sync. Changes in the data are expected to be gradual, so drifts are likely small, meaning $P_0 + D^j$ is close to P_0 and hopefully closer to P than to P^j. Since $P = \sum_j \alpha^j P^j = P_0 + \sum_j \alpha^j D^j$, nodes can check $P_0 + D^j \in C$ instead of $P^j \in C$, and retain the threshold monitoring guarantee. (In an equivalent formulation, C is shifted to center on 0_k and nodes check $D^j \in C$ directly.) Defining slack to be $S_0^j = P_0^j - P_0$ gives us exactly $W^j = P^j - S_0^j = P_0 + D^j$. Figure 2 illustrates drift slack with two nodes. We use drift slack for both the lower and upper threshold monitors.

Returning to the example, after sync we have $S_0^1 = [-1/4, 1/4]$ and $S_0^2 = [1/4, -1/4]$, thus $P_0^1 - S_0^1 = P_0^2 - S_0^2 = P_0 \in C$.

4.3 Extending the Domain of $\mathrm{H_{MLE}}$

One complication unique to entropy is that elements of P are assumed to be probabilities: $p_i \geq 0$ and $\sum_{i=1}^k p_i = 1$. Entropy $f(P) = -\sum_{i=1}^k p_i \ln p_i$ is undefined if any $p_i < 0$. Subtracting the slack vectors from the probabilities may cause them to fall outside f's domain, making it impossible to compute $f(W^j)$ when checking whether $W^j \in C_L$. One simple solution is for nodes to declare a safe zone violation and force a synchronization if any element of W^j is above 1 or below 0, but this may result in many spurious violations since P^j changes while S_0^j was fixed during the last sync.

We address this in two ways. First, we allocate slack vectors S_0^j such that their elements sum to zero so $\sum_{i=1}^k W_i^j = 1$. For drift slack, this happens naturally. Second, we "extend" the domain of f by defining another function $\check{f}$ such that: (a) $\check{f}$ is defined everywhere (its domain is $\mathbb{R}^k$); (b) $\forall P \in \mathbb{R}_+^k : \check{f}(P) \leq f(P)$; and (c) $\check{f}$ is concave.

Unfortunately, it is impossible to extend f's domain to the negative range and maintain concavity: $\lim_{x \to 0^+} g'(x) = \infty$, where $g(x) = -x \ln x$. Instead, we define $\check{f}(P) = \sum_{i=1}^k \check{g}(p_i)$, where:

$$\check{g}(x) = \begin{cases} -x \ln x & \text{if } x \geq x_0 \\ -x \ln x_0 & \text{if } x < x_0 \end{cases}$$

and $x_0 \in (0, 1]$ is a suitable small constant. Note that $\check{f}$ meets the above criteria since $\check{g}$ is defined everywhere, $\check{g}(x) \leq g(x)$ for all $x > 0$, and $\check{g}(x)$ is concave[6].

Figure 3 shows $g(x)$ and $\check{g}(x)$. There is a tradeoff in setting x_0: if x_0 is too high, then $\check{f}$ is too far below f meaning that $f(W^j) < T_L$ more often. Yet setting x_0 too low makes $\ln x_0$ large, so negative elements of W^j could overwhelm the positive ones. In practice, our experience shows that negative elements in W^j are rare and tend to be much smaller than the positive ones, so it is safe to set x_0 to a very small number. We propose the heuristic $x_0 \approxeq \frac{1}{100k}$.

Having defined concave $\check{f}$, we can monitor the lower bound using geometric monitoring: the new convex lower safe zone is $\check{C}_L = \{W^j : \check{f}(W^j) \geq T_L\}$. If $\check{f}(W^j) \geq T_L$ for all nodes, then $f(P) \geq \check{f}(P) \geq T_L$. Each node simply checks whether $W^j \in \check{C}_L$.

4.4 Entropy Sketch

What happens when k, the number of bins or event types, is too large? For example, for network monitoring we estimate the entropy of IP addresses [7], where $k = 2^{32}$ (for IPv4), or even $k = 2^{64}$ (origin-destination pairs [50]). It is infeasible to have so many counters in memory or send messages with local vectors of such size. In such cases we use sketching to estimate entropy.

An entropy *sketch* $\hat{H}$ is a data structure Y of size s and corresponding functions $h(Y)$ such that $s \ll k$ and $\mathrm{H_{MLE}} \approxeq h(Y)$ with probabilistic error bounds that usually depend on the relative size of s and k. Incoming and outgoing observations in the stream update Y instead of P. Sketching is also effective in reducing bandwidth, since $s \ll k$, and is often proposed as a distributed entropy approximation [27, 50].

We describe a CIDER variant that allows high k and bandwidth reduction by adopting Clifford and Cosma's near-optimal entropy sketch [13]. This sketch is surprisingly simple: Y is a linear projection of P, and $h(Y)$ is a concave function. Denote by B the vector of global counts $[b_1, b_2, \ldots, b_k]$. Then $Y = \frac{\mathbf{R} \cdot B}{n} = \mathbf{R} \cdot P$ where $\mathbf{R}$ is a $s \times k$ random projection matrix with i.i.d elements drawn from $F(x; 1, -1, \pi/2, 0)$ and $h(Y) = -\ln\left(\frac{1}{s}\sum_{i=1}^s \exp(y_i)\right)$. Since the sketch is linear, the global sketch vector $Y = \mathbf{R}P$ is the weighted

[6] It is continuous and $\forall a, b \in \mathbb{R}, \alpha \in [0, 1] : \check{g}(\alpha a + (1-\alpha)b) \geq \alpha \check{g}(a) + (1-\alpha)\check{g}(b)$.

mean of the local sketch vectors $Y^j = \mathbf{R}P^j$:

$$Y = \mathbf{R}P = \mathbf{R}\sum_{j=1}^{m}\alpha^j P^j = \sum_{j=1}^{m}\alpha^j \mathbf{R}P^j = \sum_{j=1}^{m}\alpha^j Y^j \quad .$$

Y^j and Y completely replace P^j and P in the algorithm: nodes only maintain and communicate sketches of size s, and need not maintain k explicit counters in memory. We monitor $\hat{H}$ in the same way that we monitor $\mathrm{H_{MLE}}$: we generate a fixed $\mathbf{R}$ (using the procedure in [13, Table 1]), apply the sketch to obtain local vectors Y^j ([13, Table 2] offers an equivalent incremental formulation of the sketch), and use local safe zones for the lower and upper bound to guarantee the approximation. Since $h(Y)$ is a function of the weighted mean of local vectors, and since it is concave[7], we can directly apply Lemma 4.1 to derive local safe zones. Since Y^j are not probabilities, we need not constrain Q's elements to sum to 1. Moreover, h is defined everywhere so we do not use the domain extension trick from Section 4.3. The resulting approximation is probabilistic rather than deterministic, since we are estimating $\hat{H}$ and not $\mathrm{H_{MLE}}$ (see [13] for details).

4.5 Dynamically-sized Sliding Windows

So far we have assumed fixed window size: n^j is constant, and $n^j = n_0^j$. Yet in many entropy monitoring settings (e.g., network montitoring [22]) observations can arrive to different nodes at different times and rates, so that the same node can have (sometimes vastly!) different numbers of observations in the window. Though geometric monitoring has been previously used to build communication-efficient approximations [20, 21, 35], previous work has always assumed fixed-size windows with identical sizes $n^j = \frac{n}{m}$, and that events arrive to all streams at a constant, identical rate. To support true time-based sliding windows, we must extend geometric monitoring to support variable-sized windows using a new slack re-weighting scheme. This new scheme is applicable to most geometric monitoring applications.

We first observe that the original convexity argument holds even if nodes do not actually know the values of α^j. Since n^j, the current number of observations in node j's sliding window, changes dynamically, the global n is unknown to any node, even the coordinator. Nevertheless, by definition, α^j are still positive and sum to 1, so $P = \sum_{j=1}^{m}\alpha_j P^j$. As before $\forall j : P^j \in C \implies P \in C$. This applies to both entropy and entropy sketch.

Slack Re-weighting. One problem with variable-size windows is that previous additive slack schemes, including drift, no longer work correctly, because they assume that the number of observations at each node is fixed. Since we now allow n^j to change, the current weights α_j do not match the original slack weights α_0^j.

Denote by $\square_0$ the value of quantity $\square$ during last sync and recall that $\alpha_0^j = \frac{n_0^j}{n_0}$, $\alpha^j = \frac{n^j}{n}$, $P = \sum_j \alpha^j P^j$, and $P_0 = \sum_j \alpha_0^j P_0^j$. For drift slack, $S_0^j = P_0^j - P_0$, and from Section 4.2 we know that $\sum_j \alpha_0^j S_0^j = 0_m$. Yet applying the original additive slacks S_0^j to the current P^j will not preserve P, even though $\sum_j \alpha_0^j S_0^j = 0_m$, since in general $\alpha^j \neq \alpha_0^j$:

$$\sum_j \alpha^j \left(P^j - S^j\right) = P - \sum_j \alpha^j P_0^j + \sum_j \alpha^j P_0 \neq P \quad .$$

We address this problem using a novel slack re-weighting scheme: $S^j = \frac{n_0^j}{n^j} S_0^j$. Though the global n is unknown, nodes know the values of their local n^j and n_0^j so each can individually update its slack. With the new scheme:

$$\begin{aligned}\sum_j \alpha^j \left(P^j - S^j\right) &= \sum_j \left(\alpha^j P^j - \alpha^j \frac{n_0^j}{n^j} S_0^j\right) = P - \sum \frac{n^j}{n}\frac{n_0^j}{n^j} S_0^j \\ &= P - \frac{n_0}{n}\sum \alpha_0^j S_0^j = P - \frac{n_0}{n} 0_m = P \quad .\end{aligned}$$

It follows that for a convex safe zone C, if for all nodes $W^j = P^j - S^j \in C$, then $P \in C$. This slack re-weighting applies to any valid additive slack, as long as $\sum_j \alpha_0^j S_0^j = 0_m$. Moreover, it does not depend on the function being monitored and can be applied to other geometric monitoring algorithms.

Algorithm 3: Lazy synchronization with slack re-weighting.

Input: $\mathcal{S}$ the set of violating nodes, $\tilde{m}$ max nodes to lazy sync

while $|\mathcal{S}| < \tilde{m}$ **do**
  $n^V, n_0^V \leftarrow \sum_{j\in\mathcal{S}} n^j \quad , \sum_{j\in\mathcal{S}} n_0^j$
  $P^V, S^V \leftarrow \sum_{j\in\mathcal{S}} \frac{n^j}{n_V} P^j \ , \ \sum_{j\in\mathcal{S}} \frac{n_0^j}{n^j} S_0^j$
  if $P^V - S^V \in C_L$ ***and*** $P^V - S^V \in C_U$ **then break**
  pick node $j \notin \mathcal{S}$ to poll with balance strategy, add j to $\mathcal{S}$
if $|\mathcal{S}| = \tilde{m}$ **then** switch to eager synchronization (Alg. 2)
forall $j \in \mathcal{S}$ **do**
  send update to j: $n_0^j, S_0^j \leftarrow \frac{n_0^V}{n^V} n_0^j, \frac{n^j n^V}{n_0^j n_0^V}\left(P^j - \left(P^V - S^V\right)\right)$

4.6 Violation Resolution

When a node reports safe zone violation, the coordinator must resolve it. The simplest policy is *full* or *eager* synchronization [23]: poll all nodes for their local vectors and recompute P_0, approximation $h_0 = f(P_0)$, thresholds $T_L = LB(h_0)$, $T_U = UB(h_0)$, safe zones C_L, C_U, local counts n_0^j, and slack vectors S_0^j; updated values are sent to nodes and monitoring resumes. Eager sync is costly, however, and can be wasteful since many local safe zone violations occur when the current approximation is still within bounds [20].

An alternative policy is *lazy synchronization* [23]: poll the other nodes one at a time, and add them to a balancing set $\mathcal{S}$ until either the weighted mean of their local vectors $P^V = \left(1/\sum_{j\in\mathcal{S}} n^j\right)\sum_{j\in\mathcal{S}} n^j P^j$ is inside the safe zones, or an upper limit $\tilde{m} < m$ on the size of $\mathcal{S}$ has been reached. If the set of nodes with violations can indeed be balanced, the coordinator updates S_0^j and n_0^j for the nodes in $\mathcal{S}$ (making sure the weighted sum is unchanged), all without contacting or sending updates to nodes not in $\mathcal{S}$. If $|\mathcal{S}|$ is too large, we assume that P itself is outside the safe zone and trigger an eager sync. Lazy sync with dynamically-sized sliding windows requires careful re-weighting of slack; Alg. 3 contains the full details.

Given the set of violating nodes, which node should the coordinator poll next? In *random* balancing, the coordinator simply chooses

[7] The function $\ln\left(\sum_i \exp y_i\right)$ is known as Log-Sum-Exp and is convex [10].

a random node. Previous work has proposed the *least-used* (LU) strategy: choose the non-violating node that been used least often in balancing [20] or the one that has been polled least recently [23].

We propose a new balancing strategy we call *opposite slack* ("oslack"): the coordinator polls the non-violating node j whose slack S_0^j has the opposite direction to the weighted mean of local slacks of the nodes combined so far (S^V in Alg. 3). The coordinator knows the values of S_0^j for all nodes, so no extra communication is incurred. The intuition behind oslack is that since the current safe zone is "centered" around P_0, we want to poll a node j such that $P^j - P_0$ balances $P^V - P_0$. If V "pulls" P in one direction, we want to find a node that "pulls" it back into the safe zone. P^j is unknown for unpolled nodes, however. The idea behind oslack is that slack S_0^j can serve as as a proxy for $P^j - P_0$. Drift slack is $P_0^j - P_0$, and if local vectors have not shifted by much, S_0^j is close to $P^j - P_0$. Figure 2 illustrates this intuition: though $P^1 \neq P_0^1$ and $P^2 \neq P_0^2$, the slacks S_0^1 and S_0^2 still give a rough idea of the directions $P^1 - P_0$ and $P^2 - P_0$.

4.7 Approximating Rényi and Tsallis Entropies

We briefly discuss approximating two parametric generalizations of entropy: the Tsallis entropy

$$\mathrm{H}_\beta = \frac{1}{1-\beta}\left(\sum_{i=1}^{k} \Pr[X_i]^\beta - 1\right) \quad ,$$

and the Rényi entropy

$$\mathrm{H}_\alpha = \frac{1}{1-\alpha} \ln\left(\sum_{i=1}^{k} \Pr[X_i]^\alpha\right) \quad .$$

The Tsallis entropy is convex for $\beta < 0$ and concave for $\beta > 0$ [28], and can be computed from the weighted mean of local probabilities P^j. Therefore, we can apply Lemma 4.1 to derive safe zones and monitor it the same way we do for Shannon entropy and the sketch. H_β is defined everywhere, so we can safely use slack.

The Rényi entropy is concave for $\alpha \in (0, 1]$ (so Lemma 4.1 applies directly) and quasiconcave for all $\alpha \geq 0$ [28]. Quasiconcave functions have convex upper level sets, so the lower bound constraint $\{P | \mathrm{H}_\alpha(P) \geq T_L\}$ is convex, and the upper constraint $\{P | \mathrm{H}_\alpha(P) \leq T_H\}$ is "reverse convex". Therefore we can use the same half-plane trick from Lemma 4.1 to derive optimal safe zones. The optimization problem of finding the closest point Q now has a quasiconvex constraint. Though quasiconvex optimization problems have been extensively studied, we consider this beyond the scope of this work.

5 EMPIRICAL EVALUATION

We evaluate CIDER's performance on several public datasets representing real-world application scenarios. We simulate running the nodes and coordinator using the recorded timestamps in the dataset, keep track of the estimated and true entropy, and count any messages sent. Unless otherwise noted, we use absolute error bounds $LB(h_0) = h_0 - \epsilon$ and $UB(h_0) = h_0 + \epsilon$ where ϵ is the desired accuracy.

Our baseline is the *centralized* estimator where each node sends new observations to the coordinator as they arrive (unlike CIDER, it does not provide the global estimate to the nodes themselves).

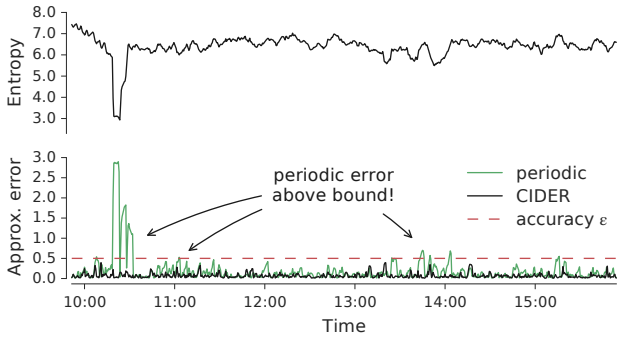


Figure 4: Entropy of source IP addresses in the CTU1 dataset, with 10 nodes and a five minute sliding window (top) and CIDER approximation error (black, bottom) with accuracy $\epsilon = 0.5$ (dashed horizontal line). CIDER maintains the desired accuracy with 1.2% communication. An equivalent periodic estimator (green) is unable to maintain this accuracy.

Our main performance metric is *communication fraction*, defined as the ratio of messages sent by CIDER to those sent by the centralized algorithm. For both algorithms, we exclude messages sent in the first window.

5.1 Network Traffic Monitoring

Traffic entropy (e.g., of IP addresses, ports, or origin-destination flows) is a commonly-suggested feature for network health monitoring [3, 7, 22, 40, 50]. Estimating traffic entropy is a thorny problem, however, due to the volume and rate of incoming network packets, distributed input sources (often routers), and the need for timely detection. Existing systems use a combination of periodic reporting (increasing latency), sketching (increasing error), and sub-sampling (increasing both error and risk of missed events). CIDER offers an alternative: low communication, deterministic bounds, and near-realtime detection of changes in entropy.

We evaluate CIDER's performance on the CTU-13 collection [22], a set of publicly available annotated traffic captures. We use the bidirectional NetFlow files from the **CTU1**, **CTU4**, **CTU9** and **CTU10** datasets, each containing 1.1M–2.8M NetFlows recorded over 4–6 hours. NetFlows are assigned to simulated "routers" (nodes) by uniformly dividing the 3rd octet of the destination IP to m bins, and we approximate the entropy of source IP address ($k = 2^{32}$) with sketch of size $s = 100$. Unless otherwise noted, we simulate $m = 10$ nodes with five minute sliding windows, use lazy synchronization with up to $\tilde{m} = 3$ nodes and oslack balancing, and require accuracy $\epsilon = 0.5$. Section 5.4 uses **CTU2** to select $\tilde{m}$ and balance strategy.

Figure 4 shows an example of such a simulation on CTU1 (length 6 hours and 15 minutes). CIDER maintains accurate approximation using only 1.2% communication of the centralized algorithm – equivalent to nodes sending updates every 4 minutes and 30 seconds. Such a periodic estimator would be unable to maintain the desired accuracy $\epsilon = 0.5$, however: its estimation error reaches almost 3.0. This translates to a four minute delay in detecting any changes, and short entropy spikes can be missed entirely. CIDER's

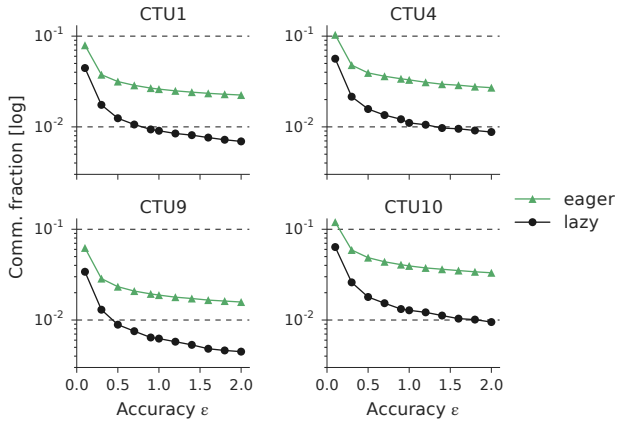


Figure 5: Communication for CIDER at different approximation accuracies when monitoring source IP entropy in the CTU datasets.

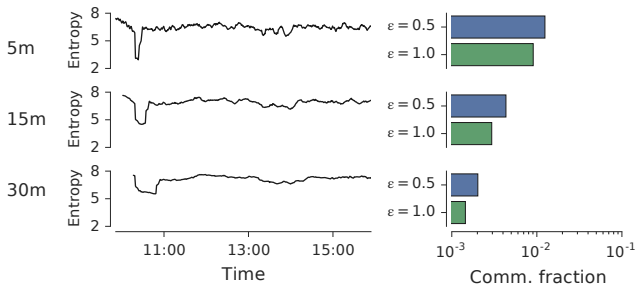


Figure 6: Larger sliding windows smooth out variations in probabilities and therefore entropy (left), resulting in lower communication when monitoring the CTU1 dataset (right).

approximation error, on the other hand, is always within bounds and guarantees no detection latency or missed spikes.

Accuracy ϵ. Figure 5 shows CIDER's communication with different approximation accuracies on the CTU1, CTU4, CTU9, and CTU10 datasets. Lazy synchronization with oslack is considerably more communication-efficient than eager synchronization, with up to an order of magnitude difference. In general, communication is reduced by roughly two orders of magnitude, even with a fairly strict $\epsilon = 0.5$. Even with highest accuracy $\epsilon = 0.1$, 0.5% of the total entropy range $0–\ln 2^{32}$, CIDER only sends 10% messages as the centralized algorithm. On the other hand, with a relaxed $\epsilon = 2.0$, enough to catch the large entropy spikes in CTU1 (Figure 4), CIDER yields 0.7% communication.

Window Size. CIDER's performance does not inherently depend on the window size, and instead reflects the behavior of local and global probabilities. Figure 6 illustrates this on the CTU1 dataset. Increasing window size means more observations, which in turn means that probabilities change gradually rather than abruptly. As entropy spikes become smaller and even disappear, CIDER adapts and requires less communication to maintain the same accuracy.

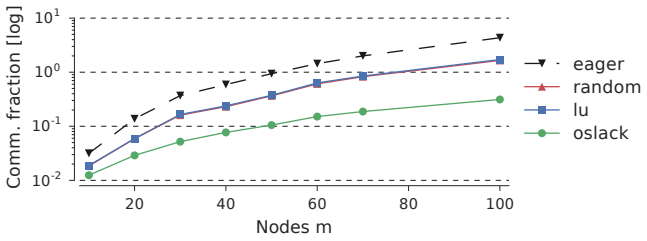


Figure 7: Communication vs. number of simulated nodes m in the CTU1 dataset. Communication with eager synchronization quickly grows above that of centralized. Lazy sync scales better, with oslack balancing yielding a considerable improvement over LU.

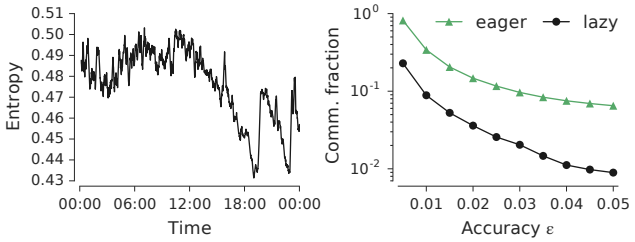


Figure 8: Left: entropy of the WC98 dataset ($m = 27$, $k = 13$) with 10 minute sliding window. Right: CIDER's communication at different approximation accuracies, reaching 0.8% with $\epsilon = 0.05$, enough to detect larger entropy fluctuations.

Number of Nodes m. Figure 7 explores how different violation resolution strategies scale as we increase the number of simulated nodes m in CTU1. Eager synchronization scales poorly, and with 50 nodes it performs no better than the centralized algorithm. Lazy sync with LU or random balancing can scale up to 70 nodes while still reducing communication. The oslack policy, however, scales much better and can monitor even 100 nodes with only 30% communication. Section 5.4 describes how we set lazy sync parameters.

5.2 Load Balancing

In the classic load balancing scenario, work items from an input stream must be assigned to one of k workers (e.g., based on a hash function). Entropy of worker load distribution can be used to identify load imbalance, and react accordingly (add workers, change work distribution, etc.). Consider the case where there are m distributed streams. Though each node ("balancer") has a local count of worker assignments, it cannot know whether global workload is balanced. Similar situations arise in key-value stores, distributed stream processing systems [38], and more. We observe that in this settings sketching approaches cannot help, since k is typically low. CIDER provides a view on global entropy to every node, allowing them to adjust work allocation as needed.

The WorldCup'98 dataset [5] contains access logs for the 1998 World Cup web site for a 3 months period. We define the **WC98** dataset as the access logs from June 10, 1998 (day 46, with 50M requests), where server ID for each request is used as the node

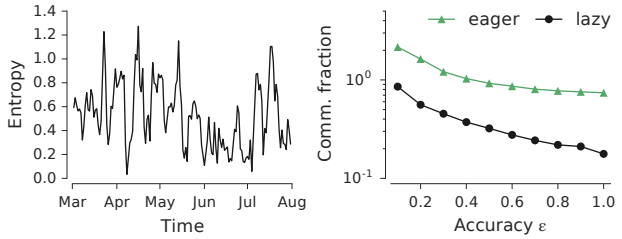


Figure 9: Left: entropy of the TAQ dataset ($m = 23$, $k = 20$) with 24 hour sliding window. Right: CIDER's communication at different approximation accuracies.

($m = 27$), and the request type (image, html, etc., $k = 13$) serves as the assigned worker (the observation). For this dataset we use $\epsilon = 0.05$, 10 minute sliding windows, and max $\tilde{m} = 5$ in lazy sync. We used data from day 44 to tune these parameters (Section 5.4).

Figure 8 shows CIDER's performance on the WC98 dataset for accuracies in the very tight range 0.005–0.05 (which is 0.2%–2% of maximum entropy, $\ln 13 =\sim 2.565$). In this range, CIDER reduces communication by between one and two orders of magnitude. Lazy sync is even more critical here, with an almost 10 times communication reduction compared to eager synchronization. In practice we might only care about large changes in entropy. With $\epsilon = 0.25$ (10% of entropy range), for example, CIDER almost never syncs, requiring only 0.25% communication – a three orders of magnitude communication reduction. This illustrates one advantage of CIDER over existing approaches: when entropy changes slowly, practically no communication is necessary; if entropy starts to change rapidly, CIDER will ramp up communications to adapt.

5.3 Air Quality Monitoring

The Revised Air Quality Index [12] (RAQI) combines metrics from five air pollutants[8] into a single number, and incorporates Shannon entropy as one of its components to represent variations within the measurement period. This component is the entropy over a window of 24 hours of $\max\{A_1 \ldots A_5\}$, where A_i are individual pollutant measurements converted to AQI scores using the piecewise linear transformation described in [1].

We used pollutant data from North Taiwan available from the Taiwan Air Quality Monitoring Protection Agency[9] to track this entropy component. The **TAQ** dataset is defined as the hourly data from January 1 through July 31 2015 for the $m = 23$ monitoring stations that report the five pollutants. The first two months are used to tune $\tilde{m}$ for lazy synchronization (Section 5.4), and the rest for evaluation. We divided the AQI range of 1–500 to $k = 20$ uniform bins. We use lazy sync with oslack balancing, with $\tilde{m} = 11$ max nodes, and the standard RAQI sliding window of 24 hours.

Performance of CIDER on the TAQ dataset is shown in Figure 9. Entropy fluctuates widely, both due to large hourly differences, but also because local windows only have $n^j = 24$ observations in the window so the MLE estimator is at the lower range of accuracy (where $n^j \sim k$ [39, 41]). CIDER must sync often (as would any other algorithm), but is still able to reduce communication to 17%–85%. For example, in remote, battery-operated wireless monitoring stations, this would translate to substantial increases in battery life.

[8] PM_{10}, O_3, SO_2, CO, and NO_2
[9] http://taqm.epa.gov.tw/taqm/en/YearlyDataDownload.aspx

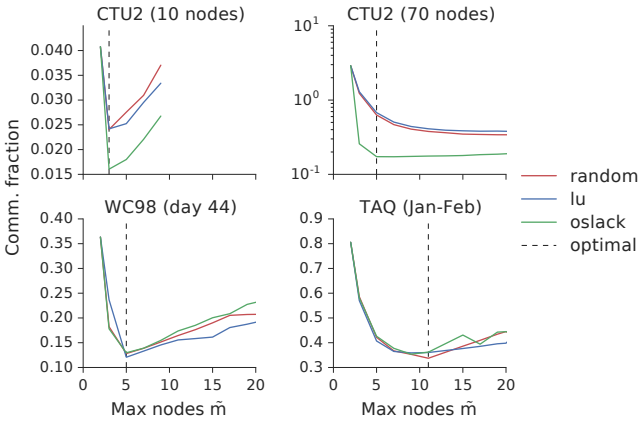


Figure 10: Communication vs. max number of balancing nodes on different datasets. On CTU2, random balancing works as well as LU, while oslack performs considerably better than both. On the other datasets all three perform similarly at the optimal $\tilde{m}$.

5.4 Lazy Synchronization

Lazy synchronization is superior to eager synchronization, but we must still set $\tilde{m}$, the maximum size of the balancing set, and select a balancing strategy (random, LU, or oslack).

Figure 10 shows CIDER's performance with different values of $\tilde{m}$, for each of the three datasets. To avoid overfitting, we used different data to tune $\tilde{m}$. LU and random achieve similar communication on all datasets, and are essentially equivalent. On the WC98 and TAQ datasets, oslack performance is equivalent to the other two, especially at the optimal value of $\tilde{m}$. On the CTU2 dataset, however, oslack is substantially better, especially with 70 simulated nodes. We conclude that oslack performs as well or better than other balancing strategies, and use it throughout. This is further confirmed by Figure 7 where LU performance is identical to that of random, while oslack balancing is essential to good scaling.

6 CONCLUSIONS

We present CIDER, a practical, communication-efficient entropy estimator for distributed streams. Unlike previous work, CIDER provides deterministic approximation bounds, and supports both insertions and deletions, allowing for variable-sized or time-based windows. Like previous work, it allows the use of entropy sketches to both reduce message size and to handle very high cardinality (k in the billions). We also extend geometric monitoring with variable-sized windows using a novel slack (drift) re-weighting scheme, which can be used in other geometric monitoring algorithms.

CIDER adapts to the underlying data and the user-specified bounds, and provides no guarantees for reduced communication, other than the trivial. This is consistent with Arackaparambil et al. [4] conclusion that nontrivial savings cannot be guaranteed

when monitoring non-monotone functions with deletions. Nevertheless, in practice we see that CIDER achieves considerable communication savings on real datasets: up to three orders of magnitude with relaxed accuracy, and often two orders of magnitude with strict approximation bounds.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Horizon 2020 Research And Innovation Programme under grant agreements no. 688380 and no. 727277-2. We thank the Max & Rachel Javit fund in the Technion Autonomous Systems Program (TASP) for their generous support.

REFERENCES

[1] 2016. Technical Assistance Document for the Reporting of Daily Air Quality – the Air Quality Index (AQI). (May 2016). https://www.airnow.gov/index.cfm?action=pubs.index

[2] R. E. Anderson. 2004. Entropy of EEG during anaesthetic induction: a comparative study with propofol or nitrous oxide as sole agent. *British Journal of Anaesthesia* 92, 2 (2004), 167–170.

[3] C. Arackaparambil, S. Bratus, J. Brody, and A. Shubina. 2010. Distributed Monitoring of Conditional Entropy for Anomaly Detection in Streams. In *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW)*.

[4] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. 2009. Functional Monitoring Without Monotonicity. In *Proc. ICALP '09*.

[5] Martin Arlitt and Tai Jin. 1998. 1998 World Cup Web Site Access Logs. (1998). http://ita.ee.lbl.gov/html/contrib/WorldCup.html

[6] Christoph Bandt and Bernd Pompe. 2002. Permutation Entropy: A Natural Complexity Measure for Time Series. *Phys. Rev. Lett.* 88, 17 (2002), 174102.

[7] Przemyslaw Berezinski, Bartosz Jasiul, and Marcin Szpyrka. 2015. An Entropy-Based Network Anomaly Detection Method. *Entropy* 17, 4 (2015), 2367–2408.

[8] Lakshminath Bhuvanagiri and Sumit Ganguly. 2006. Estimating Entropy over Data Streams. In *Proceedings of the 14th Conference on Annual European Symposium - Volume 14 (ESA'06)*.

[9] J. Bruhn. 2006. Depth of anaesthesia monitoring: what's available, what's validated and what's next? *British Journal of Anaesthesia* 97, 1 (2006), 85–94.

[10] G.C. Calafiore and L. El Ghaoui. 2014. *Optimization Models*. Cambridge University Press.

[11] Amit Chakrabarti, Khanh Do Ba, and S. Muthukrishnan. 2006. Estimating Entropy and Entropy Norm on Data Streams. In *Proc. STACS '06*.

[12] Wan-Li Cheng, Yu-Chih Kuo, Pay-Liam Lin, Ken-Hui Chang, Yu-Song Chen, Tso-Mei Lin, and Ruth Huang. 2004. Revised air quality index derived from an entropy function. *Atmospheric Environment* 38, 3 (2004), 383–391.

[13] Peter Clifford and Ioana Cosma. 2013. A simple sketching algorithm for entropy estimation over streaming data. In *Proc. AISTATS '13*, Vol. 31.

[14] Graham Cormode. 2013. The Continuous Distributed Monitoring Model. *SIGMOD Rec.* 42, 1 (2013), 5–14.

[15] Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. 2012. Continuous Sampling from Distributed Streams. *J. ACM* 59, 2, Article 10 (2012), 10:1–10:25 pages.

[16] Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *JMLR* 17, 83 (2016), 1–5.

[17] Richard Klaus Ellerkmann, Vidal-Markus Liermann, Thorsten Michael Alves, Ingobert Wenningmann, Sascha Kreuer, Wolfram Wilhelm, Heiko Roepcke, Andreas Hoeft, and Jörgen Bruhn. 2004. Spectral Entropy and Bispectral Index as Measures of the Electroencephalographic Effects of Sevoflurane. *Anesthesiology* 101, 6 (2004), 1275–1282.

[18] M. Ermes, J. Parkka, J. Mantyjarvi, and I. Korhonen. 2008. Detection of Daily Activities and Sports With Wearable Sensors in Controlled and Uncontrolled Conditions. *IEEE Transactions on Information Technology in Biomedicine* 12, 1 (2008), 20–26.

[19] Arik Friedman, Izchak Sharfman, Daniel Keren, and Assaf Schuster. 2014. Privacy-Preserving Distributed Stream Monitoring. In *21st Annual Network and Distributed System Security Symposium (NDSS '14)*.

[20] Moshe Gabel, Daniel Keren, and Assaf Schuster. 2014. Communication-efficient Distributed Variance Monitoring and Outlier Detection for Multivariate Time Series. In *Proc. IPDPS '14*.

[21] Moshe Gabel, Daniel Keren, and Assaf Schuster. 2015. Monitoring Least Squares Models of Distributed Streams. In *Proc. KDD '15*.

[22] S. García, M. Grill, J. Stiborek, and A. Zunino. 2014. An Empirical Comparison of Botnet Detection Methods. *Computers & Security* 45 (2014), 100–123.

[23] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. 2013. Sketch-based Geometric Monitoring of Distributed Stream Queries. *Proc. VLDB Endow.* 6, 10 (2013), 937–948.

[24] Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Izchak Sharfman, and Assaf Schuster. 2012. Prediction-based Geometric Monitoring over Distributed Data Streams. In *Proc. SIGMOD '12*.

[25] Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Izchak Sharfman, and Assaf Schuster. 2014. Distributed Geometric Query Monitoring Using Prediction Models. *TODS* 39, 2 (2014), 16:1–16:42.

[26] Nikos Giatrakos, Yannis Kotidis, Antonios Deligiannakis, Vasilis Vassalos, and Yannis Theodoridis. 2013. In-network Approximate Computation of Outliers with Quality Guarantees. *Inf. Syst.* 38, 8 (2013), 1285–1308.

[27] Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. 2008. Sketching and Streaming Entropy via Approximation Theory. In *Proc. FOCS '08*.

[28] Siu-Wai Ho and Sergio Verdú. 2015. Convexity/Concavity of Renyi Entropy and α-mutual Information. In *2015 IEEE International Symposium on Information Theory (ISIT)*.

[29] Piotr Indyk. 2006. Stable Distributions, Pseudorandom Generators, Embeddings, and Data Stream Computation. *J. ACM* 53, 3 (2006), 307–323.

[30] Michael Kamp, Mario Boley, Daniel Keren, Assaf Schuster, and Izchak Sharfman. 2014. Communication-Efficient Distributed Online Prediction by Dynamic Model Synchronization. In *Proc. ECML PKDD '14*.

[31] N. Kannathal, Min Lim Choo, U. Rajendra Acharya, and P.K. Sadasivan. 2005. Entropies for detection of epilepsy in EEG. *Computer Methods and Programs in Biomedicine* 80, 3 (2005), 187–194.

[32] Daniel Keren, Guy Sagy, Amir Abboud, David Ben-David, Assaf Schuster, Izchak Sharfman, and Antonios Deligiannakis. 2014. Geometric Monitoring of Heterogeneous Streams. *IEEE TKDE* 26, 8 (2014), 1890–1903.

[33] Daniel Keren, Izchak Sharfman, Assaf Schuster, and Avishay Livne. 2012. Shape Sensitive Geometric Monitoring. *IEEE TKDE* 24, 8 (2012), 1520–1535.

[34] Arnon Lazerson, Moshe Gabel, Daniel Keren, and Assaf Schuster. 2017. One for All and All for One: Simultaneous Approximation of Multiple Functions over Distributed Streams. In *Proc. DEBS '17*.

[35] Arnon Lazerson, Daniel Keren, and Assaf Schuster. 2016. Lightweight Monitoring of Distributed Streams. In *Proc. KDD '16*.

[36] Arnon Lazerson, Izchak Sharfman, Daniel Keren, Assaf Schuster, Minos Garofalakis, and Vasilis Samoladas. 2015. Monitoring Distributed Streams Using Convex Decompositions. *Proc. VLDB Endow.* 8, 5 (2015), 545–556.

[37] S. Muthukrishnan. 2003. Data Streams: Algorithms and Applications. In *Proc. SODA '03*.

[38] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, David Garcia-Soriano, Nicolas Kourtellis, and Marco Serafini. 2015. The Power of Both Choices: Practical Load Balancing for Distributed Stream Processing Engines. In *Proc. ICDE '15*.

[39] Ilya Nemenman, William Bialek, and Rob de Ruyter van Steveninck. 2004. Entropy and information in neural spike trains: Progress on the sampling problem. *Physical Review E* 69, 5 (2004).

[40] George Nychis, Vyas Sekar, David G. Andersen, Hyong Kim, and Hui Zhang. 2008. An Empirical Evaluation of Entropy-based Traffic Anomaly Detection. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC '08)*.

[41] Liam Paninski. 2003. Estimation of entropy and mutual information. *Neural computation* 15, 6 (2003), 1191–1253.

[42] Philippe Renevey and Andrzej Drygajlo. 2001. Entropy Based Voice Activity Detection in Very Noisy Conditions. In *EUROSPEECH 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*. 1887–1890.

[43] Guy Sagy, Daniel Keren, Izchak Sharfman, and Assaf Schuster. 2010. Distributed Threshold Querying of General Functions by a Difference of Monotonic Representation. *Proc. VLDB Endow.* 4, 2 (2010), 46–57.

[44] Nicola Scafetta and Paolo Grigolini. 2002. Scaling detection in time series: Diffusion entropy analysis. *Phys. Rev. E* 66, 3 (2002), 036130.

[45] Izchak Sharfman, Assaf Schuster, and Daniel Keren. 2007. A geometric approach to monitoring threshold functions over distributed data streams. *TODS* 32, 4 (2007), 23.

[46] Adwitiya Sinha and Daya Krishan Lobiyal. 2013. Performance evaluation of data aggregation for cluster-based wireless sensor network. *Human-centric Computing and Information Sciences* 3, 1 (2013).

[47] Ran Wolff. 2015. Distributed Convex Thresholding. In *Proc. PODC '15*.

[48] David P. Woodruff and Qin Zhang. 2012. Tight Bounds for Distributed Functional Monitoring. In *Proc. STOC '12*.

[49] Gal Yehuda, Daniel Keren, and Islam Akaria. 2017. Monitoring Properties of Large, Distributed, Dynamic Graphs. In *Proc. IPDPS '17*.

[50] Haiquan (Chuck) Zhao, Ashwin Lall, Mitsunori Ogihara, Oliver Spatscheck, Jia Wang, and Jun Xu. 2007. A Data Streaming Algorithm for Estimating Entropies of OD Flows. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07)*.