Monitoring Persistent Items in the Union of Distributed Streams

Sneha Aman Singh[a], Srikanta Tirthapura[a,*]

[a]*Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, 50010, USA.*

Abstract

A persistent item in a stream is one that occurs regularly in the stream without necessarily contributing significantly to the volume of the stream. Persistent items are often associated with anomalies in network streams, such as botnet traffic and click fraud. While it is important to track persistent items in an online manner, it is challenging to zero-in on such items in a massive distributed stream. We present the first communication-efficient distributed algorithms for tracking persistent items in a data stream whose elements are partitioned across many different sites. We consider both infinite window and sliding window settings, and present algorithms that can track persistent items approximately with a probabilistic guarantee on the approximation error. Our algorithms have a provably low communication cost, and a low rate of false positives and false negatives, with a high probability. We present detailed results from an experimental evaluation that show the communication cost is small, and that the false positive and false negative rates are typically much lower than theoretical guarantees.

Keywords: distributed streams, persistent items

1. Introduction

It is a significant problem today to find trends and anomalies from large data sets. This problem is especially challenging on streaming data that is continuously changing through a sequence of updates. Motivated by distributed network monitoring, we consider monitoring of data that is not only streaming, but also distributed across multiple sites, so that no single processor observes all the data. A single network monitor can observe a local stream of network traffic, but the target of monitoring is the logical stream that is formed through the union of all local streams. Monitoring each individual stream in isolation may not yield the desired insights, and the interaction between the different streams needs to be considered carefully. This setup is called the distributed streams model [17, 18, 11, 34, 26, 12, 36]. Web log analytics [14, 23, 32] is an example application that can be modeled as a distributed stream. In this system, there are multiple

*Corresponding author. Phone: +1 (515) 294 3546. Fax: +1 (515) 294 3637.
Email addresses: sneha@iastate.edu (Sneha Aman Singh), snt@iastate.edu (Srikanta Tirthapura)

web servers each of which have their own log of web accesses, which is a (large) locally observable stream, but patterns such as typical user behavior, and anomalies that could lead to malicious user behavior, have to be detected on the union of the distributed web logs.

We address the identification of a feature called a "persistent item" from a massive distributed stream. A persistent item is one that occurs regularly in the stream, but does not necessarily contribute significantly to the volume of the stream. Let n denote the total number of timeslots in a stream $\mathcal{S}$. The *persistence* of an item d, denoted $p(d)$, is defined as the number of distinct timeslots where d appeared in a stream. Clearly, $0 \leq p(d) \leq n$. Note that multiple occurrences of the same item in the same timeslot do not contribute repeatedly to the persistence of the item. For a parameter $0 \leq \alpha < 1$, an *α-persistent item* is defined as an item whose persistence is at least αn. The above metric was used in [19] in the context of botnet traffic detection.

A persistent item is different from a "frequent item" in the stream (often known as a "heavy-hitter"). A frequent item is one that appears with a high frequency in the stream, and hence contribute significantly to the volume of the stream. A persistent item need not be a frequent item. For instance, consider an item that occurs exactly once in every timeslot, so that its persistence is 100 percent. The frequency of this item is very low, so that it will not be considered a frequent item in the stream. Similarly, a frequent item may not be persistent either; consider for example an item that occurs in a bursty manner within a timeslot, but never reoccurs within other timeslots. While this item contributes significantly to the volume of the stream, its persistence is very low.

Persistence is typical of many stealthy types of traffic on the Internet. Identifying persistent items can help in identifying anomalous and potentially malicious behavior in a network. For instance, Giroire et al. [19] showed that tracking all persistent destinations arising in traffic from end hosts in a domain led one to identify botnet Command and Control (C&C) destinations. The C&C destinations take control over compromised end hosts to create a botnet and carry out malicious activities in the network. Giroire et. al. observed that the C&C centers had to be in regular contact with the compromised end hosts to carry out their activities, and hence the persistence of the C&C destinations were high in the streams emanating from the end hosts. However, in order to evade detection by traditional volume-based anomaly detectors, the C&C traffic was designed to be low-volume and hence the C&C centers did not show up as heavy-hitters within the streams. Another instance is in Pay-Per-Click Online Advertising [1], where identifying persistent items can be used to detect click fraud [39]. In this instance, rival companies generate false clicks on advertisements at regular but infrequent intervals. In order to evade detection by volume-based detectors, the volume of such false clicks is kept low, and hence these do not appear as heavy hitters in the click stream.

In general, persistence captures behavior when a set of entities (perhaps controlled by a malicious user) together have regular communication with a remote entity, but try to hide the communication by keeping its volume small and having it originate from different entities at different times. Such behaviors are not caught by tracking frequent items within a stream.

We consider the following distributed streaming model, which has also been adopted in prior work [17, 18, 11]. The distributed system has k sites, numbered from 1 to k; each site i receives a local stream $\mathcal{S}_i$. There is a special coordinator site that communicates with the individual sites and is required to perform all aggregation and mining tasks in the (logical) stream $\bigcup_{i=1}^{k} \mathcal{S}_i$ formed by the union of all streams. A persistent item is defined as follows. Suppose time is divided into "timeslots"[1] Each local site observes a stream of tuples (d, t), where d is an item identifier, and t the timeslot at which d appeared. Note that multiple occurrences of the same item in the same timeslot, whether at the same site or at different sites, do not contribute repeatedly to the persistence of the item.

An item can be highly persistent in the distributed stream without being persistent in any single local stream. Consider the following situation where a particular destination IP address was present in every timeslot from 1 till n, but kept moving from one local site another in different timeslots, to evade detection. The persistence of the item in any local stream $\mathcal{S}_i$ is only $1/k$, but its overall persistence in the distributed stream is 100%. Identifying persistent items in a distributed stream can help detect such coordinated and distributed malicious behavior.

The goal of this work is to devise an algorithm for identifying persistent items, which minimizes (1) the communication between the processors and (2) the memory footprint of the algorithm, both per node, and overall. While memory has always been a primary concern in data stream algorithm design in a centralized setting, in a distributed stream, the communication cost is even more important [17, 18, 11, 34, 30, 38], hence communication will be our primary metric.

1.1. Approximate Identification of Persistent Items

We first note that any algorithm for exactly identifying persistent items and none other than the persistent items must necessarily incur a large communication cost. In the worst case, this would need communication of the order of the total stream size. Hence, we consider approximate identification of persistent items, with a provable guarantee on the quality of approximation. Given persistence threshold $\alpha, 0 < \alpha \leq 1$, approximation parameter $\epsilon, 0 < \epsilon < \alpha$, error probability $\delta \in [0, 1]$, the task is to design a low communication cost and space

[1] We assume that time is loosely synchronized between the different sites, so that the different sites agree on which "timeslot" is currently in play. Since timeslots are typically of the order of minutes or more [19], the clocks only need to be synchronized to within a few seconds or more.

efficient algorithm that identifies α-persistent items from $\bigcup_{i=1}^{k} \mathcal{S}_i$, with the following properties:

- Low False Negative: If an item d has a persistence $p(d) \geq \alpha n$, then d is identified as α-persistent, with probability at least $(1 - \delta)$.
- Low False Positive: If an item d has a persistence $p(d) < (\alpha - \epsilon)n$, d is not reported as α-persistent, with probability at least $(1 - \delta)$.

We assume a synchronous communication model, where the system progresses in rounds. In each round, each site can observe one element (or none), send a message to the coordinator, and receive a response from the coordinator. The coordinator may receive up to k messages in a round, and respond to each of them in the same round. This model is essentially identical to the model assumed in previous work on distributed stream monitoring [11]. Our results do not change if the sites communicated at the end of each timeslot, rather than at the end of observing each element. The sizes of the different local streams at the sites, their order of arrival, and the interleaving of the arrivals at different sites, can all be arbitrary. The algorithm cannot make any assumption about these.

We consider different models for limiting the scope of the data on which aggregation is performed. We start with aggregation over the entire stream seen so far – referred to as the "infinite window" model, and we also consider the popular "sliding window" model [13, 18, 20, 6, 31, 5, 16], which restricts the scope to the most recently observed elements.

1.2. Contributions

We present the first communication-efficient algorithms for tracking persistent items over the union of multiple distributed streams, with approximation parameter ϵ and error probability δ.

Infinite Window Algorithm. We first present an algorithm for the setting where the data of interest is the union of all items over all the k streams observed so far. Let n denote the total number of timeslots so far. The expected space complexity over all sites is $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$ and the expected number of bytes transmitted across all sites is $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$ bytes, where $P = \sum_{d \in M} p(d)$, where M is the set of all distinct items observed in the stream, i.e. P is the sum of persistence values of all distinct items observed in the union of all streams.

Sliding Window Algorithm. Next we consider the setting where the data of interest is the union of data observed by all the k streams during the n most recent timeslots, and we present an algorithm for identifying persistent items within this data. The expected space complexity of our distributed algorithm

over all sites is $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$ and the expected number of communication bytes over all sites is $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$ bytes, where P is the sum of persistence values within the last n time slots, of all the distinct items seen in all the streams.

Simulations. We simulated our algorithm using real-world network trace data as well as synthetic data. These simulations show that our algorithm tracks α-persistent items with the observed guarantees, and that the communication and space overhead is much smaller than distributed implementation of existing algorithms.

1.3. Solution Overview

A straightforward approach to tracking persistent items is as follows. Every site i maintains a data structure S_i, which contains the set of all distinct timeslots that each item has appeared in stream $\mathcal{S}_i$. Note that this requires storing a set of up to n elements for each distinct item that has appeared in $\mathcal{S}_i$. Upon a query, each site i sends S_i to the coordinator, who can exactly compute the persistence of the item in the distributed stream, and return only those items that has persistence of at least αn. While this requires no coordination among the sites prior to the query, the total communication required at the time of the query is prohibitive, since it communicates up to $\Theta(n)$ bytes per item, per site, which can be very large when the number of distinct items is large. Clearly, this approach is expensive in terms of space required per site as well.

A centralized small-space streaming algorithm for persistent items such as the one in [24] can be used to track persistent items in each individual stream, but cannot be directly used in a distributed context. The reason is that the algorithm in [24] depends on using a simple counter at each site to track the number of slots an item has appeared in. In the distributed case, overlapping occurrences of the same item in the same timeslot across different sites should still be discounted. Hence, a simple extension of the centralized algorithm will not work here.

In our approach, we first reduce the number of items that are tracked using a hash-based random sampler, similar to the one used in the centralized streaming algorithm in [24]. This sampler is used to (with high probability) selectively maintain state for only those items whose persistence crosses a given threshold. The threshold is chosen such that an α-persistent item is very likely to be tracked. Once tracking state has been established for an item, we still need to maintain its persistence as more copies of this item arrive. We could potentially do this through maintaining for each such item, a list of timeslots where the item has appeared. When a query for persistent items is posed, the lists for different tracked items is sent to the coordinator,

who computes the union of different lists across all the sites, to compute the persistence of the tracked item. However, this naive approach leads to a high memory requirement and communication cost.

We reduce the memory and communication cost through using a distributed distinct counting algorithm [15, 21, 3, 7, 17, 4, 37, 22] to maintain, in a coordinated manner, a count of distinct timeslots of occurrence for each tracked item across all the sites. A distributed distinct counting algorithm estimates the number of distinct elements in the union of multiple distributed streams. More precisely, if $dist(\mathcal{S})$ is the number of distinct elements in a distributed stream $\mathcal{S}$, then given a relative error $0 < \gamma < 1$ and an error probability $0 < \upsilon < 1$, a (γ, υ)-approximate distinct counting algorithm returns an estimate X such that $\Pr[|dist(\mathcal{S}) - X| > \gamma.dist(\mathcal{S})] \leq \upsilon$. The algorithm that we use from [17, 4], is practical, and has an overall space requirement of $O\left(\frac{\log 1/\upsilon}{\gamma^2}\right)$ words.

With our approach, there are two sources of error in the estimated persistence of an item. (1) We do not track the persistence of each item, but only those which pass through the sampler. While this results in reduced communication when compared with tracking the persistence of each item, it also results in an error in the measured persistence of each item, even for those items that are tracked. (2) After tracking state has been established for an item, the overall persistence in the distributed stream in forthcoming timeslots is only computed approximately. Our analysis ensures that the combined error from these two sources does not exceed the desired threshold. To achieve this, the total error budget is divided among the two sources of error such that the communication cost is minimized and the final approximation guarantees are achieved. Our analysis for the infinite window case the sliding window case are described in Sections 2 and 3 respectively.

1.4. Related Work

Prior work on identifying persistent items in a stream has considered the centralized case. This includes work by Giroire et al. [19], who track persistent items in a centralized stream by exactly computing the persistence of each distinct item in the stream, and an improved small space approximation algorithm by Lahiri et al. [24].

A frequent item or a "heavy hitter" in a stream is one whose frequency in the stream is significant when compared with the volume of the stream. There is much prior work on identifying frequent items or heavy hitters from a data stream, including [9, 29, 27, 28, 33, 35, 25]. As discussed earlier, a frequent item may not be persistent, and a persistent item in not necessarily frequent either. Frequent item identification algorithms that are based on "sketches", such as the Count Sketch [8] and the Count-Min Sketch [10], can be implemented in a distributed manner. These algorithms maintain multiple counters, each of which is the sum of many random variables. The sketch for the union of several streams is simply the sum of the

sketches over all the streams. However, adapting these algorithms for the case of persistent items does not seem to be easy since these sketches count the number of occurrences (frequency) as opposed to the number of occurrences in distinct timeslots (persistence).

Roadmap. We present the algorithm for tracking persistent items in an infinite window in Section 2, followed by the algorithm for a sliding window in Section 3, and the experimental evaluation in Section 4.

2. Infinite Window

We now present an algorithm for the case of an infinite window, i.e. when the data of interest is the union of all items from the beginning of time, that arrived across all streams.

Intuition: To reduce space and communication, the first step is to avoid tracking every item, especially items with a low value of persistence. While tracking items with a low persistence cannot be completely avoided, it can be reduced through sampling. Sites 1 through k share a common hash function $h : ([1, m] \times [1, n]) \rightarrow (0, 1)$. For two tuples (d_1, t_1) and (d_2, t_2) that are unequal either in one attribute or both, $h(d_1, t_1)$ and $h(d_2, t_2)$ are independent random values chosen uniformly at random from the interval of real numbers $(0, 1)$.

Each site i maintains state for a subset of items that have arrived so far. When an item (d, t) arrives in $\mathcal{S}_i$, if d is already being tracked by i, then the state corresponding to d is updated by adding t to the set of time slots that d has appeared in. If d is not being tracked by i, then tracking state is established for d if $h(d, t) < \tau$ (for a value τ to be decided), and a message is sent to all sites to start tracking d. Clearly, if an item d appears in time slot t, tracking state is established for d with probability τ. We note the following.

- Multiple occurrences of d within the same time slot t do not increase the probability of d being tracked.
- A low-persistence item d which appears only in a few distinct time slots is not likely to be tracked. On the other hand, a high-persistence item d' which appeared in many distinct slots will be tracked with a high probability.
- Since the same hash function h is shared by all sites, the result after distributed occurrences of d is the same as if d was being observed by the same site.

Once tracking state has been established for an item d, future occurrences of d in subsequent time slots are treated without needing further communication among the sites. A challenge here is that even with state maintained at different nodes for an item d, it is still non-trivial to track the number of occurrences of d in distinct time slots. For this purpose, we use a distributed distinct counter, from [17]. Equivalently, we could

use other algorithms for distinct counting that can be implemented in a distributed setting, such as the one by Bar-Yossef et al. [4]. We use the one in [17] because it is simple and gives very good practical performance. The accuracy and error probability of this distinct counter influences the overall space complexity of our algorithm. Before we present the formal algorithm description, we present the guarantees expected from the distinct counter.

When a query is posed for the set of persistent items, the coordinator combines the estimates of all the distributed distinct counters to compute an estimate of the persistence of each item being tracked. This estimate is used to decide whether or not an item is persistent. There are two sources of error in this estimator: (1) the error due to sampling, before the item starts being tracked, and (2) the error due to the approximate distinct counter for the item which is already being tracked. We first present the guarantees provided by a distributed distinct counting algorithm. For a relative error parameter $0 < \gamma < 1$ and an error probability parameter $0 < \upsilon < 1$, a distinct counter $\mathcal{D}_\gamma^\upsilon$ takes as input a stream of updates S and maintains an estimate of $dist(S)$, the number of distinct items in S.

Theorem 1 ([17]). *There is a distinct counter $\mathcal{D}_\gamma^\upsilon$ that takes space $O\left(\frac{\log 1/\upsilon}{\gamma^2}\right)$ words of space, and whenever a query is asked for $dist(S)$, returns an estimate X such that $\Pr[|X - dist(S)| > \gamma \cdot dist(S)] \leq \upsilon$, for $0 < \gamma < 1$ and $0 < \upsilon < 1$. This distinct counter can handle distributed updates, and the distributed state can be combined together at the end of observation.*

Note that we express the space complexity above in terms of the number of words, assuming that each item identifier and timestamp can be stored in a constant number of words.

Algorithm Description: The inputs to our algorithm are: 1) m - domain size of the identifiers, 2) n - total number of timeslots in the distributed stream, 3) α - persistence threshold, 4) ϵ - approximation parameter, and 5) δ - error probability. The distributed algorithm has three parts: Algorithm 1 defines the input parameter, and describes the initialization of the datastructures and global variables used, Algorithm 2 describes the algorithm at each local site, and Algorithm 3 describes the algorithm at the coordinator node C.

Each site i maintains a sketch S_i for stream $\mathcal{S}_i$ seen so far, comprising of tuples of the form $(d, \mathcal{D}_{\delta_2}^{\epsilon_2}[i](d))$. Here, d is an item ID, and $\mathcal{D}_{\delta_2}^{\epsilon_2}[i](d)$ is the distinct counting datastructure maintained for estimating the number of distinct time slots when the item d appeared. The distinct counting data structure is maintained using the distributed distinct counting algorithm with approximation parameter ϵ_2 and error probability δ_2. Here, ϵ_2 and δ_2 are parameters whose values are determined based on optimizing communication cost while obeying the correctness constraints

The coordinator C maintains a sketch S which has tuples of the form (d, t_d) where d is the item ID and

t_d is the time when the item d was first tracked in the distributed stream. When an item (d, t) arrives at site i in timeslot t, then the algorithm first looks into S_i to check if d is being tracked (Algo 2: line 2). If not, then (d, t) is passed through the hash function h. The algorithm starts tracking d if $h(d, t) < \tau$, where $\tau = 2/(\epsilon_1 n)$ and $\epsilon_1 = c_\epsilon \epsilon$ (Algo 2: line 3), where $0 < c_\epsilon < 1$ is a constant; note this happens with probability τ. Site i communicates with the other sites (Algo 2: lines 4-5) and the coordinator C (Algo 2: line 6) to inform them about the newly tracked item. Once site i starts tracking the item d, it makes an entry of the form $(d, \mathcal{D}^{\epsilon_2}_{\delta_2})$ in S_i and starts maintaining a distinct count datastructure $\mathcal{D}^{\epsilon_2}_{\delta_2}[i](d)$ for item d (Algo 2: line 8). Once the item is tracked, with every appearance of the item in a new timeslot, $\mathcal{D}^{\epsilon_2}_{\delta_2}[i](d)$ is updated using the distinct counting algorithm (Algo 2: line 10).

If site i receives information about a newly tracked item d from some other site, it starts tracking d and creates a new local entry $(d, \mathcal{D}^{\epsilon_2}_{\delta_2})$ in S_i (Algo 2: lines 12-15). It then updates $(d, \mathcal{D}^{\epsilon_2}_{\delta_2})$ in S_i for d with every appearance of item d in a new timeslot in site i (Algo 2: line 8). When the coordinator receives information about a newly tracked item d from any of the sites, it makes a new entry (d, t_d) in S (Algo 3: line 1).

When a query is made to the coordinator C, for each tracked item $d \in S$, it takes a union of the corresponding distinct count data structure $(d, \mathcal{D}^{\epsilon_2}_{\delta_2})[j](d)$ across all sites as per the distinct counting algorithm to estimate the number of distinct slots $\hat{n}_d$, where item d appeared in the distributed stream since it was tracked (Algo 3: lines 3-6). While we do not know how many distinct slots d may have appeared in before it was first tracked, we can see this number is a geometric random variable X with parameter τ; we estimate the value of X using its expectation. We estimate the persistence of d, equal to the total number of distinct slots where d appeared in the entire distributed stream, as $\hat{p}_d$ as $\hat{n}_d + E(X) = \hat{n}_d + 1/\tau$ (Algo 3: line 10). Also, in order to optimize our results, we compute $\hat{p}_d$ as $\hat{n}_d + t_d$ for the condition $(1/\tau) > t_d$, (Algo 3: lines 7-8).

Algorithm 1: Infinite Window : Initialization

Input: m - Domain Size of identifiers; n - Total no. of time slots; α - persistence threshold; ϵ - error parameter; δ - error probability

1 Hash function $h : ([1, m] \times [1, n]) \rightarrow (0, 1)$
2 Approximation Parameters: $\epsilon_1 \leftarrow c_\epsilon \epsilon$, $\epsilon_2 \leftarrow (1 - c_\epsilon)\epsilon/4\alpha$ `// 0 < c_ε < 1 is a constant`
3 Error Probability: $\delta_2 = c_\delta \delta$ `//` $0 < c_\delta \leq min\left(1, \frac{2}{\log(1/\delta)}\right)$ `is a constant`
4 Filter parameter $\tau \leftarrow \frac{2}{\epsilon_1 n}$
5 Threshold $T \leftarrow (1 - \epsilon_2)(\alpha n - \frac{1}{\tau} + 1)$
6 $S \leftarrow \emptyset$
7 **for** $i = 1, 2, .., k$ **do**
8 $\quad S_i \leftarrow \emptyset$

Algorithm 2: Infinite Window: Algorithm at node i

On receiving item (d, t) **at node** i
if *(*$d \notin S_i$*)* **then**
  if $h(d, t) < \tau$) **then**
    for *every node* $j = 1 \dots k$, $j \neq i$ **do**
      Send "Start Tracking (d, t)" to j
    Send (d, t) to the coordinator
    $S_i \leftarrow S_i \cup \{(d, \mathcal{D}_{\delta_2}^{\epsilon_2}[i](d))\}$
    Insert t into $\mathcal{D}_{\delta_2}^{\epsilon_2}[i](d)$
else `// if` $d \in S_i$
  Insert t into $\mathcal{D}_{\delta_2}^{\epsilon_2}[i](d)$

On receiving message "Start Tracking (d, t)**"**
`// Create a new data structure tracking` d
$S_i \leftarrow S_i \cup \{(d, \mathcal{D}_{\delta_2}^{\epsilon_2}[i](d))\}$
Insert t into $\mathcal{D}_{\delta_2}^{\epsilon_2}[i](d)$

Algorithm 3: Infinite Window: Algorithm at the coordinator C.

Upon receiving (d, t_d)**:** Insert (d, t_d) into S
Upon receiving a query for the set of Persistent Items:
for *each* $(d, t_d) \in S$ **do**
  for $i = 0, 1, ..., k-1$ **do**
    Compute the union of $D_{\epsilon_2}^{\delta_2}[i](d)$ data structures over all sites i.
    Let $\hat{n}_d$ be the estimate of the distinct count over this union.
  if $t_d < (1/\tau)$ **then**
    $\hat{p}(d) \leftarrow \hat{n}_d + t_d$
  else
    $\hat{p}(d) \leftarrow \hat{n}_d + \frac{1}{\tau}$
  if $\hat{p}(d) \geq T$ **then**
    Report d as α-persistent

2.1. Infinite Window : Correctness

Let $G(\tau)$ be the geometric random variable with parameter τ. Let $p(d)$ denote the persistence of item d, and n_d denote the number of distinct slots where d appeared in $\mathcal{S}$ after (and including) the time slot when the algorithm started tracking d.

Lemma 1. *If $G(\tau) \leq p(d)$, then $n_d = p(d) - G(\tau) + 1$, else $n_d = 0$.*

Proof. Let the distinct time slots that d appears in distributed stream be $t_1, t_2, \ldots$, in increasing order. d is not tracked until we reach a time slot t_i such that $h(d, t_i) < \tau$. The number of time slots required for this to occur is $G(\tau)$. Note this is true even though the different sites are observing the tuples in a distributed manner, since their decisions are based on the output of a hash function on (d, t). The expression for n_d follows. □

Lemma 2. ***False Negative:*** *If an item d is α-persistent, then the probability that it is not reported by the coordinator in Algorithm 3 is at most $\left(e^{-2} + \frac{2\delta}{\log(1/\delta)}\right)$.*

Proof. Consider an α-persistent item d. Let A denote the event that d is not reported. Let $\hat{p}(d)$ be the estimate of its persistence at the end of observation. Also, let $\hat{n}_d$ be an estimate of n_d, the number of distinct time slots where d appeared in $\mathcal{S}$ after being tracked. $\hat{n}_d$ is obtained from the union of the distinct count datastructures over the k sites. Per the above algorithm, $\hat{p}(d) = \hat{n}_d + 1/\tau$, and d is not reported if $\hat{p}(d) < T$. Consider that $T = (1 - \epsilon_2)(\alpha + 1 - 1/\tau)$ and $\delta_2 = c_\delta \delta$ where $c_\delta \leq \frac{2}{\log(1/\delta)}$.

$$\Pr[A] = \Pr[\hat{p}(d) < T] = \Pr\left[\hat{n}_d < \left(T - \frac{1}{\tau}\right)\right]$$

Let B denote the event $(1 - \epsilon_2) n_d \leq \hat{n}_d$. We have the following:

$$\Pr[A] = \Pr[A|B]\Pr[B] + \Pr[A|\bar{B}]\Pr[\bar{B}] \leq \Pr[A|B] + \Pr[\bar{B}] \tag{1}$$

$$
\begin{aligned}
\Pr[A|B] &= \Pr\left[\hat{n}_d < \left(T - \frac{1}{\tau}\right) \Big| (1-\epsilon_2)n_d \le \hat{n}_d\right] \\
&\le \Pr\left[(1-\epsilon_2)n_d < \left(T - \frac{1}{\tau}\right)\right] \\
&= \Pr\left[p(d) - G(\tau) + 1 < \frac{(T - 1/\tau)}{(1-\epsilon_2)}\right] \qquad \text{using Lemma 1} \\
&= \Pr\left[G(\tau) > p(d) + 1 - \frac{T}{(1-\epsilon_2)} + \frac{1}{(1-\epsilon_2)\tau}\right] \\
&\le \Pr\left[G(\tau) > \alpha n + 1 - \left(\alpha n + 1 - \frac{1}{\tau}\right) + \frac{1}{(1-\epsilon_2)\tau}\right] \qquad \text{substituting } T \text{ and given } p(d) \ge \alpha n \\
&\le \Pr\left[G(\tau) > \frac{2}{\tau}\right] \qquad = (1-\tau)^{\frac{2}{\tau}} \\
&\le e^{-2} \qquad \text{since } (1-\tau)^{\theta} \le e^{-\tau\theta}
\end{aligned}
$$

The probability of $\bar{B}$ depends on the guarantee given by the distinct counter $\mathcal{D}_{\epsilon_2}^{\delta_2}$. Note that the number of insertions into the distinct counter is n_d, and the estimate returned by the distinct counter is $\hat{n}_d$. Using Theorem 1, we have: $\Pr[(1-\epsilon_2)n_d \le \hat{n}_d \le (1+\epsilon_2)n_d] \ge (1-\delta_2)$. Hence,

$$
\begin{aligned}
\Pr[\bar{B}] &= \Pr[\hat{n}_d < (1-\epsilon_2)n_d] \\
&\le \Pr[\hat{n}_d < (1-\epsilon_2)n_d] + \Pr[\hat{n}_d > (1+\epsilon_2)n_d] \\
&\le \delta_2 \qquad \le \frac{2\delta}{\log(1/\delta)} \qquad \text{given } \delta_2 = c_\delta \delta \le \frac{2\delta}{\log(1/\delta)}
\end{aligned}
$$

Using these back in Equation 1, we get the desired result. □

Lemma 3. False Positives: *An item d with persistence $p(d) < (\alpha - \epsilon)n$ is reported by the coordinator in Algorithm 3 with probability at most $\frac{2\delta}{\log(1/\delta)}$.*

Proof. Consider an item d with persistence $p(d) < (\alpha - \epsilon)n$. Let A denote the event that d is reported as being α-persistent. If $\hat{p}(d)$ is the estimate of persistence of d at the end of observation, then $\hat{p}(d) > T$. Also, per the algorithm, $\hat{p}(d) = \hat{n}_d + 1/\tau$.

$$
\Pr[A] = \Pr\left[\hat{p}(d) > T\right] = \Pr\left[\hat{n}_d > \left(T - \frac{1}{\tau}\right)\right]
$$

Let B denote the event $\hat{n}_d \le (1+\epsilon_2)n_d$. If $T = (1-\epsilon_2)\left(\alpha n + 1 - \frac{1}{\tau}\right)$, $\epsilon_1 = c_\epsilon \epsilon$, and $\epsilon_2 = \frac{\epsilon(1-c_\epsilon)}{4\alpha}$, where, c_ϵ is a constant s.t. $0 < c_\epsilon < 1$ then,

$$
\begin{aligned}
\Pr[A|B] &= \Pr\left[\hat{n_d} > T - \frac{1}{\tau} \middle| \hat{n}_d \leq (1+\epsilon_2)n_d\right] \\
&\leq \Pr\left[(1+\epsilon_2)n_d \geq T - \frac{1}{\tau}\right] \\
&= \Pr\left[p(d) - G(\tau) + 1 \geq \left(\frac{1-\epsilon_2}{1+\epsilon_2}\right)\left(\alpha n + 1 - \frac{1}{\tau}\right) - \frac{1}{(1+\epsilon_2)\tau}\right] \qquad \text{substituting } T \text{ and using Lemma 1} \\
&= \Pr\left[G(\tau) \leq p(d) + 1 - \left(\frac{1-\epsilon_2}{1+\epsilon_2}\right)\left(\alpha n + 1 - \frac{1}{\tau}\right) + \frac{1}{(1+\epsilon_2)\tau}\right] \\
&\leq \Pr\left[G(\tau) \leq (\alpha n - \epsilon n) + 1 - \left(\frac{1-\epsilon_2}{1+\epsilon_2}\right)(\alpha n + 1) + \frac{2-\epsilon_2}{(1+\epsilon_2)\tau}\right] \qquad \text{given } p(d) < (\alpha - \epsilon)n \\
&\leq \Pr\left[G(\tau) \leq (\alpha n - \epsilon n) + 1 - (1-2\epsilon_2)(\alpha n + 1) + \frac{2}{\tau}\right] \qquad \text{as, } (1-2\epsilon_2) < (1-\epsilon_2)/(1+\epsilon_2) \\
&= \Pr\left[G(\tau) \leq 2\epsilon_2 \alpha n + 2\epsilon_2 + \frac{2}{\tau} - \epsilon n\right] \\
&\leq \Pr\left[G(\tau) \leq 4\epsilon_2 \alpha n + \epsilon_1 n - \epsilon n\right] = 0 \qquad \text{using } \tau = 2/\epsilon_1 n, \text{ and, } (2\epsilon_2 \alpha n + 2\epsilon_2 \leq 4\epsilon_2 \alpha n)
\end{aligned}
$$

Using Theorem 1, we have: $\Pr[(1-\epsilon_2)n_d \leq \hat{n}_d \leq (1+\epsilon_2)n_d] \geq (1-\delta_2)$ Hence,

$$
\begin{aligned}
\Pr[\bar{B}] &= \Pr[\hat{n}_d > (1+\epsilon_2)n_d] \\
&\leq \Pr[\hat{n}_d < (1-\epsilon_2)n_d] + \Pr[\hat{n}_d > (1+\epsilon_2)n_d] \\
&\leq \delta_2 \qquad \leq \frac{2\delta}{\log(1/\delta)}
\end{aligned}
$$

Using the relation $\Pr[A] \leq \Pr[A|B] + \Pr[\bar{B}]$, we get the desired result. We obtain that for $c_\epsilon = 1/3$, the space cost and the communication cost of this algorithm is optimized. □

Theorem 2. *By running at least $\frac{\log \delta}{\log(e^{-2}+c_\delta \delta)}$ parallel instances of the algorithm, we get the following guarantee:*

1. *An item d with persistence $p(d) \geq (\alpha n)$ is reported as α-persistent with probability at least $1-\delta$.*
2. *An item d with persistence $p(d) < (\alpha - \epsilon)n$ is not reported as persistent with probability at least $1-\delta$.*

Proof. Let $\theta = \frac{\log \delta}{\log(e^{-2}+c_\delta \delta)}$. We return the union of all persistent items returned by all the parallel instances. For an α-persistent item d, the probability that d is not reported is equal to the probability that it is not reported by any of the θ instances. This probability is no more than $(e^{-2} + c_\delta \delta)^\theta$, which is bounded by δ (where $0 < c_\delta \leq \frac{2}{\log(1/\delta)}$).

Consider an item d with persistence less than $(\alpha - \epsilon)n$. The probability that d is reported is the probability that d is reported by at least one of the θ parallel instances. Using the union bound, this probability is no

more than $\frac{2\delta\theta}{\log(1/\delta)}$. Upon substituting θ, we get the desired result. □

2.2. Infinite Window: Complexity

We present an analysis of the communication and space complexity of the algorithm for an infinite window. Let P be the sum of the persistence of all the distinct items in the distributed stream, n be the total number of time slots in the stream. Recall that k is the number of sites.

Theorem 3. *The expected space complexity of the distributed algorithm per site is* $O\left(\frac{\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$, *and the expected space complexity over all sites is* $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$

Proof. The space complexity of tracking a single item is equal to the cost of an approximate distinct count data structure $\mathcal{D}_{\epsilon_2}^{\delta_2}$, for maintaining the number of distinct time slots for the item. Let $Z(d)$ be a random variable for item d such that $Z(d) = 1$ if item d is tracked, else $Z(d) = 0$.

$$\begin{aligned}\Pr[Z(d)=1] &= 1 - \Pr[Z(d)=0] = 1-(1-\tau)^{p(d)} \\ &\leq 1 - e^{-2\tau p(d)} \quad \text{using Taylor's expansion} \\ &\leq 2\tau p(d) \leq 1-(1-2\tau p(d)) \leq 2\tau p(d)\end{aligned}$$

We know that space taken by distinct count operator for each item d at each site is $O\left(\frac{\log(1/\delta_2)}{{\epsilon_2}^2}\right)$ (Theorem 1). The expected space taken by an item d per site is:

$$\begin{aligned}\Pr[Z(d)=1]\left(\frac{\log(1/\delta_2)}{{\epsilon_2}^2}\right) &\leq \frac{2\tau p(d)\log(1/\delta_2)}{{\epsilon_2}^2} = \frac{4p(d)\log(1/\delta_2)}{{\epsilon_2}^2\epsilon_1 n} \\ &= O\left(\frac{p(d)\log(1/\delta_2)\alpha^2}{\epsilon^3 n}\right) \quad \text{since } \epsilon_2 = O\left(\frac{\epsilon}{\alpha}\right); \epsilon_1 = O(\epsilon); \delta_2 = O(\delta) \\ &\leq O\left(\frac{p(d)\log\left(\frac{\log(1/\delta)}{2\delta}\right)\alpha^2}{\epsilon^3 n}\right) \qquad \text{given } \delta_2 = c_\delta\delta \leq \frac{2\delta}{\log(1/\delta)}\end{aligned}$$

Hence, total expected space taken by the algorithm per site is:

$$= O\left(\sum_d \frac{p(d)\log\left(\frac{\log(1/\delta)}{2\delta}\right)\alpha^2}{\epsilon^3 n}\right) = O\left(\frac{\alpha^2\log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$$

The total space over the entire distributed algorithm is k times the space cost of each site. □

Theorem 4. ***Communication:*** *The expected communication complexity of the distributed algorithm taken over all sites is* $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$ *bytes.*

Proof. For each item that is tracked, the algorithm incurs $O(k)$ messages to begin tracking the item. Finally, in order to identify persistent items, it is necessary to have another round of communication among all the sites and the coordinator. The total number of messages exchanged is thus $O(kN)$ where N is the number of items that are tracked. Since we know that $\mathbb{E}[E] = O\left(\frac{P}{\epsilon n}\right)$ (see the proof in Theorem 3), the expected number of messages communicated over all sites is $O(\frac{kP}{\epsilon n})$.

If we consider the number of bytes communicated, we find that each item leads to a communication of $O(\frac{\alpha^2 \log(1/\delta_2)}{\epsilon^2})$ bytes, due to the distinct count data structure. Hence, the expected number of message bytes communicated between the sites and the coordinator is $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$ □

3. Sliding Window

At time slot c, the current window of size n is defined as the set of all events within the n most recent time slots, i.e. slots $(c - n + 1)$ to c, both endpoints inclusive. An item d is defined to be α-persistent in a sliding window of size n if it occurred in at least αn distinct time slots within the current window. We now present a distributed algorithm for approximately tracking the set of all α-persistent items within the sliding window of size n.

Intuition: The sliding window algorithm uses the same sampling technique as for the infinite window case, and if a site decides to track an item, it communicates with the other sites, following which each site sets up local state for this item, and future occurrences of this item are handled locally without requiring further communication. The main challenge with the sliding window case is that as future time slots arrive, old occurrences go out of scope and have to be removed from consideration from the data structures. At each site i, S_i is continuously updated to discard expired time slots for each item.

Unlike the algorithm for infinite window, for each item d that is tracked, the error due to sampling is a concern only as long as the starting time slot for tracking d, i.e. t_d, does not expire from the sliding window. After slot $t_d + n$, a summary of all subsequent occurrences of d are tracked approximately by the data structure. Thus the query processing will distinguish between the cases when the query is made after $t_d + n$ (Algo 6: lines 11-13), and when the query is made before $t_d + n$ (Algo 6: lines 6-10). Another change is that the distinct elements algorithm should work over sliding windows, rather than for the entire stream.

Algorithm Description: Similar to the infinite window version of persistence algorithm (Section 2), the sliding window algorithm has three parts: Algorithm 4 initializes data structures, Algorithm 5 is the algorithm run at each site, and Algorithm 6 gives the algorithm run at the coordinator node C. The input to our algorithm is the same as that of infinite window version, except that n, in this case, is the maximum number of timeslots in a window (Algorithm 4).

The algorithm (Algo 5) used at each site i is similar to the one used for infinite window. However, the distinct count data structure is now maintained by the distinct counting algorithm for a sliding window [13, 18, 40].

When a query is made, the coordinator C takes a union of the distinct count datastructures for the k sites (Algo 6: lines 2-5) and computes $\hat{n}_d$, number of distinct slots when d appeared in the current window, per the distinct counting algorithm for sliding windows. As discussed above, the query processing in our algorithm distinguishes between the cases when the query is made after $t_d + n$ (Algo 6: lines 11-13), and when the query is made before t_d+n (Algo 6: lines 6-10). If a query is made before t_d+n, then the persistent items are tracked in the same way as done by the infinite window version of the algorithm, Algo 3 . However, if a query is made after $t_d + n$, then persistence of an item d is estimated as $\hat{n}_d$.

For relative error parameter $0 < \gamma < 1$ and an error probability parameter $0 < \upsilon < 1$, a distinct counter $\mathcal{D}^{\upsilon}_{\gamma}$ takes as input a stream of updates S and at any given time t maintains an estimate of $dist(\mathcal{S})$, the number of distinct elements in $\mathcal{S}$, for the elements that occurred in most recent n slots.

Theorem 5 ([18, 13, 40])**.** *There is a distinct counter $\mathcal{D}^{\upsilon}_{\gamma}$ that takes space $O(\frac{\log 1/\upsilon}{\gamma^2})$ words of space, and whenever a query is asked for $dist(\mathcal{S})$ in the most recent n time slots, returns an estimate X such that $\Pr[|X - dist(\mathcal{S})| > \gamma \cdot dist(\mathcal{S})] \leq \upsilon$, where $0 < \gamma < 1$ and $0 < \upsilon < 1$.*

A detailed description of the algorithm is presented in Algorithms 4, 5, and 6.

Algorithm 4: Sliding Window: Initialization

Input: m - Domain Size of identifiers; n- maximum no. of time slots in a window; α - persistence threshold; ϵ - error parameter; δ - error probability;

1 Hash function $h : ([1, m] \times [1, n]) \rightarrow (0, 1)$
2 Approx. parameters $\epsilon_1 \leftarrow c_\epsilon\epsilon$; $\epsilon_2 \leftarrow (1 - c_\epsilon)\epsilon/4\alpha$ `// 0 < c_ε < 1 is a constant`
3 Error Probability: $\delta_2 = c_\delta\delta$ `//` $0 < c_\delta \leq min\left(1, \frac{2}{\log(1/\delta)}\right)$ `is a constant`
4 Filter parameter $\tau \leftarrow \frac{2}{\epsilon_1 n}$
5 Threshold $T \leftarrow (1 - \epsilon_2)(\alpha n + 1 - \frac{1}{\tau})$
6 Sketch at coordinator $S \leftarrow \emptyset$
7 **for** *each site* $i = 1 \ldots k$ **do**
8 $\quad S_i \leftarrow \emptyset$

3.1. Sliding Window : Correctness

Let t_d be the slot when item d started to be tracked. Also, for a query q made on the distributed streams, let t_q be the last slot of the most recent window $[t_q - n + 1, t_q]$ on which query q has been posed.

Lemma 4. *Let $G(\tau)$ be the geometric random variable with parameter τ. Also, let n_d denote the number of distinct slots in the distributed streams where d appears in current window after being tracked by the*

Algorithm 5: Sliding Window: Algorithm at node i

On receiving item (d,t) at node i
if $(d,t) \notin S_i$ **then**
    if $h(d,t) < \tau$) **then**
        for *every node* $j = 1 \dots k$, $j \neq i$ **do**
            Send "Start Tracking (d,t)" to j
        Send (d,t) to the coordinator
        `// Create a new data structure for` d
        $S_i \leftarrow S_i \cup \{(d, \mathcal{D}_{\delta_2}^{\epsilon_2}[i](d))\}$
        Insert t in $\mathcal{D}_{\delta_2}^{\epsilon_2}[i](d))$
else `// if` $d \in S_i$
    Insert t in $\mathcal{D}_{\delta_2}^{\epsilon_2}[i](d)$

On receiving message "Start Tracking (d,t)**"**
`// Create a new data structure for` d
$S_i \leftarrow S_i \cup \{(d, \mathcal{D}_{\delta_2}^{\epsilon_2}[i](d))\}$
Insert t in $\mathcal{D}_{\delta_2}^{\epsilon_2}[i](d)$

Algorithm 6: Sliding Window:Algorithm at the coordinator C:

On receiving tuple (d,t_d) **:** $S \leftarrow S \cup \{(d,t)\}$.
On receiving a query for the set of Persistent Items: for *each* $(d,t_d) \in S$ **do**
    for $i = 1 \dots k$ **do**
        Compute the union of $D_{\epsilon_2}^{\delta_2}[i](d)$ data structures over all sites i.
        Let $\hat{n}_d$ be the estimate of the distinct count over this union.
    if $t \leq t_d + n$ **then** `//` t `- current slot`
        if $t_d < (1/\tau)$ **then**
            $\hat{p}(d) \leftarrow \hat{n}_d + t_d$;
        else
            $\hat{p}(d) \leftarrow \hat{n}_d + (1/\tau)$;
    else
        $\hat{p}(d) \leftarrow \hat{n}_d$;
        $T \leftarrow (1-\epsilon_2)\alpha n$;
    if $\hat{p}(d) \geq T$ **then**
        Report d as α-persistent item;

algorithm. For each item d, n_d can be expressed differently depending on t_q and $G(\tau)$ in the following manner.

1. **If** $t_q \leq t_d + n$*: if* $G(\tau) \leq p(d)$*,* $n_d = p(d) - G(\tau) + 1$*, else* $n_d = 0$*.*
2. **If** $t_q > t_d + n$*:* $n_d = p(d)$*.*

Proof. The proof of the above Lemma is divided into two parts, for the two cases described above. Proof of part 1) is the same as that of proof of Lemma 1.

For part 2), at $(t_d + n)$-th slot, the first slot when d was tracked expires, i.e. t_d expires. From the slot t_d onwards, every occurrence of d is tracked. Hence, if the current window of the most recent n slots does not include t_d, then persistence of d, $p(d)$, over the current window is the number of occurrences of d in the current window, i.e. $p(d) = n_d$. □

Lemma 5. Low False Negative: *An α-persistent item d having a persistence $p(d) \geq \alpha n$ during the most recent n slots is not reported as α-persistent by the coordinator in Algorithm 6 with a probability at most*

1. $e^{-2} + \frac{2\delta}{\log(1/\delta)}$ *if* $t_q \leq t_d + n$
2. $\frac{2\delta}{\log(1/\delta)}$ *if* $t_q > t_d + n$

Proof. Consider an α-persistent item d, with persistence $p(d) \geq \alpha n$.

1. **If** $t_q \leq t_d + n$**:** Proof is same as that of Lemma 2, where n is the maximum number of slots in current window instead of the entire distributed stream. Note that $\delta_2 = c_\delta \delta$. The error probability can be reduced to δ by running $\frac{\log(\delta)}{\log(e^{-2}+c_\delta \delta)}$ parallel instances.
2. **If** $t_q > t_d + n$ Let A denote the event that d is not reported, i.e. the event that false negative occurs. Using the proof in Lemma 4, we can also conclude that the estimate of persistence of d, $\hat{p}(d)$, in the most recent n slots is the estimate returned by the distinct counter for item d, $\hat{n}_d$, to approximate the count of distinct number of slots where d occurred over the most recent n slots, i.e. $\hat{p}(d) = \hat{n}_d$. Per the above algorithm, item d is not reported as α-persistent if $\hat{p}(d) < T$. $\Pr[A] = \Pr[\hat{p}(d) < T] = \Pr[\hat{n}_d < T]$. Let B denote the event $(1 - \epsilon_2)n_d < \hat{n}_d$. From the above algorithm, $T = (1 - \epsilon_2)\alpha n$.

$$
\begin{aligned}
\Pr[A|B] &= \Pr[\hat{n}_d < T | (1-\epsilon_2)n_d < \hat{n}_d] \\
&\leq \Pr[(1-\epsilon_2)n_d < T] \\
&= \Pr[(1-\epsilon_2)p(d) < T] \qquad \text{using Lemma 4} \\
&\leq \Pr[(1-\epsilon_2)\alpha n < T] \qquad \text{given } p(d) \geq \alpha n \\
&= 0 \qquad \text{substituting } T
\end{aligned}
$$

The probability of $\bar{B}$ depends on the guarantee given by the distinct counter $\mathcal{D}_{\epsilon_2}^{\delta_2}$. Using Theorem 5 we have: $\Pr[(1-\epsilon_2)n_d \leq \hat{n}_d \leq (1+\epsilon_2)n_d] \geq (1-\delta_2)$ Hence,

$$\begin{aligned}\Pr[\bar{B}] &= \Pr[\hat{n}_d < (1-\epsilon_2)n_d] \\ &\leq \Pr[\hat{n}_d < (1-\epsilon_2)n_d] + \Pr[\hat{n}_d > (1+\epsilon_2)n_d] \\ &\leq \delta_2 \qquad \leq \frac{2\delta}{\log(1/\delta)}\end{aligned}$$

Using $\Pr[A] \leq \Pr[A|B] + \Pr[\bar{B}]$, we get the desired result.

□

Lemma 6. Low False Positive: *An item d which is far from persistent in the most recent n slots, i.e, whose persistence $p(d) < (\alpha-\epsilon)n$ in the current window of size n, is reported as α-persistent with probability at most $\frac{2\delta}{\log(1/\delta)}$.*

Proof. Consider an item d with persistence $p(d) < (\alpha-\epsilon)n$. d is far from persistent. The proof has two cases based on when query is posed with respect to t_d, the slot when d is first tracked by algorithm.

1. **If $t_q \leq t_d + n$:**, proof of above lemma is the same as that of Lemma 3, n denoting the maximum number of slots in a sliding window, and not the total number of slots in the entire distributed stream.
2. **If $t_q > t_d + n$:** Persistence of an item d is the number of occurrences of d within the current window which equals n_d, i.e. $p(d) = n_d$ from Lemma 4. Also, $\hat{p}(d) = \hat{n}_d$. Let A denote the event that d is reported, i.e. the false positive occurs. Also, per the above algorithm, d is reported if $\hat{p}(d) \geq T$.

$$\Pr[A] = \Pr[\hat{p}(d) \geq T] = \Pr[\hat{n}_d \geq T]$$

Let B denote the event $(1+\epsilon_2)n_d \geq \hat{n}_d$. We have $T = (1-\epsilon_2)\alpha n$ and $\epsilon_2 = (1-c_\epsilon)\epsilon/4\alpha$.

$$\begin{aligned}\Pr[A|B] &= \Pr[\hat{n}_d \geq T | (1+\epsilon_2)n_d \geq \hat{n}_d] \\ &\leq \Pr[(1+\epsilon_2)n_d \geq T] \\ &= \Pr[(1+\epsilon_2)p(d) \geq T] \\ &\leq \Pr\left[(\alpha-\epsilon)n \geq \frac{T}{(1+\epsilon_2)}\right] \qquad \text{given } p(d) < (\alpha-\epsilon)n \\ &\leq \Pr\left[0 \leq \alpha n - \epsilon n - (1-2\epsilon_2)\alpha n\right] \qquad \text{substituting } T, \text{ and } \tfrac{1-\epsilon_2}{1+\epsilon_2} > (1-2\epsilon_2) \\ &= \Pr\left[0 \leq 2\epsilon_2\alpha - \epsilon\right] = 0 \qquad \text{since } \epsilon_2 < \epsilon/2\alpha\end{aligned}$$

Using Theorem 5,

$$\begin{aligned}\Pr[\bar{B}] &= \Pr[\hat{n}_d > (1+\epsilon_2)n_d] \\ &\leq \Pr[\hat{n}_d < (1-\epsilon_2)n_d] + \Pr[\hat{n}_d > (1+\epsilon_2)n_d] \\ &\leq \delta_2 \qquad \leq \frac{2\delta}{\log(1/\delta)}\end{aligned}$$

□

Theorem 6. *By running at least* $\frac{\log \delta}{\log(e^{-2}+c_\delta \delta)}$ *parallel instances of the above algorithm, where* $c_\delta \leq \frac{2}{\log(1/\delta)}$*, α-persistent items can be tracked with the following properties:*

1. *An item d with persistence $p(d) \geq (\alpha n)$ is reported as α-persistent with probability at least $1-\delta$.*
2. *An item d with persistence $p(d) < (\alpha - \epsilon)n$ is not reported as persistent with probability at least $1-\delta$.*

Proof. Suppose we run θ parallel instances of the above algorithm, and take the union of the items returned by all the instances. For the first part, consider an α-persistent item d. If d is not returned, it must not be returned by any of the instances. With respect to the time of arrival of a persistent item d, if a query q is posed on a window $[t_q - n + 1, t_q]$, then we have two cases: 1) If $t_q \leq t_d + n$, then d is reported with probability $(e^{-2} + c_\delta \delta)^\theta$, where $0 < c_\delta \leq \frac{2}{\log(1/\delta)}$. So, if we run $\frac{\log \delta}{\log(e^{-2}+c_\delta \delta)}$ parallel instances, the probability that false negative occurs is at most δ. 2) If $t_q > t_d + n$, then d is reported with probability $\left(\frac{2\delta}{\log(1/\delta)}\right)^\theta$, and the proof follows.

For an item d with persistence less than $(\alpha - \epsilon)n$, the proof is similar to the case of infinite window. □

3.2. Sliding Window: Complexity Analysis

Theorem 7. ***Expected Space:*** *Total expected space required by the sliding window algorithm per site is* $O\left(\frac{\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$ *and the expected space complexity over all sites is* $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$

Proof. Here, P is the sum of the persistence (total persistence till current time slot, c) of all the distinct items in the distributed stream during the period $[c-n+1, c]$. The proof is similar to that of Theorem 3. □

Theorem 8. ***Low Communication Overhead:*** *The expected communication complexity of the sliding window algorithm over all sites is* $O\left(\frac{k\alpha^2 \log\left(\frac{\log(1/\delta)}{2\delta}\right)P}{\epsilon^3 n}\right)$

Proof. Proof is similar to that of Theorem 4, except that the communication occurs between sites only when a new item is tracked and for the same item no further communication is done until the query is posed. □

4. Experiments

We report on the observed performance of our implementation of the infinite window and the sliding window algorithms.

Data. We have used synthetic as well as real-world data sets for our experiments. The real-world data is a *network traffic trace* from CAIDA [2] taken at a US west coast OC48 peering link for a large ISP in 2002 and 2003, where we consider each source-destination pair to be an item. The network trace has approximately 400 million tuples with about 26 million distinct items. The trace has been captured over a duration of 1 hour. We also generated synthetic data using a zipfian distribution, with $\alpha = 1.5$. This dataset has 500 million tuples, consisting of approximately 40 million distinct items.

Algorithms. We compared our algorithm with two other algorithms. The first one which we call algorithm A, is an exact distributed algorithm for tracking persistence, which identifies α-persistent items by keeping track of the exact persistence number of each item, through maintaining the distinct slots it occurred in. Algorithm A is the most expensive in terms of space and communication.

The second algorithm, which we call Algorithm B, is a small space algorithm which selectively tracks items using a hash-based sampler in the same way as our algorithm does. But for each item d that Algorithm B tracks, it computes the exact number of distinct time slots where the item reappears by maintaining a list of distinct time slots of appearance. Since Algorithm B does not incur an error cost in counting the number of re-occurrences once it starts tracking an item, it can actually track fewer items than our algorithm for the same value of ϵ. However, for each item tracked, B has to maintain significant amount of state for the item (a list of all slots where the item appeared, at each site).

Our experiments evaluate the performance of our algorithms in terms of communication cost and accuracy, where accuracy is measured through the false positive and false negative rates. We also performed experiments to show the effect of the width of time slot on the communication cost of the dataset. Unless specified otherwise, we set the error probability δ to e^{-2}. For the infinite window case, we divided the real world trace into 34 million non-overlapping time slots (width of each time slot being 0.1 millisecond) and the zipfian dataset into 36 million non-overlapping slots. To evaluate the sliding window version of the algorithm, we considered a window size of 30 million distinct time slots (width of each time slot being 0.1 millisecond) for real world trace and 25 million distinct time slots for zipfian dataset.

Communication Cost vs Accuracy. In the first set of experiments, we kept the number of sites constant, at 10, and varied the approximation parameter ϵ. The results from the experiments on zipfian data is shown

in Figure 1a for the infinite window case and in Figure 1b for the sliding window case. From this data we make the following observations.

There is a clear trade-off between accuracy and communication cost. The communication cost decreases as we increase the value of ϵ for our algorithm as well as for Algorithm B. The communication cost incurred by our algorithm for both infinite and sliding window is an order of magnitude smaller than that of Algorithms B and A. In fact, we observe that the communication cost of algorithm B is only slightly smaller than the naive algorithm A.

The results of experiments on the network trace are presented in Figures 1c and 1d. These are similar to the results for zipfian data, and our algorithm has significantly lower communication cost when compared with Algorithms A and B. However, since the size of the dataset is smaller and it has a relatively small number of distinct time slots, the cost incurred by algorithm B is low. Hence, in this case the communication cost of algorithm B, though higher than our algorithm, is not as high as in the case of zipfian data file. This shows that the benefits of our algorithm are even greater on large datasets with a large number of time slots.

Communication Cost vs Number of Sites. In order to evaluate the scalability with the size of the distributed system, we varied the number of sites in the system, while keeping the approximation error ϵ fixed at 0.025. The results for zipfian data are shown in Figures 2a and 2b, and for the network trace in Figures 2c and 2d. We observe that the communication cost of the algorithm increases linearly with the number of sites in the system, in accordance with the theoretical results. The results also show that our algorithm consistently performs better than the other two algorithms. Algorithms A and B also show a similar linear increase in communication cost with the number of sites.

The increase in communication cost of our algorithms and that of algorithm B is due to the fact that every site has a copy of each item tracked in the distributed system. Hence, increase in the number of sites would lead to an almost linear increase of the space requirement and communication cost. In the case of the algorithm A, the reason for the increase in the communication cost is as follows: multiple appearance of an item in the same time slot does not affect the size of the datastructure maintained by the site, but if an item appears multiple times in the same slot across different sites, then multiple copies (same as the number of sites where they appear) of the items need to be maintained, increasing the communication cost.

Communication Cost vs Width of Timeslot. We performed experiments to study the effect of the width of time slots for a given dataset on communication cost. We keep the value of ϵ fixed at .025 and the number of sites fixed at 10. We use real network trace for our experiments. Using the same dataset traces, we vary the width of each time slot from 0.1 millisecond to 2 milliseconds and measure the communication cost of

our algorithm.

The results are shown in Figure 3a for infinite window and Figure 3b for sliding window. We observe that the communication cost of algorithm A decreases slightly with the increase in the width of time slot, assuming that the threshold for persistence is kept fixed. The reason is that the distinct number of time slots decreases with the increase in the width of time slot, hence, algorithm A has to maintain a smaller datastructure. However, interestingly, the communication costs of our algorithm and Algorithm B increase with the width of time slot. This is due to the fact that as the width of the time slot increases, the number of persistent items for a given persistence threshold increases, and the data structures become larger. Though the number of messages decreases, the size of the messages increases, leading to an overall increased communication cost.

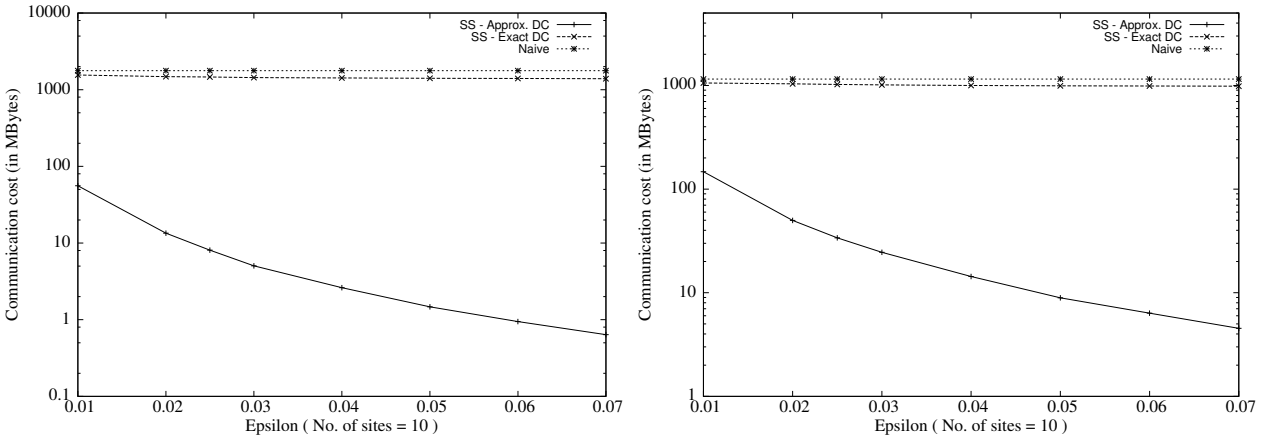
We have also included a graph showing the change in the total number of elements tracked across the distributed system when the width of the time slot is varied. The results are shown in figure 4a for infinite window and in figure 4b for sliding window. The number of elements tracked does not vary for algorithm A as it tracks all the elements in the distributed dataset. However, for our algorithm and for algorithm B, the number of elements tracked increases with the increase in the width of time slot.

We also compared the space cost, which is defined as the total space taken by the data structures at the sites and the coordinator. In general, the space taken by our algorithm is much smaller than that of Algorithms A and B, for most parameter settings. In Table 1, we show the space cost of each algorithm for the sliding windows scenario, on the zipfian data on a distributed system of 10 nodes, for different values of ϵ. The space cost of A is constant, since it is unaffected by the setting of ϵ, while that of B is rather large due to the need to maintain the exact set of distinct time slots where the tracked elements appeared. The results for the other data sets, and for the infinite windows version are similar.

Table 1: Space cost for Zipfian data on system of 10 nodes for algorithms A, B and our algorithm (SS) for sliding windows

Epsilon	Space taken by SS (in MBytes)	Space taken by B (in MBytes)	Space taken by A (in MBytes)
.01	149.129	1089.92	1860.38
.02	50.3261	1068.78	1860.38
.03	24.7846	1045.55	1860.38
.04	14.4616	1032.57	1860.38
.05	9.01553	1024.79	1860.38
.06	6.40722	1021.77	1860.38
.07	4.57717	1017.46	1860.38

Accuracy. We measure the actual false negative rate and the false positive rate of our algorithm for different values of δ, using 10 sites, keeping the value of ϵ fixed at 0.025. We use real network trace and zipfian dataset



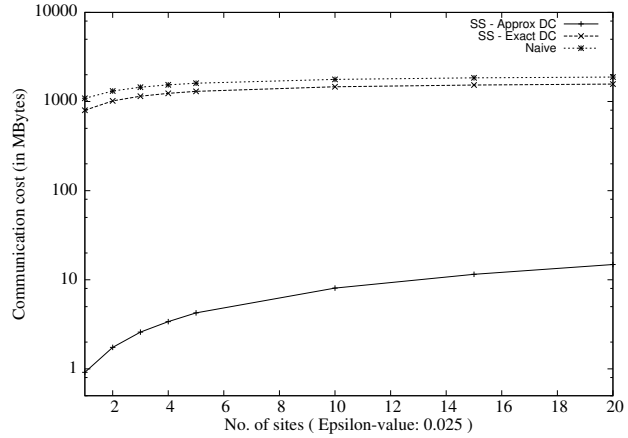
(a) Zipfian data $\alpha = 1.5$ (Infinite Window)

(b) Zipfian data $\alpha = 1.5$ (Sliding Window - 25 million time slots)

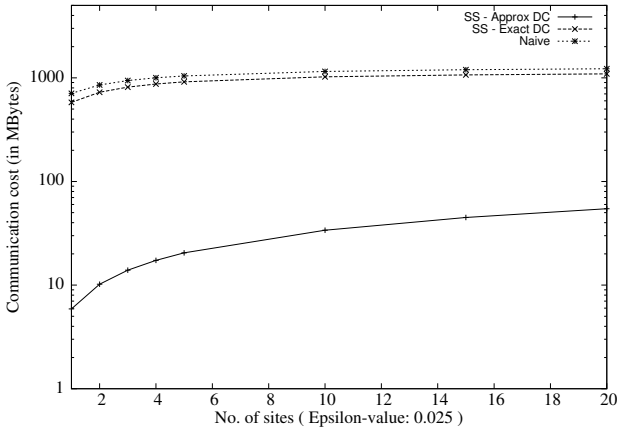
(c) Network Trace (Infinite Window)

(d) Network Trace (Sliding Window - 30 million time slots)

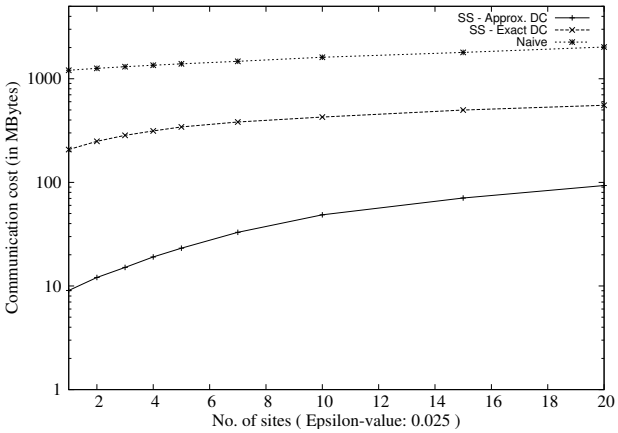
Figure 1: Communication Overhead for varying relative error 'ϵ', keeping number of sites fixed at 10



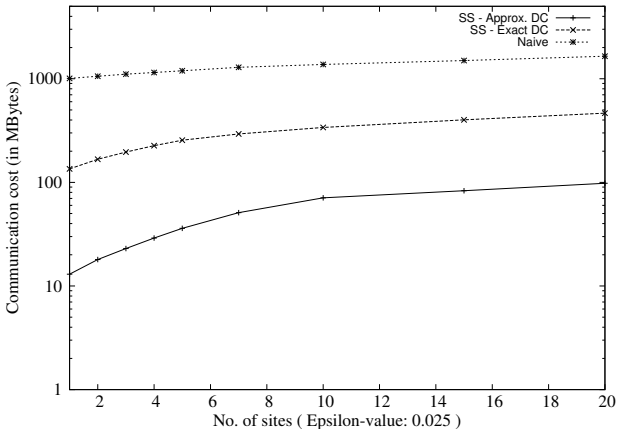
(a) Zipfian $\alpha = 1.5$ (Infinite Window)



(b) Zipfian $\alpha = 1.5$ (Sliding Window - 25 million time slots)

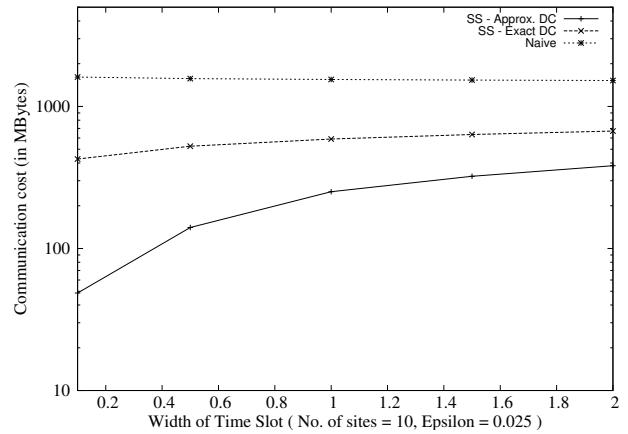


(c) Network Trace (Infinite Window)

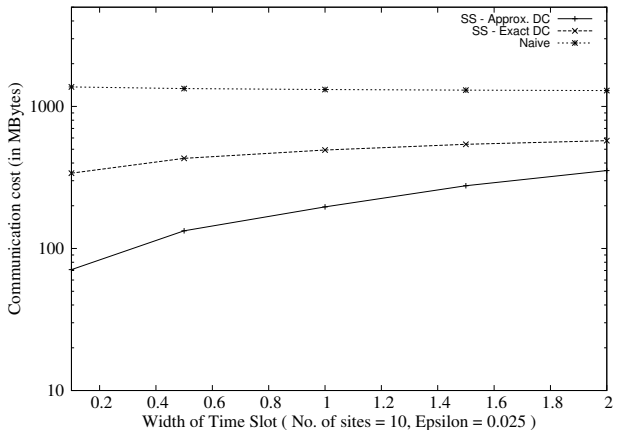


(d) Network Trace (Sliding Window - 30 million time slots)

Figure 2: Communication Overhead for varying number of sites keeping ϵ fixed at 0.025

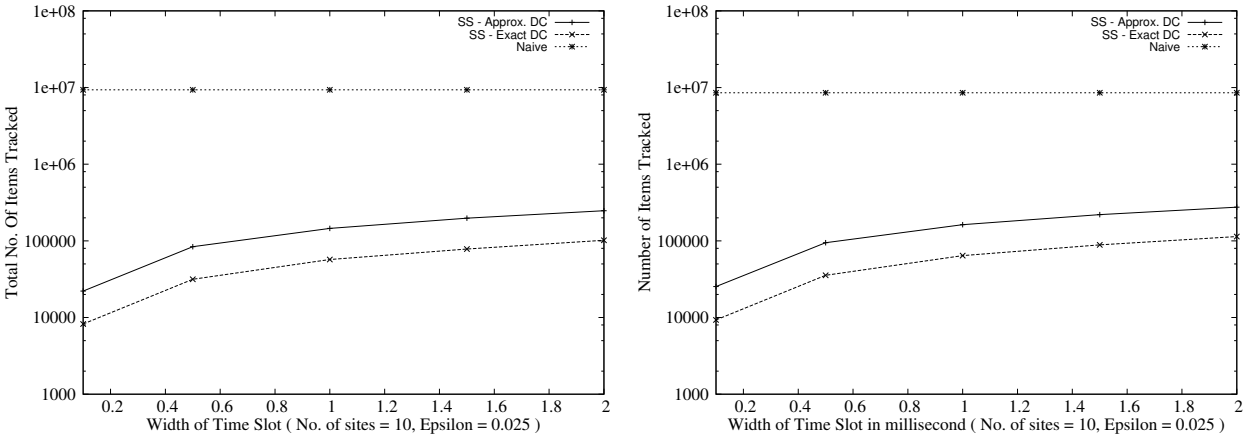


(a) Network Trace (Infinite Window)



(b) Network Trace (50 minutes Sliding Window)

Figure 3: Communication Overhead as a function of the width of time slot (in milliseconds)



(a) Network Trace (Infinite Window)

(b) Network Trace (50 minutes Sliding Window)

Figure 4: Total number of items tracked across all sites as a function of the width of the time slot (in milliseconds)

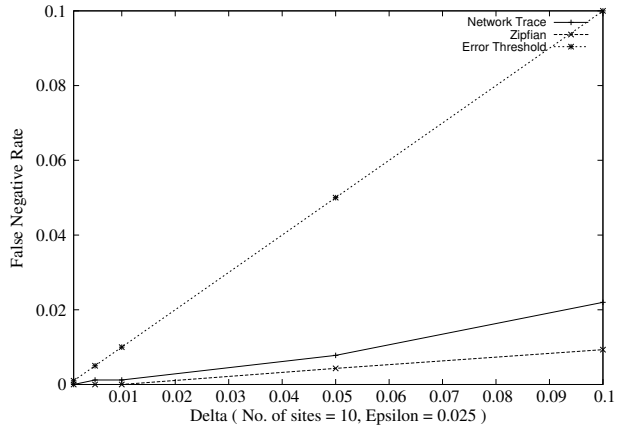
for our experiments. For the experiments corresponding to the sliding window version of our algorithm, we consider the window size of 30 million time slots for the network trace and 25 million time slots for zipfian.

According to our paper, we report an item d as persistent if its approximate persistence $\hat{p}_d$ is at least αn. Also, an item d is not reported as persistent if its estimated persistence $\hat{p}_d$ is less than $(\alpha n - \epsilon n)$. Note that for the infinite window version of the algorithm, n denotes the total number of time slots in the entire dataset, and for the sliding window version of the algorithm, it denotes the maximum number of time slots in the current window.

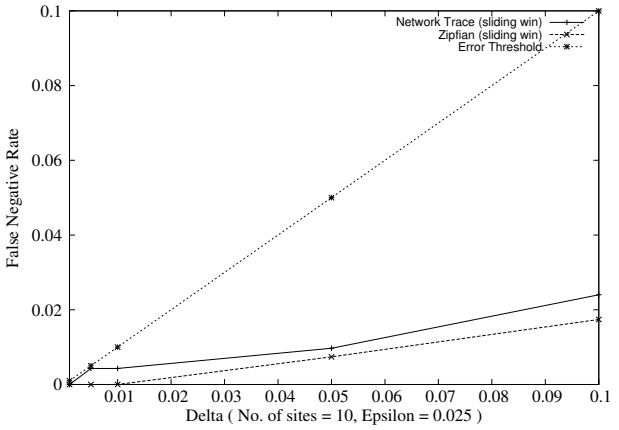
We define the false negative rate as a fraction of the persistent items which were not reported as persistent, and the false positive rate as a fraction of the non-persistent items which were reported as persistent. Per Theorem 2, the false negative rate and the false positive rate given by our algorithm is bounded by error probability δ.

The false negative rate for the zipfian and the network trace is shown in Figure 5a for infinite window and Figure 5b for sliding window. For this experiment, we vary δ from 0.001 to 0.1. The plot named "Error Threshold" plots the maximum expected error for each value of δ. We observe that the false negative rate for both datasets is always less than the error probability δ. In fact, for our experiments, the false negative rate of network trace and zipfian did not exceed 0.025 for any value of δ. We also observe that for $\delta = 0.001$, there are no false negatives.

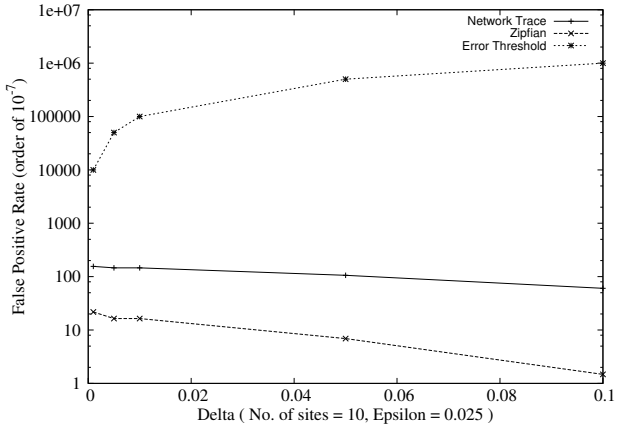
Similarly, we observe that the the false positive rates for the zipfian and the network trace, shown in Figure 5c for infinite window and in Figure 5d for sliding window, is much below the error threshold. The false positive rate of network trace and zipfian given by both infinite window and sliding window version of



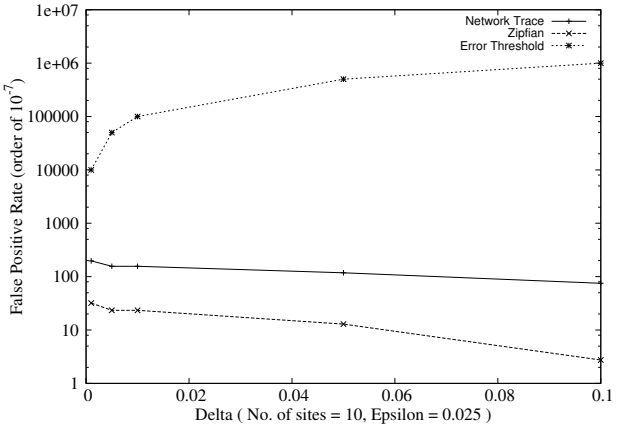
(a) False Negative Rate - Infinite Window



(b) False Negative Rate - Sliding Window (30 million time slots)



(c) False Positive Rate - Infinite Window



(d) False Positive Rate - Sliding Window (30 million time slots)

Figure 5: False Negative Rate and False Positive Rate as a function of δ for Network Trace and Zipfian

our algorithm, are in the order of 10^{-5} to 10^{-7}.

5. Conclusion

We presented algorithms for communication-efficient monitoring of persistent items in a distributed stream of events. These can help detect situations such as when a malicious adversary is establishing a regularly spaced connection to a remote entity, but is trying to evade detection through keeping the volume of communication low and by having the communication originate from different sites. The total distributed state maintained by our algorithms is far less than the number of distinct items observed in the stream, and the communication overhead is also small compared with the number of events and the number of items observed. Our experimental evaluations show that the communication cost and memory cost of our algorithms are much smaller than those of straightforward algorithms, and their false positive and false negative rates

are typically much lower than theoretical predictions.

Acknowledgments

This work was funded in part by the National Science Foundation through grants 0834743 and 0831903 and through a fellowship from IBM. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US National Science Foundation or the IBM Corporation.

References

[1] Pay per click - wikipedia.
`http://en.wikipedia.org/wiki/Pay_per_click`.

[2] The CAIDA UCSD Anonymized OC48 Internet Traces 2002-2003.
`https://data.caida.org/datasets/oc48/oc48-original/`.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th annual ACM symposium on Theory of computing (STOC)*, pages 20–29, 1996.

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*, pages 1–10, 2002.

[5] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 147–156, 2009.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 668–668, 2003.

[7] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 268–279, 2000.

[8] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 693–703, 2002.

[9] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, Aug. 2008.

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

[11] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 77–86, 2010.

[12] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, pages 57–, 2006.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows: (extended abstract). In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 635–644, 2002.

[14] D. Eyers, T. Freudenreich, A. Margara, S. Frischbier, P. Pietzuch, and P. Eugster. Living in the present: On-the-fly information processing in scalable web architectures. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms (CloudCP)*, pages 6:1–6:6, 2012.

[15] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, Sept. 1985.

[16] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 13–24, 2001.

[17] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the 13th annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 281–291, 2001.

[18] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proceedings of the 14th annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 63–72, 2002.

[19] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 326–345, 2009.

[20] L. Golab, D. DeHaan, E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC)*, pages 173–178, 2003.

[21] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB)*, pages 311–322, 1995.

[22] R. Kannan, S. Vempala, and D. P. Woodruff. Principal component analysis and higher correlations for distributed data. In *Proceedings of the 27th Annual Conference on Learning Theory (COLT)*, pages 1040–1057, 2014.

[23] J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of 6th International Workshop on Networking Meets Databases (NetDB)*, 2011.

[24] B. Lahiri, J. Chandrashekar, and S. Tirthapura. Space-efficient tracking of persistent items in a massive data stream. In *Proceedings of the 5th ACM international conference on Distributed event-based system (DEBS)*, pages 255–266, 2011.

[25] B. Lahiri and S. Tirthapura. Finding correlated heavy-hitters over data streams. In *Proceedings of the IEEE 28th International Performance Computing and Communications Conference (IPCCC)*, pages 307–314, 2009.

[26] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 767–778, 2005.

[27] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases (VLDB)*, pages 346–357, 2002.

[28] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th international conference on Database Theory (ICDT)*, pages 398–412, 2005.

[29] J. Misra and D. Gries. Finding repeated elements. Technical report, Ithaca, NY, USA, 1982.

[30] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proceedings of the VLDB Endowment*, 5(10):992–1003, June 2012.

[31] K. Patroumpas and T. Sellis. Window specification over data streams. In *Proceedings of the 2006 international conference on Current Trends in Database Technology (EDBT)*, pages 445–464, 2006.

[32] A. Rabkin and R. Katz. Chukwa: A system for reliable large-scale log collection. In *Proceedings of the 24th International Conference on Large Installation System Administration*, pages 1–15, 2010.

[33] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*, pages 239–249, 2004.

[34] S. Tirthapura and D. P. Woodruff. Optimal random sampling from distributed streams revisited. In *Proceedings of the 25th International Conference on Distributed Computing (DISC)*, pages 283–297, 2011.

[35] S. Tirthapura and D. P. Woodruff. A general method for estimating correlated aggregates over a data stream. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE)*, pages 162–173, 2012.

[36] S. Tirthapura, B. Xu, and C. Busch. Sketching asynchronous streams over a sliding window. pages 82–91, 2006.

[37] D. P. Woodruff and Q. Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 941–960, 2012.

[38] D. P. Woodruff and Q. Zhang. When distributed computation is communication expensive. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*, pages 16–30, 2013.

[39] L. Zhang and Y. Guan. Detecting click fraud in pay-per-click streams of online advertising networks. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems (ICDCS)*, pages 77–84, 2008.

[40] W. Zhang, Y. Zhang, M. A. Cheema, and X. Lin. Counting distinct objects over sliding windows. In *Proceedings of the 21st Australasian Conference on Database Technologies - Volume 104*, pages 75–84, 2010.