Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams

T-H. Hubert Chan, Mingfei Li, Elaine Shi*, and Wenchang Xu

Abstract. We consider applications scenarios where an untrusted aggregator wishes to continually monitor the heavy-hitters across a set of distributed streams. Since each stream can contain sensitive data, such as the purchase history of customers, we wish to guarantee the privacy of each stream, while allowing the untrusted aggregator to accurately detect the heavy hitters and their approximate frequencies. Our protocols are scalable in settings where the volume of streaming data is large, since we guarantee low memory usage and processing overhead by each data source, and low communication overhead between the data sources and the aggregator.

1 Introduction

Consider k data streams at k data sources, where items from some set $\mathcal{U}$ arrive at each stream. An *untrusted* aggregator wishes to continually monitor the most recent *heavy hitters* (i.e. the frequent items) over a sliding window – however, the data sources do not trust the aggregator, and wish to guarantee the privacy of their data streams. For example, a public health provider would like to monitor the potential outbursts of new epidemics (where the heavy hitters are the most common symptoms or diseases) by studying hospital visit records from k hospitals. Since medical records contain highly sensitive information, the hospitals may be legally obliged to protect their patients' privacy from the third-party public health provider. In Figure 1, we show another example where each stream represents a store, and the aggregator wishes to track the most popular items in the past week.

1.1 Results and Contributions

In this paper, we propose novel protocols that allow an untrusted aggregator to continually monitor the heavy hitters over a sliding window of duration W, while protecting the privacy of each data source. Since the aggregator is untrusted and there is no single trusted entity, standard privacy frameworks like PINQ [16] cannot be used directly in our distributed setting. In our protocols, each data source periodically sends sanitized (and potentially also encrypted) updates to the aggregator in order to notify the aggregator of latest trends as observed by the data source. The aggregator is then able

* This material is based upon work partially supported by the Air Force Office of Scientific Research under MURI Grant No. 22178970-4170 and No. FA9550-08-1-0352. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

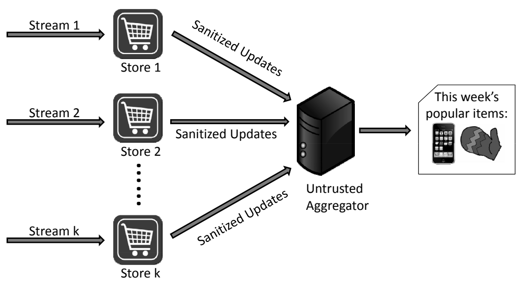

Fig. 1. Problem setup. In this example, k stores wishes to continually monitor the popular items over the past week. The aggregator is assumed to be untrusted; and the stores may be concerned about protecting their secret business information such as sales revenue, and protecting the privacy of their customers.

to reconstruct the most recent popular items and their respective frequencies through these sanitized updates.

We conduct experiments using the Netflix Contest Dataset, and demonstrate that our algorithms can achieve low communication bandwidth and good utility in realistic application scenarios. We next explain the privacy guarantees and desirable features that our constructions achieve.

Two Levels of Privacy Protection. We propose protocols that achieve the following two different aspects of privacy.

Event-level differential privacy. Our first construction, referred to as the PDCH-LU algorithm (which stands for *Private Distributed Continual Heavy-hitter - Lazy Update*), achieves *event-level differential privacy*. Roughly speaking, the sanitized updates released should be insensitive to the occurrence or non-occurrence of a single event. Intuitively, event-level differential privacy allows a store to guard the privacy of its customers, by concealing whether a certain purchase has taken place.

In our constructions, we achieve event-level differential privacy through the addition of appropriate noises before the release of any statistics. Note also that ϵ event-level differential privacy immediately implies $m\epsilon$ user-level differential privacy, where m is the maximum number of items for each user.

Aggregator obliviousness. Through the use of bloom filters and special encryption schemes, our second protocol, referred to as PDCH-BF (which stands for *Private Distributed Continual Heavy-hitter - Bloom Filter*), achieves even stronger privacy guarantees: specifically, it achieves *aggregator obliviousness* in addition to *event-level differential privacy*.

On a high level, aggregator obliviousness advocates the *need-to-know* principle, i.e., the aggregator should ideally learn the *least amount of information* necessary to perform the heavy-hitter monitoring task. Specifically, in our second construction PDCH-BF, apart from the approximate frequencies of a subset of the relatively more popular items across all streams, the aggregator learns nothing else.

To achieve aggregator obliviousness, our second protocol PDCH-BF employs bloom filters, as well as special encryption schemes [14, 19, 20] which support secure aggregation and controlled decryption of selective statistics.

Notice that aggregator obliviousness immediately implies the following: 1) in the example in Figure 1 the aggregator can learn the approximate frequencies of (a subset of) the items, but it cannot learn the transaction volume of each individual store which may be considered secret business information; and 2) although the aggregator can learn which the heavy hitters are and their approximate frequencies across all streams, the aggregator fails to learn which streams are contributing to these heavy-hitters, and how much each stream is contributing to these heavy hitters.

A more detailed discussion of our privacy notions and their nuances can be found in Section 2.3.

Low Computational and Communication Overhead. Our protocols require only a small amount of computation and memory from each data source. To process an item from the stream, a data source needs to update only a small number of counters; and in each time step, it needs to sample only a small number of random variables. This is desirable in numerous application settings – for example, in sensor network applications, where each node has low computational resources; or network intrusion detection scenarios, where routers cannot afford expensive real-time computation due to the large bandwidth throughput.

Our protocols also requires low communication costs between the data sources and the aggregator. Moreover, all communications are uni-directional from the data sources to the aggregator, and the data sources need not have interactions among themselves. In contrast, generic secure multi-party computation construction [12] requires expensive interactive communication between the data sources.

A Paradigm to Privatize Streaming Algorithms. Our approach also represents a new paradigm for privatize streaming algorithms. Our techniques for privatizing streaming algorithms is applicable to a broad class of streaming algorithms which fall within the following 3-phase framework, i.e., throughout the execution of such a streaming algorithm, objects are created and they go through three phases: (1) *Construction Phase*. The object is still being modified according to incoming items from the stream, and at this point the object is not used to produce any output. (2) *Active Phase*. The object is no longer modified and can be used to produce output in the time steps for which the object still exists. (3) *Expiration*. The object is deleted so that space can be reused.

The key idea to differentially privatize such streaming algorithms is that as soon as an object becomes active, random noise is applied to ensure its differential privacy before it is used to produce any output. In fact, the continual counting mechanism for bit streams by Chan *et al.* [4] follows this design paradigm, where an object corresponds to a counter recording the number of 1's appearing in some time interval. However, not all objects can be readily privatized. In the randomized version of the algorithm by Arasu and Manku [1] and also the deterministic algorithm by Lee and Ting [15], an object contains information about the position of some item. It is not clear how to privatize such an object while keeping its usefulness. Fortunately, the counters in the algorithm by Misra and Gries [18] can be privatized.

1.2 Related Work

Our work builds on several well-known lines of research. We describe some of the works that are the most related to ours and refer the readers to the cited references for more extensive review on the relevant research areas.

Differential Privacy in Continual Setting. Ever since Dwork [6] has introduced differential privacy, this notion has gained popularity in both the theory and security communities (see [7] for a quick review of the latest development). The idea of introducing randomness to perturb the outputs of algorithms allows a clean and formal way to analyze the tradeoff between preserving input privacy and achieving output accuracy. Recently, privacy has been studied in the continual setting [4, 8, 9]. Specifically, a change in the input in the current time step would not only affect the output in the current time step, but also might have a long lasting effect in the future. Useful continual differentially private algorithms would need to mask this long term effect without sacrificing too much on accuracy. Chan *et al.* [4] gave a differentially private mechanism to continually report the number of 1's seen so far in a bit stream with additive error that is polylogarithmic in the number of time steps. In the streaming model, Dwork *et al.* [9] also distinguish between *event-level privacy* and *user-level privacy*: event-level privacy hides the occurrence of a particular event, while user-level privacy prevents adversaries from determining whether the stream contains any of a particular user's activities at all. In this paper, we use event-level differential privacy as our privacy notion. Mir *et al.* [17] also considered the problem of private streaming algorithms to return the counts of heavy hitters, not the heavy hitters themselves. However, they consider a more general setting in which in each update, the counter of an item can be increased or decreased arbitrarily, as long as the counter remains non-negative. Moreover, their notion of privacy hides the following change in the stream: any subset of occurrences of an item can be replaced with another item, and remaining updates can be arbitrarily reordered; on the other hand, the total count of all items is public knowledge in their setting.

Untrusted Aggregator. There have been works on studying the case when the aggregator is untrusted [14, 19, 20], where cryptographic techniques are employed. We first consider protocols in which each node will desensitize its data first so that cryptography will not be necessary; in order to achieve a stronger notion of privacy and security, we augment our protocols by employing cryptographic techniques.

Streaming Algorithms for Heavy Hitters. The first algorithm to output frequent items was given by Misra and Gries [18] (MG algorithm). They designed a deterministic algorithm that reads a stream of W items and at the end gives the count of every item in the stream with relative error λ (i.e., additive error at most λW); the algorithm only uses $O(\frac{1}{\lambda})$ words of memory. The MG algorithm was rediscovered several times [5, 13].

Using the MG algorithm concurrently on overlapping blocks of different sizes, Arasu and Manku [1] gave a deterministic algorithm that continually estimate the count of every item with relative error λ with respect to the current window; the query and the update time is $O(\frac{1}{\lambda}\log\frac{1}{\lambda})$, while $O(\frac{1}{\lambda}\log^2\frac{1}{\lambda})$ words of memory is required. They also gave a randomized version, where both the time and the memory is $O(\frac{1}{\lambda}\log\frac{1}{\delta\lambda})$, where δ is the failure probability.

Lee and Ting [15] augmented the counters in the MG algorithm to include approximate positions of where items occur, and consequently they improved Arasu and Manku's algorithm performance to $O(\frac{1}{\lambda})$, for both running time and memory requirement.

Distributed Streaming Protocols with Low Communication Cost. In the distributed streaming model, each of k nodes receives its own stream, and the nodes communicate with the aggregator, who wishes to estimate the number of times each item appears in all the streams in the current window with relative error λ. Yi and Zhang [21] considered the special case with infinite window size and gave a 2-way communication (between nodes and aggregator) protocol with total communication cost of $O(\frac{k}{\lambda}\log N)$ words, where N is the total number of items arriving at all streams. Chan *et al.* [3] considered the case of a sliding window, and gave a one-way communication (from nodes to aggregator) protocol. Under the special case of exactly one item arriving in each time step for each stream, the communication cost for their protocol in L consecutive time steps is $O(\frac{k}{\lambda}\cdot\lceil\frac{L}{W}\rceil\log W)$ words.

Related Notion of Privacy. In Gantal *et al.* [10], it is mentioned that Dwork and McSherry proposed *semantic privacy* which measures how the posterior distribution on the database changes after the transcript is observed. In particular, it is shown that ϵ-differential privacy implies $(e^{\epsilon}-1)$-semantic privacy.

2 Preliminaries

2.1 Notations and Conventions

Given a positive integer m, we let $[m] := \{1, 2, \ldots, m\}$, and use $\mathbb{N} := \{1, 2, 3, \ldots\}$ to index time steps. Let $\mathcal{U}$ be a set of n items. We use the standard notation $\tilde{O}(\cdot)$ to suppress poly-logarithmic factors.

We assume that an integer can be represented by $O(1)$ words. Although later on we use random distribution on unbounded integers, we show that with high probability the magnitudes of the sampled integers are small. Hence, we can use modulo arithmetic over some large enough integer. We do not explicit explain how each data source obtains its source of randomness, but we assume that it takes $O(1)$ operations to sample a random variable that is independent of the input data stream.

2.2 Problem Setup

We assume a set of $k \in \mathbb{N}$ *streams*, originating at k *data sources* (also referred to as *nodes*) respectively. We assume that each data source only has limited memory and computational power. Each stream $\sigma^{(i)} \in \mathcal{U}^{\mathbb{N}}$ where $i \in [k]$ is a sequence of items from $\mathcal{U}$, where $\sigma^{(i)}(t) \in \mathcal{U}$ is the item appearing at time step t in the i-th stream.

We consider an *untrusted aggregator* who wishes to continually monitor the heavy hitters over a sliding window of size $W \in \mathbb{Z}$. Formally, the window at time step t over stream σ is the multiset $\mathcal{W}_t(\sigma) := \sigma([t-W+1, t])$ containing all items coming to stream σ between time $t - W + 1$ and t. Given k streams $\{\sigma^{(i)} : i \in [k]\}$, we write $\mathcal{W}_t^{(i)} := \mathcal{W}_t(\sigma^{(i)})$ and denote $\mathcal{W}_t^{[k]} := \uplus_{i\in[k]}\mathcal{W}_t(\sigma^{(i)})$ as the multiset containing all items from the k streams in the window at time t.

The notion of *heavy hitters* is formally defined as below. Given a multiset $\mathcal{W}$, we use $|\mathcal{W}|$ to denote the number of items it contains and for $x \in \mathcal{U}$, $\mathsf{count}_x(\mathcal{W})$ is the number of times item x appears in $\mathcal{W}$. Given $0 < \theta < 1$, we say an item $x \in \mathcal{U}$ is a θ-heavy hitter in the multiset $\mathcal{W}$ if $\mathsf{count}_x(\mathcal{W}) \geq \theta \cdot |\mathcal{W}|$.

Definition 1 (Approximate Heavy Hitters). *Given $0 < \lambda, \theta < 1$ and a multiset $\mathcal{W}$, a set $S \subseteq \mathcal{U}$ is a λ-approximation for θ-heavy hitters in $\mathcal{W}$ if*
1. *the set S contains all θ-heavy hitters in $\mathcal{W}$; and*
2. *if $x \in S$, then x is a $(\theta - \lambda)$-heavy hitter in $\mathcal{W}$.*

Definition 2 (λ-approximate Count). *Given a multi-set $\mathcal{W}$ on items in $\mathcal{U}$, and an item $x \in \mathcal{U}$, an estimate $\widehat{c}(x)$ is called a λ-approximate count for x with respect to $\mathcal{W}$, if $|\widehat{c}(x) - \mathsf{count}_x(\mathcal{W})| \leq \lambda|\mathcal{W}|$.*

Observe that if we have a λ-approximate count for every item $x \in \mathcal{U}$ with respect to $\mathcal{W}$, then we can compute a 2λ-approximation for heavy hitters in $\mathcal{W}$.

Communication Protocol. Consider a node receiving some stream σ. At every time step t, upon receiving the item $\sigma(t)$, the node might send messages to the aggregator to update some counters. In the protocols that we consider, each message contains items of the form $c \in \mathbb{Z}$ or $\langle x, c\rangle \in \mathcal{U} \times \mathbb{Z}$, each of which we assume can be expressed in $O(1)$ words. Given a (randomized) protocol Π, we denote by $\Pi(\sigma)$ the (randomized) transcript which consists of the messages sent at every time step by the node that applies the protocol on the stream σ. We wish to reduce the amount of communication, say the average number of words sent per time step.

Remark 1. In order to show that each item count has small relative error, we need a lower bound on the total number items in the finite stream to absorb the noise error. Moreover, the way in which we use PMG in combination with Arasu and Manku's algorithm [1] for fix-sized windows requires the assumption that exactly one item comes in the stream at every time step.

2.3 Defining Privacy

As mentioned earlier, we define our privacy notions based on the following principles: 1) We advocate a need-to-know principle, the aggregator should ideally learn the least amount of information necessary to perform the heavy hitter monitoring task. 2) While the amount of information revealed to the aggregator is kept at a minimum, the information eventually revealed to the aggregator should satisfy event-level differential privacy. In other words, any statistics revealed should be insentive to the occurrence or non-occurrence of a single event. Intuitively, this helps to conceal whether some event of interest has happened, e.g., whether a customer Alice has purchased a specific item.

Below, we formally define differential privacy and aggregator obliviousness.

Differential Privacy. In our setting, each node regards the contents on its stream as private data. In particular, from the transcript of a node, the aggregator should not be able to distinguish between input streams that are close to each other. Formally, two different streams $\sigma_1, \sigma_2 \in \mathcal{U}^{\mathbb{N}}$ are *adjacent* or *neighbors* (denoted as $\sigma_1 \sim \sigma_2$) if they differ at exactly one time step. We use the notion of event-level differential privacy for protocols.

Definition 3 (Differential Privacy for Protocols). *Given $\epsilon > 0$, a (randomized) protocol Π is ϵ-differentially private if for any adjacent streams σ_1 and σ_2, any subset $\mathcal{O}$ of possible output transcripts,* $\Pr[\Pi(\sigma_1) \in \mathcal{O}] \leq \exp(\epsilon) \cdot \Pr[\Pi(\sigma_2) \in \mathcal{O}]$, *where the randomness comes from the protocol.*

Aggregator Obliviousness. As we shall see, as an intermediate step in our protocols, each node has some private number and the goal is for the aggregator to learn the sum of all the nodes' numbers, but nothing more. Formally, each node has some data in $\mathcal{D}$ and we use $\mathbf{x} \in \mathcal{D}^n$ to denote a configuration of all the nodes' data. Intuitively, a protocol Π is *aggregator oblivious* with respect to some function $f : \mathcal{D}^n \to \mathcal{O}$ if for all $\mathbf{x}$ and $\mathbf{y}$ such that $f(\mathbf{x}) = f(\mathbf{y})$, no polynomial-time adversary can distinguish between the transcripts $\Pi(\mathbf{x})$ and $\Pi(\mathbf{x})$ with non-negligible probability.

Definition 4 (Aggregator Obliviousness). *Let $\kappa \in \mathbb{N}$ be a security parameter. A protocol ensemble $\{\Pi_\kappa\}_{\kappa \in \mathbb{N}}$ is aggregator obliviousness with respect to the function $f : \mathcal{D}^n \to \mathcal{O}$ if there exists a negligible function $\eta : \mathbb{N} \to \mathbb{R}^+$ such that for all $\mathbf{x}$ and $\mathbf{y}$ such that $f(\mathbf{x}) = f(\mathbf{y})$, for all decisional probabilistic polynomial-time Turing machines $\mathcal{A}$,*
$|\Pr[\mathcal{A}(\Pi_\kappa(\mathbf{x})) = 1] - \Pr[\mathcal{A}(\Pi_\kappa(\mathbf{y})) = 1]| \leq \eta(\kappa)$,
where the probability is over the randomness of the protocol Π_κ and the Turing machine $\mathcal{A}$.

2.4 Defining Utility

Recall that for each $i \in [k]$, each node i receives some stream $\sigma^{(i)}$ and follows some (randomized) protocol to send messages to the aggregator in every time step. Based on the messages received up to time t, the aggregator computes for each $x \in \mathcal{U}$ some number $\mathcal{A}(t, x)$, which is an estimate for $\mathsf{count}_x(\mathcal{W}_t^{[k]})$. Observe that $\mathcal{A}(t, x)$ is a random variable, whose randomness comes from the randomized protocols. We use the following notion to measure the usefulness of $\mathcal{A}(t, \cdot)$ with respect to $\mathcal{W}_t^{[k]}$ for each t.

Definition 5 ((ξ, δ)-Usefulness). *Suppose $\mathcal{W}$ is a multiset containing items in $\mathcal{U}$, and $\mathcal{A} \in \mathbb{R}^{\mathcal{U}}$ is a collection of random variables indexed by $\mathcal{U}$. Then, $\mathcal{A}$ is (ξ, δ)-useful with respect to $\mathcal{W}$, if with probability at least $1 - \delta$, for every item $x \in \mathcal{U}$,*
$|\mathcal{A}(x) - \mathsf{count}_x(\mathcal{W})| \leq \xi$*; in particular, if $\xi = \lambda|\mathcal{W}|$, then $\mathcal{A}(x)$ is a λ-approximate count for x with respect to $\mathcal{W}$.*

Definition 6 ((ξ, δ)-Simultaneous Usefulness.). *Let $\mathcal{T}$ be an index set. Suppose for any $t \in \mathcal{T}$, $\mathcal{W}_t$ is a multiset containing items in $\mathcal{U}$, and $\mathcal{A}_t$ is a collection of random variables indexed by $\mathcal{U}$. Then, $\{\mathcal{A}_t\}_{t \in \mathcal{T}}$ is (simultaneously) (ξ, δ)-useful with respect to $\{\mathcal{W}_t\}_{t \in \mathcal{T}}$, if with probability at least $1 - \delta$, for every $t \in \mathcal{T}$ and every item $x \in \mathcal{U}$,*
$|\mathcal{A}_t(x) - \mathsf{count}_x(\mathcal{W}_t)| \leq \xi$.

3 Achieving Differential Privacy

3.1 Roadmap

This section describes a protocol between the data sources and an aggregator, allowing the aggregator to continually monitor the heavy hitters over a sliding window. We will show how to achieve event-level differential privacy in this section. Later in Section 4, we show how to achieve aggregator obliviousness.

We proceed with the following three-step recipe:

1. **The PMG algorithm outputs the heavy hitters in a single stream.** In Section 3.2, we describe a private streaming algorithm, which allows a single data source to output the approximate heavy hitters in a stream, after a one-pass scan of the entire stream. Since this algorithm builds on top of the Misra-Gries streaming algorithm [18], we refer to it as the Private Misra-Gries (PMG) algorithm. The MG Algorithm maintains an approximate vector of item counts by storing only non-zero counts explicitly for only a small number of items. The main technical challenge to privatize the MG Algorithm is to show that this approximate vector has small sensitivity.
2. **The PCC algorithm continually monitors the recent heavy hitters in a single stream.** In Section 3.3, we extend the aforementioned PMG algorithm to derive a Private Continual Heavy-hitter (PCC) algorithm, which supports the continual monitoring of heavy hitters over a sliding window in a single stream. The main technique in this step is the use of a *binary interval tree* which allows us to achieve small memory when the window size is large.
3. **The PDCH-LU protocol continually monitors the recent heavy hitters across multiple streams.** Finally, in Section 3.4, we extend the above PCC algorithm, which works for a single stream, to the distributed setting. In the resulting protocol PDCH-LU (*Private Distributed Continual Heavy-hitter - Lazy Update*), in order to save communication cost, each data source sends sanitized updates to the aggregator whenever necessary (hence lazy updates), to inform the aggregator of latest trends in its observed stream. The aggregator can in turn continually output the approximate heavy hitters across all streams over a sliding window.

3.2 Private Misra-Gries Algorithm

In this section, we consider a sub-problem, which will be a useful building block to construct the private streaming protocol at a node. Given a stream of length T and an error parameter $0 < \lambda < 1$, the goal is to estimate the number of times each item in $\mathcal{U}$ appears in the stream with additive error λT.

Our approach is based on the (non-private) MG Algorithm [18], which keeps explicit counters for only $O(\frac{1}{\lambda})$ items. Observe that since we accept λT additive error, if an item $x \in \mathcal{U}$ appears for less than λT times in the stream, then we do not need to keep a counter explicitly for $x \in U$ and can give an estimate count of zero. Because at most $O(\frac{1}{\lambda})$ items can appear for at least λT times in a stream of length T, intuitively it is sufficient to keep $O(\frac{1}{\lambda})$ explicit counters. The MG Algorithm makes sure that at any point in time at most $O(\frac{1}{\lambda})$ items have explicit non-zero counts. If an item arrives and we need to create an extra counter, all existing non-zero counters are decremented by 1; this step keeps the number of non-zero counters small. On the other hand, whenever a non-zero counter of some item x decreases by 1, there are $\Theta(\frac{1}{\lambda})$ other items that also have their counts decreased by 1. Since this can happen for at most λT times, the final count for each item has additive error at most λT.

At the end of the MG Algorithm, the output corresponds to a count vector $f \in \mathbb{Z}^{\mathcal{U}}$, which has at most $O(\frac{1}{\lambda})$ non-zero coordinates. We show in Lemma 4 that this vector has sensitivity at most $O(\frac{1}{\lambda})$, and hence we can apply the techniques of geometric noise to achieve differential privacy. The properties of the private version of the MG Algorithm are given in the following lemma, which is the main result of this section.

Lemma 1 (Private MG Algorithm). *Given a privacy parameter $\epsilon > 0$ and an approximation parameter $0 < \lambda < 1$, there is a (randomized) mechanism $\mathcal{M}$, denoted as $\mathsf{PMG}(\epsilon, \lambda)$ (Private Misra-Gries), that takes any finite stream σ and after one pass outputs a vector $\widehat{f} \in \mathbb{Z}^{\mathcal{U}}$ such that the following properties hold.*

1. *ϵ-Differential Privacy: for any adjacent streams σ_1 and σ_2, any subset $\mathcal{O} \subseteq \mathbb{Z}^{\mathcal{U}}$, $\Pr[\mathcal{M}(\sigma_1) \in \mathcal{O}] \leq \exp(\epsilon) \cdot \Pr[\mathcal{M}(\sigma_2) \in \mathcal{O}]$, where the probability is over the randomness from the mechanism.*
2. *Utility: Suppose $0 < \delta < 1$, and the length T of the stream σ satisfies $T \geq \frac{32}{\lambda^2 \epsilon} \log \frac{n}{\delta}$. Then, the vector $\widehat{f}$ is $(\lambda T, \delta)$-useful with respect to the multiset $\sigma([T])$ of items in the stream.*
3. *The mechanism uses only $O(\frac{1}{\lambda})$ words of memory. In particular, at most $O(\frac{1}{\lambda})$ coordinates of $\widehat{f}$ are non-zero. Moreover, it takes $O(\frac{1}{\lambda})$ operations to process each item, and samples $O(\frac{1}{\lambda})$ random variables in total.*

We prove Lemma 1 for the rest of the section. We let $\beta := \lceil \frac{2}{\lambda} \rceil$ be a parameter of memory usage.

In the literature, it is common to achieve differential privacy by adding geometric noise. However, since we want to be careful about memory usage, we want the output of the mechanisms to be integral. Therefore, we add noises sampled from symmetric geometric distributions [11].

Definition 7 (Geometric Distribution). *Let $\alpha > 1$. We denote by $\mathsf{Geom}(\alpha)$ the symmetric geometric distribution that takes integer values such that the probability mass function at l is $\frac{\alpha-1}{\alpha+1} \cdot \alpha^{-|l|}$.*

The following property states that the symmetric geometric distribution $\mathsf{Geom}(\alpha)$ is highly concentrated around 0. Hence, we assume that each integer can be expressed using $O(1)$ words and overflow rarely happens.

Fact 1 *Let r be a random variable sampled from symmetric geometric distribution $\mathsf{Geom}(\alpha)$. Let $l \geq 0$ be a non-negative integer. Then, $\Pr[|r| > l] \leq \frac{1}{\alpha^l}$.*

The following property of symmetric geometric distribution is useful for designing differentially private counting mechanisms.

Fact 2 *Let $u, v \in \mathbb{Z}^n$ be two vectors such that $||u - v||_1 \leq \Delta$, where $||u - v||_1 = \sum_{i=1}^{n} |u_i - v_i|$ is the ℓ_1-norm of $u - v$. Let $r \in \mathbb{Z}^n$ be a random vector whose coordinates are independent random variables sampled from symmetric geometry distribution $\mathsf{Geom}(\exp(\frac{\epsilon}{\Delta}))$. Then, for any vector $p \in \mathbb{Z}^n$, $\Pr[u + r = p] \leq \exp(\epsilon) \cdot \Pr[v + r = p]$.*

Definition 8 (Sensitivity). *Let $f : \mathcal{U}^T \to \mathbb{Z}^n$ be a function that takes a stream of length T as input. The sensitivity of f, denoted by $\Delta(f)$, is $\max_{\sigma_1 \sim \sigma_2} ||f(\sigma_1) - f(\sigma_2)||_1$.*

By Fact 2, for any function $f : \mathcal{U}^T \to \mathbb{Z}^n$ such that $\Delta(f) \leq \Delta$, we can make its output ϵ-differentially private by adding independent random noise sampled from $\mathsf{Geom}(\exp(\frac{\epsilon}{\Delta}))$ to coordinates of f.

We modify the Misra-Gries Algorithm [15, 18] to get a counting mechanism, which we call Private Misra-Gries Algorithm (PMG). The algorithm is shown in Algorithm 1. We show that the mechanism has fast running time and uses small memory. For the privacy and the utility parts of Lemma 1, we give the complete proof in Appendix A.

Lemma 2 (Running Time and Memory Usage). *The algorithm $\mathsf{PMG}(\epsilon, \lambda)$ takes $O(\frac{1}{\lambda})$ operations to process each item, and samples $O(\frac{1}{\lambda})$ random variables in total. Moreover, the mechanism can be implemented using $O(\frac{1}{\lambda})$ words of memory.*

Proof. Observe that in the for loop between Lines 3 and 10, it takes $O(\frac{1}{\lambda})$ word operations per time step. At first glance, the for loop in Line 11 seems to require n iterations. We argue that this dependence on n is not necessary.

Suppose after Line 10, $A \subseteq \mathcal{U}$ is the subset of items x such that $f(x) \neq 0$. Then, we know $|A| \leq \beta$, for each $x \in A$, we sample r_x and $\widehat{f}(x) := \max\{f(x) + r_x, 0\}$ as before. Next, observe that we do not actually have to sample a fresh independent r_x from Geom for each $x \in \mathcal{U} \setminus A$ and compute $\widehat{f}(x) = \max\{r_x, 0\}$; we just need the largest β of the corresponding $\widehat{f}(x)$'s. Hence, it suffices to sample β (dependent) random variables having the same joint distribution as the β largest values from $n - |A|$ independent truncated $\max\{0, \mathsf{Geom}\}$ distributions. The details are given in Appendix D.

We can pick a random subset B of β items from $\mathcal{U} \setminus A$ to match those β random numbers such that each $x \in B$ has a corresponding $\widehat{f}(x)$. Finally, we just need to find the β largest numbers in $\{\widehat{f}(x) : x \in A \cup B\}$. Hence, we just need to sample $O(\frac{1}{\lambda})$ random variables in total.

In Algorithm 1, at any time during the execution between Line 1 and Line 10, there are at most $\beta + 1$ items x such that $f(x) > 0$. Moreover, because of the conditional statement at Line 14, the number of items x such that $\widehat{f}(x) \neq 0$ is at most $\beta + 1$. Since we only need to explicitly store the non-zero values of $f(x)$ and $\widehat{f}(x)$, the above algorithm uses $O(\beta) = O(\frac{1}{\lambda})$ words of memory. □

3.3 Private Continual Heavy-Hitter Monitoring over a Sliding Window

In this section, we use PMG to construct a differentially private mechanism named Private Continual Heavy-hitter (PCC), that uses small memory and maintains some efficient data structure at every time step.

```
Input: A privacy parameter $\epsilon$, an approximation parameter $\lambda$, and a finite stream $\sigma \in \mathcal{U}^T$
       of length $T$.
Output: A vector $\widehat{f} \in \mathbb{Z}^{\mathcal{U}}$, where $\widehat{f}(x)$ is a $\lambda$-approximate count of $x$ with respect to
        $\sigma([T])$ with high probability.
1  For each $x \in \mathcal{U}$, $f(x)$ and $\widehat{f}(x)$ are (implicitly) initialized to 0;
2  $\beta \leftarrow \lceil \frac{2}{\lambda} \rceil$;
3  for $t \leftarrow 1$ to $T$ do
4      $f(\sigma(t)) \leftarrow f(\sigma(t)) + 1$;
5      if the number of items $x$ such that $f(x) > 0$ exceeds the threshold $\beta$ then
           // the decreasing step
6          for $x \in \mathcal{U}$ such that $f(x) > 0$ do
7              $f(x) \leftarrow f(x) - 1$;
8          end
9      end
10 end
   // The sensitivity $\Delta(f) \leq \beta + 1$
11 for each $x \in \mathcal{U}$ do
       // We see later in Lemma 2 and Appendix D that there is a
          faster procedure that samples only $O(\beta)$ random
          variables and achieves the same output distribution.
12     Sample a fresh independent noise $r_x \sim \mathsf{Geom}(\exp(\frac{\epsilon}{\beta+1}))$;
13     $\widehat{f}(x) \leftarrow \max\{f(x) + r_x, 0\}$;
       // we only keep the top $\beta$ non-zero $\widehat{f}(x)$'s
14     if there are more than $\beta$ items $x$ with $\widehat{f}(x) \neq 0$ then
15         Let $y$ be the item with smallest non-zero $\widehat{f}(y)$ (resolving ties arbitrarily);
16         Set $\widehat{f}(y) \leftarrow 0$;
17     end
18 end
19 Output $\widehat{f}$;
```

Algorithm 1: Private Misra-Gries Algorithm $\mathsf{PMG}(\epsilon, \lambda)$

As intuitively illustrated as in Figure 2, we build a binary interval tree, where each leaf node represents $\lceil \frac{\lambda W}{4} \rceil$ (= 1 in the figure) time steps, and each non-leaf node represents a range of time steps. (Note that Figure 2 only depicts the bottom few levels of this binary interval tree due to reasons stated below). We refer to each node as a *block*, and run the subroutine PMG algorithm for each block, with appropriate privacy and approximation parameters. To output the heavy hitters for any time range, it suffices to "sum up" a logarithmic number of blocks in the binary interval tree – since any range can be represented by the union of a logarithmic number of disjoint blocks. In Figure 2, since we consider a window size of $W = 7$, we only need the bottom 3 levels of the binary tree. The main purpose of the binary interval tree technique is to save memory and allow faster computation.

Small memory. The amount of memory necessary is $\tilde{O}(\frac{1}{\lambda})$ independent of W. To achieve this, we use a *garbage collection* technique, where a data source saves only

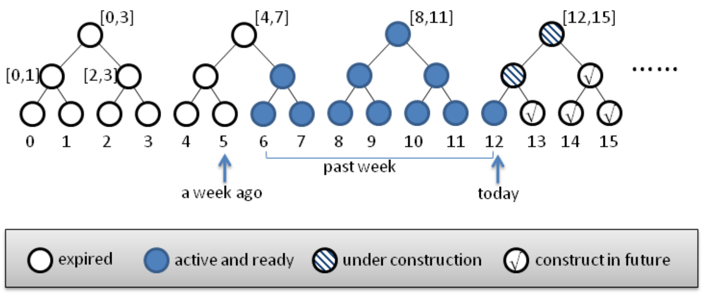

Fig. 2. Continual counting over a sliding window of the past week.

the blocks which will later be needed, and discard all "expired" blocks which will no longer be needed. As shown in Figure 2, at any point of time, a block can be one of the following four types: 1) *expired*, i.e., will no longer be needed in the future; 2) *active*, i.e., the block has completed construction, and will be needed now or in the future; 3) *under-construction*, i.e., the heavy hitters for this block are currently being constructed; or 4) *future*, i.e., construction for this block will start at some point in the future. Specifically, in the PCC algorithm, a data source saves only the blocks that are either active or under construction – and the number of such nodes is $O(\frac{1}{\lambda})$ at any point in time. Observe that the number of counters kept for binary nodes at different levels are different, and we later give a calculation to show that the total number of counters kept at any time is $O(\frac{1}{\lambda}\log^2\frac{1}{\lambda})$.

Faster Computation. Observe that we could achieve even smaller memory if we only store the leaf nodes of the binary construction. However, in order to produce an estimate count over a window, we would need to look at $\Omega(\frac{1}{\lambda})$ leaf nodes. Using the binary construction, we only need to look at $O(\log\frac{1}{\lambda})$ binary tree nodes for count estimation at each step.

Low Communication Bandwidth. Notice we could achieve even smaller memory if only one leaf node (corresponding to $W_0 = \lceil\frac{\lambda W}{4}\rceil$ time steps) in the binary tree construction is stored at any time; however, updates need to be sent to the aggregator every W_0 time steps. As we shall see in Section 3.4, if information about each stream in the current window is kept at each node, then the communication bandwidth to the aggregator can be greatly reduced.

Note that a similar binary tree technique was also used in [1, 4].

We now give a formal description of the PCC algorithm, as well as its theoretic guarantees. At every time step t, PCC maintains a dictionary $\mathcal{P}_t$, which is a collection of at most $O(\frac{1}{\lambda}\log\frac{1}{\lambda})$ pairs $(x, c_x) \in \mathcal{U}\times\mathbb{Z}$ where for every item $x \in \mathcal{U}$, item x appears in at most one pair in $\mathcal{P}_t$. Hence, $\mathcal{P}_t$ can also be interpreted as a vector $\mathbb{Z}^{\mathcal{U}}$ or a function from $\mathcal{U}$ to $\mathbb{Z}$ in the natural way: $\mathcal{P}_t(x) := c_x$ if $(x, c_x) \in \mathcal{P}_t$, and $\mathcal{P}_t(x) := 0$ otherwise. Observe that only non-zero counts need to be stored in the dictionary , and we denote by $|\mathcal{P}_t|$ the number of items x having non-zero counts $\mathcal{P}_t(x)$. The following lemma is the main result of the section.

Lemma 3 (Private Continual Heavy Hitter Monitoring). *Given a privacy parameter $\epsilon > 0$, and an approximation parameter $0 < \lambda < 1$, there exists a continual counting mechanism $\mathcal{M}$, denoted as $\mathsf{PCC}(\epsilon, \lambda)$ (Private Continual Heavy-hitter), that takes an infinite data stream $\sigma \in \mathcal{U}^{\mathbb{N}}$, and maintains at every time step t a dictionary $\mathcal{P}_t \in \mathbb{Z}^{\mathcal{U}}$. We write $\mathcal{M}(\sigma) := (\mathcal{P}_t)_{t \in \mathbb{N}} \in \mathbb{Z}^{\mathbb{N} \times \mathcal{U}}$. The following properties hold.*

1. *ϵ-Differential Privacy: for any adjacent streams σ_1 and σ_2, any subset $\mathcal{O} \subseteq \mathbb{Z}^{\mathbb{N} \times \mathcal{U}}$, $\Pr[\mathcal{M}(\sigma_1) \in \mathcal{O}] \leq \exp(\epsilon) \cdot \Pr[\mathcal{M}(\sigma_2) \in \mathcal{O}]$, where the probability is over the randomness from the mechanism.*
2. *Utility: Suppose $0 < \delta < 1$ and $L > 0$. If $W \geq \Theta(\frac{1}{\lambda^2 \epsilon}(\log^2 \frac{1}{\lambda}) \cdot \log(\frac{n}{\delta} \log \frac{1}{\lambda}))$. Then, at every time step $t \in \mathbb{N}$, $\mathcal{P}_t$ is $(\lambda W, \delta)$-useful with respect to $\mathcal{W}_t$. If $W \geq \Theta(\frac{1}{\lambda^2 \epsilon}(\log^2 \frac{1}{\lambda}) \cdot \log(\frac{L+W}{\lambda W} \cdot \frac{n}{\delta}))$, then for any $T \geq L+1$, $\{\mathcal{P}_t\}_{t \in [T-L,T]}$ is simultaneously $(\lambda W, \delta)$-useful with respect to $\{\mathcal{W}_t\}_{t \in [T-L,T]}$.*
3. *The mechanism uses only $O(\frac{1}{\lambda} \log^2 \frac{1}{\lambda})$ words of memory. Moreover, it takes $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ operations to process each item in the stream, and samples at most $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ random variables in each time step.*
4. *For every time step t, $|\mathcal{P}_t| = O(\frac{1}{\lambda} \log \frac{1}{\lambda})$.*

Scheme Description $\mathsf{PCC}(\epsilon, \lambda)$**.** We assume $\frac{1}{\lambda}$ is a power of 2; otherwise, let $\lambda' := \max\{\frac{1}{2^k} : \frac{1}{2^k} < \lambda\}$ and run $\mathsf{PCC}(\epsilon, \lambda')$.

Let $W_0 := \lceil \frac{\lambda W}{4} \rceil > 0$, and let $\ell := \log \frac{4}{\lambda}$. Note that we have $\frac{W}{2} \leq W_0 \cdot 2^\ell \leq W$. We divide the time steps into binary hierarchical blocks with $\ell + 1$ levels, where all blocks at the same level have the same size, are disjoint and cover all time steps. In particular, for $0 \leq i \leq \ell$ and $j \geq 1$, the jth block at level i is the interval $[(j-1)W_i + 1, jW_i]$ of $W_i := 2^i \cdot W_0$ time steps; we use block $\mathcal{B}_j^i$ to denote the multiset of items from stream σ contained in this interval, i.e., $\mathcal{B}_j^i = \sigma([(j-1)W_i + 1, jW_i])$. At any time t, each block $\mathcal{B}_j^i$ is in one of the following four states.

1. **future:** None of $\mathcal{B}_j^i$'s items has come into $\mathcal{W}_t(\sigma)$, i.e., $(j-1)W_i + 1 > t$;
2. **under-construction:** Some of $\mathcal{B}_j^i$'s items are in $\mathcal{W}_t(\sigma)$, and the remaining items have not come into $\mathcal{W}_t$, i.e., $(j-1)W_i + 1 \leq t$ and $jW_i > t$;
3. **active:** $\mathcal{B}_j^i$ is totally within $\mathcal{W}_t(\sigma)$, i.e., $t - W + 1 \leq (j-1)W_i + 1 \leq jW_i \leq t$;
4. **expired:** At least one of $\mathcal{B}_j^i$'s items has expired, i.e., $(j-1)W_i + 1 < t - W + 1$.

For each block $\mathcal{B}_j^i$, right before the time step when its state becomes under-construction, i.e., $t = (j-1)W_i + 1$, $\mathsf{PCC}(\epsilon, \lambda)$ initiates an instance of $\mathsf{PMG}(\epsilon_i, \lambda_i)$ on $\mathcal{B}_j^i$, where $\epsilon_i := \frac{\epsilon}{2^{\ell - i + 1}}$ and $\lambda_i := \frac{1}{2^i(\ell+1)}$. When $\mathcal{B}_j^i$ becomes active, $\mathsf{PMG}(\epsilon_i, \lambda_i)$ produces a vector $\widehat{f}_{\mathcal{B}_j^i} \in \mathbb{Z}^{\mathcal{U}}$, which has $O(\frac{1}{\lambda_i})$ non-zero coordinates. Then, $\mathsf{PCC}(\epsilon, \lambda)$ uses $O(\frac{1}{\lambda_i})$ words of memory to maintain this vector until $\mathcal{B}_j^i$ expires.

Cover by Disjoint Active Blocks. Observe that at any time t, there exists a collection $\mathcal{C}_t$ of disjoint active blocks, such that $\mathcal{C}_t$ contains at most two blocks from each level, and that the union of blocks in $\mathcal{C}_t$ is the union of all active level-0 blocks. At any time, $\mathsf{PCC}(\epsilon, \lambda)$ maintains the dictionary $\mathcal{P}_t$, where $\mathcal{P}_t = \{(x, \sum_{\mathcal{B} \in \mathcal{C}_t} \widehat{f}_{\mathcal{B}}(x)) : x \in \mathcal{U}$ and $\sum_{\mathcal{B} \in \mathcal{C}_t} \widehat{f}_{\mathcal{B}}(x) > 0\}$.

We prove that PCC maintains differential privacy. The analysis of the remaining properties is similar to that in [1, Section 5], and we include the proofs in Appendix B.

Privacy Guarantee. Observe that the output of PCC is a deterministic function of PMG's outputs on all blocks. Hence, we need only to show ϵ-differential privacy is maintained with respect to $(\widehat{f_{\mathcal{B}}})_{\forall \mathcal{B}}$. Consider an item arriving at time step t. We analyze which of the blocks would be affected if $\sigma(t)$ is replaced with a different item. It is not hard to see that the item $\sigma(t)$ can be in at most one block at each level. Observe that $\widehat{f_{\mathcal{B}}}$ for a level-i block $\mathcal{B}$ maintains ϵ_i-differential privacy, where $\epsilon_i = \frac{\epsilon}{2^{\ell-i+1}}$ and observe that $\sum_{i=0}^{l} \epsilon_i \leq \epsilon$. Hence, we conclude that $\mathsf{PCC}(\epsilon, \lambda)$ preserves ϵ-differential privacy.

3.4 Privately and Continually Monitoring Heavy Hitters Across Distributed Streams

In this section, we use PCC to design a protocol between k data sources and an aggregator, allowing the aggregator to continually monitor the global heavy hitters. The resulting protocol, called PDCH-LU (which stands for *Private Distributed Continual Heavy-hitter - Lazy Update*) has low communication cost; moreover, the messages sent by each data source is differentially private against the aggregator.

Observe that each node could send the privatized updates to the aggregator at every time step in order for the aggregator to compute the approximate heavy hitters. However, to save communication bandwidth, we use a *lazy update* approach: updates are only required when the count of an item changes by a huge amount. Chan et al. [3] gave a distributed algorithm (called Approximate Counting (AC)) based on this idea and proved that it achieves small error. Since the AC Algorithm in [3] only needs an approximate count in the current window for each item in each stream at any time, our PCC algorithm is sufficient for this purpose. We give the main result and the construction of the protocol; the detailed analysis is given in Appendix C.

Theorem 1. *Suppose $\epsilon > 0$ is a privacy parameter, $0 < \lambda < 1$ is an approximation parameter, W is the window size and L is some positive integer. Given k streams each received by a node, every node can run an ϵ-differentially private communication protocol with the same time and space performance as $\mathsf{PCC}(\epsilon, \frac{\lambda}{11})$ in Lemma 3 to send messages to the aggregator such that if $W \geq \Theta(\frac{1}{\lambda^2 \epsilon}(\log^2 \frac{1}{\lambda}) \log(\frac{L+W}{\lambda W} \cdot \frac{kn}{\delta}))$, then for every time interval $\mathcal{T} = [T+1, T+L]$ (where $T \geq W$), with probability at least $1-\delta$, at every time $t \in \mathcal{T}$, the aggregator can maintain a λ-approximate count for every item with respect to the current window in all streams, and the total communication cost by all nodes in the period $\mathcal{T}$ is $O(\frac{k}{\lambda} \cdot \lceil \frac{L}{W} \rceil \log W)$ words.*

Algorithm for the Aggregator. The aggregator maintains a counter $c_i(x)$ (initially 0) for each stream $i \in [k]$ and each item $x \in \mathcal{U}$. Upon receiving a message $\langle x, c \rangle$ from node i, the aggregator updates the counter $c_i(x) := c$. In each time step t, the aggregator calculates a count $c(x) = \sum_{i \in [k]} c_i(x)$; to produce a 2λ-approximate set of θ-heavy hitters, the aggregator releases the set of items x such that $c(x) \geq (\theta - \lambda)kW$.
Protocol for Each Data Source. We use Algorithm AC (Approximate Counting) in [3, Section 2.2] to get a protocol (shown in Algorithm 2), denoted as $\mathsf{PDCH\text{-}LU}(\epsilon, \lambda)$ (Private Distributed Heavy-hitter - Lazy Update). Each node i runs an instance of $\mathsf{PDCH\text{-}LU}(\epsilon, \lambda)$ on the stream $\sigma^{(i)}$ it receives.

```
Input: A privacy parameter $\epsilon$, an approximation parameter $\lambda$, and a stream $\sigma \in \mathcal{U}^{\mathbb{N}}$
Run an instance of $\mathsf{PCC}(\epsilon, \frac{\lambda}{11})$ and (implicitly) initialize $\mathsf{Last}(x) := 0$ for each $x \in \mathcal{U}$;
// We only store non zero $\mathsf{Last}(x)$'s.
for $t \leftarrow 1$ to $\infty$ do
    for each $x$ such that $\mathcal{P}_t(x) > 0$ or $\mathsf{Last}(x) > 0$ do
        Up: if $\mathcal{P}_t(x) > \mathsf{Last}(x) + \frac{9}{11} \cdot \lambda W$, send $\langle x, \mathcal{P}_t(x) \rangle$ and set $\mathsf{Last}(x) \leftarrow \mathcal{P}_t(x)$;
        Off: if $\mathsf{Last}(x) > 0$ and $\mathcal{P}_t(x) < \frac{3}{11} \cdot \lambda W$, send $\langle x, 0 \rangle$ and set $\mathsf{Last}(x) \leftarrow 0$;
        Down: if $\mathcal{P}_t(x) < \mathsf{Last}(x) - \frac{9}{11} \cdot \lambda W$, send $\langle x, \mathcal{P}_t(x) \rangle$ and set
        $\mathsf{Last}(x) \leftarrow \mathcal{P}_t(x)$;
    end
end
```

Algorithm 2: $\mathsf{PDCH\text{-}LU}(\epsilon, \lambda)$

4 Achieving Aggregator Obliviousness

The main construction described earlier in Section 3 is a protocol for k nodes to communicate with an aggregator, whose task is to keep track of heavy hitters over a sliding window. This protocol guarantees differential privacy at the event level, i.e., the statistics released to the aggregator is not affected by the change of one event.

In this section, we describe a protocol which achieves a stronger level of privacy protection, i.e., we additionally achieve aggregator obliviousness on top of event-level differential privacy. Specifically, we wish to reveal the minimum amount of information possible to the aggregator, for it to successfully perform the heavy hitter monitoring task.

The main techniques we use to achieve aggregator obliviousness include Bloom filters [2] as well as special encryption schemes [14, 19, 20] that support the controlled decryption of selected statistics. Using these techniques to augment the PMG protocol described earlier, we achieve aggregator obliviousness in the sense that the aggregator learns only the approximate counts of each item, but nothing else. In particular, the aggregator does not learn which data sources are contributing to the heavy hitters and how much their contributions are.

In our protocol to be described later in this section, each node communicates the update with the aggregator every $W_0 = \lceil \frac{\lambda W}{4} \rceil$ time steps. We then employ Bloom filters to effectively reduce the bandwidth overhead. Observe that without the Bloom filters, we would need to perform n secure additions, one for each item, for each update. We shall see that using Bloom filters, we can reduce the dependence of the number of additions per update on n to $O(\log n)$.

4.1 Background on Special Encryption Scheme

As a building block for achieving aggregator obliviousness, we employ a special encryption scheme which supports the conditional decryption of selected statistics. In particular, we can use either the encryption scheme proposed by Shi *et al.* [20], Rastogi *et al.* [19], or Kursawe *et al.* [14] In comparison, the scheme by Shi *et al.* [20] requires uni-directional communication from the data sources to the aggregator, but the decryption algorithm is more expensive; whereas the scheme by Rastogi *et al.* [19] requires bi-directional communication between the data sources and the aggregator, but has smaller decryption overhead. The scheme by Kursawe *et al.* [14] only needs uni-directional communication and has low overhead, but each node needs to store $\Theta(k)$ keys corresponding to all other nodes. We now give a high-level overview of these special encryption schemes. The special encryption schemes we employ typically involve the following algorithms or phases:

Setup. In a one-time setup phase, cryptographic keying materials are distributed to all data sources and the aggregator. In particular, each data source receives an encryption key, and the aggregator receives a cryptographic capability which will allow it to later decrypt the sum of all data sources in each aggregation time step. The setup phase can either be performed by an offline authority (which will no longer be needed after the setup phase); or through an interactive multi-party protocol amongst the data sources and the aggregator.

Periodic Encryption and Aggregation. In each time step, each data source $i \in [k]$ encrypts a value x_i using the encryption key established in the setup phase, and sends the ciphertext to the aggregator. After receiving ciphertexts from all data sources, the aggregator can use its cryptographic capability to decrypt the value $\sum_{i=1}^{k} x_i$, but learn nothing else. In the construction by Shi *et al.* [20] and Kursawe *et al.* [14], this decryption is done solely by the aggregator, whereas in the scheme by Rastogi *et al.* [19], the aggregator needs to communicate with the data sources to perform decryption.

In the remainder of this section, we will use this special encryption scheme as a blackbox – for more algebraic details on how these schemes are constructed, we refer the readers to [14, 19, 20].

4.2 Augmenting PMG Algorithm with Secure Bloom Filters

Straightforward Solution using Cryptography. We first describe a straightforward construction using one of the special encryption schemes [14, 19, 20]. A brief background on these encryption schemes was given in Section 4.1. We show that one drawback of this straightforward construction is its high bandwidth overhead. Later, we shall employ Bloom filters to reduce the bandwidth consumption.

The basic idea is to apply the special encryption scheme all items in the universe.

Suppose each of the k nodes is running $\mathsf{PMG}(\epsilon, \lambda)$ on its finite stream as described in Section 3.2. Each node $v \in [k]$ produces some $\widehat{f}_v : \mathcal{U} \to \mathbb{Z}$, which is represented by at most $\beta = O(\frac{1}{\lambda})$ non-zero counters. In every time step, for each $x \in \mathcal{U}$, each data source encrypts its observed frequency $\widehat{f}_v(x)$, and sends the ciphertext to the aggregator. The aggregator may then use its cryptographic capability to decrypt the total frequency $\sum_{v \in [k]} \widehat{f}_v(x)$ for each $x \in \mathcal{U}$ – and meanwhile, the security of these encryption schemes guarantee that the aggregator learns nothing else beyond the total frequency of each item.

It is not hard to see that each node needs to send $\Omega(n)$ words (proportional to the size of $\mathcal{U}$) to the aggregator. Even though each node only has β non-zero counters, it still has to participate in every addition such that the aggregator does not know where the non-zero values come from. We would like to decrease the communication cost through the use of Bloom filters.

Construction with Bloom Filters. Let $0 < \delta < 1$ be the desired failure probability, i.e., with probability at least $1 - \delta$, the aggregator can retrieve $\sum_{v \in [k]} \widehat{f}_v(x)$ for all $x \in \mathcal{U}$. Let $P := \lceil \ln \frac{n}{\delta} \rceil$ and $Q := \lceil ek\beta \rceil$, where e is the natural number and β is the maximum number of non-zero counters for each node. We assume there is a public family $\{\mathcal{H}_p : \mathcal{U} \to [Q]\}_{p \in [P]}$ of random hash functions that satisfy the following properties.

1. The functions $\mathcal{H}_p$ are totally independent over different $p \in [P]$.
2. For each $p \in [P]$, $\mathcal{H}_p$ is pairwise independent over $\mathcal{U}$, i.e., for $x \neq y$, $\mathcal{H}_p(x)$ and $\mathcal{H}_p(y)$ are independent; moreover, for each $x \in \mathcal{U}$ and each $q \in [Q]$, $\Pr[\mathcal{H}_p(x) = q] = \frac{1}{Q}$.

Bloom Filters for Each Node. Each node v constructs a table A_v of size $P \times Q$ that is constructed in the following way.

1. Initially, every entry $A_v[p][q] := 0$.
2. For each $x \in \mathcal{U}$ such that $\widehat{f}_v(x) \neq 0$ (note that there are at most β such x's), for each $p \in [Q]$, increment $A[p][\mathcal{H}_p(x)]$ by $\widehat{f}_v(x)$.

Secure Addition of Bloom Filters. Using one of the secure periodic schemes [14, 19, 20] described above, the aggregator learns for each $p \in [P]$ and $q \in [Q]$, $A[p][q] := \sum_{v \in [k]} A_v[p][q]$. Observe that each node sends $O(PQ) = O(\frac{k}{\lambda} \log \frac{n}{\delta})$ words to the aggregator.

Retrieving Sum of Counts for Each Item. For item x, the aggregator computes $\min_{p \in [P]} A[p][\mathcal{H}_p(x)]$.

Theorem 2. *With probability at least* $1 - \delta$*, the aggregator retrieves* $\sum_{v \in [k]} \widehat{f}_v(x)$ *accurately for all* $x \in \mathcal{U}$*.*

Proof. Fix some $x \in \mathcal{U}$. Observe that the aggregator makes a mistake for item x *iff* for all $p \in [P]$, there exists some $y \neq x$ such that $\mathcal{H}_p(x) = \mathcal{H}_p(y)$ and there exists $v \in [k]$ such that $\widehat{f}_v(y) \neq 0$.

Observe that each node v can only have at most β non-zero counts. For fixed $p \in \mathcal{P}$, by pairwise independence of $\mathcal{H}_p$ over $\mathcal{U}$ and uniformity of $\mathcal{H}_p(y)$ over $[Q]$, the probability that there exists some $y \neq x$ such that some node has non-zero count for y and $\mathcal{H}_p(x) = \mathcal{H}_p(y)$ is at most $\frac{k\beta}{Q} \leq \frac{1}{e}$.

By the total independence of $\mathcal{H}_p$ over p's, it follows that the probability that the aggregator makes a mistake for item x is at most $\frac{1}{e^P} \leq \frac{\delta}{n}$. Using the union bound over all $x \in \mathcal{U}$, it follows that the probability that the aggregator makes a mistake for some item is at most δ, as required.

4.3 Distributed Protocol Achieving Aggregator Obliviousness

We next describe how the PMG Algorithm augmented with Bloom Filters can be used to design a distributed protocol that achieves aggregator obliviousness – the resulting protocol is referred to as PDCH-BF (Private Distributed Continual Heavy-hitter - Bloom Filter).

Each node performs the binary construction as in the PCC algorithm described in Section 3.3. In particular, for a block at level i with size $W_i = 2^i \cdot W_0 = 2^i \cdot \frac{\lambda W}{4}$, $\mathsf{PMG}(\epsilon_i, \lambda_i)$ is run with $\epsilon_i := \frac{\epsilon}{2^{\ell - i + 1}}$ and $\lambda_i := \frac{1}{2^i(\ell+1)}$. As soon as PMG is completed for a block, a Bloom filter is constructed for that block as described in Section 4.2. The Bloom filter for that block is encrypted and sent to the aggregator.

For a particular block, after the aggregator has received the ciphertexts of the Bloom filters from all the k nodes, it can decrypt the sum of the Bloom filters and reconstruct the counts for all items in that block.

Observe that to estimate the counts in a window, the aggregator just needs the counts from at most $2\log\frac{1}{\lambda}$ blocks. Hence, in order to achieve failure probability of δ due to the Bloom filters for each window, it suffices to set the Bloom filter failure probability for each block to be $\frac{\delta}{2\log\frac{1}{\lambda}}$. There is also failure probability δ due to the randomness introduced to achieve differential privacy. Hence, Lemma 3 implies the aggregator can compute λ-approximate heavy hitters within a window with probability $1 - 2\delta$.

Communication Cost. For each node, the communication cost for a block at level i is $O(\frac{k}{\lambda_i}\log\frac{n}{\delta_0})$ words, where $\lambda_i = \frac{1}{2^i \cdot (l+1)}$, $l = \log\frac{4}{\lambda}$ and $\delta_0 = \frac{\delta}{2(l+1)}$. Moreover, a block at level i is constructed every $W_i := 2^i \cdot W_0 = 2^i \cdot \frac{\lambda W}{4}$ time steps. Hence, it follows that the average number of words of communication per time step is $O(\frac{k}{\lambda W}\log^2\frac{1}{\lambda}\log\frac{n\log\frac{1}{\lambda}}{\delta})$.

5 Experiments

5.1 Experimental Setup

The data used in our experiment was constructed from the Netflix Contest Dataset, which contains $n = 17770$ movies with 480189 users' ratings from 1999-11-11 to 2005-12-31. We divided the users randomly into 100 groups to construct 100 streams. We selected roughly 200 days' data of each stream from 2002-09-23 to 2003-04-30 to conduct our experiments, where we continually monitored the moving average over a window size of 90 days. We plot the result for the last 10 days by when the system should have become stable. We use the following parameters in our experiments: heavy-hitter fraction $\theta = 0.004$ and error $\lambda = 0.001$. In our experiments, we consider differential privacy parameter $\epsilon = 1, 2, 5, 10$. Note that there is no consensus on what privacy parameters are acceptable in practice, and even for $\epsilon = 1$, the scheme still offers some guarantee on privacy.

For the Bloom filters, we choose the number of hash functions to be $P = 8$, and the array size for each hash function to be $Q = \lceil e \cdot 3600 \rceil$ (we explain the choice of Q below).

Practical Optimization. Several aspects of our protocols can be further optimized in this application.

1. *Empirical Sensitivity of* PMG *Count Vector and Bloom Filter Size.* Since the value of λ we choose is small, in practice, the number of distinct movies observed by each node in each day never exceeds $O(\frac{1}{\lambda})$, the number of counters in PMG. If we assume that the daily number of distinct movies never exceeds the number of counters in PMG, the real sensitivity is $O(1)$, and so we can use $\mathsf{Geom}(e^{\epsilon})$ noise for

PMG, instead of $\mathsf{Geom}(e^{\Theta(\epsilon\lambda)})$. Moreover, we observe that the number of distinct movies observed by all nodes in each day never exceeds 3600 and hence we can choose $Q = \lceil e \cdot 3600 \rceil$ to be the size of the Bloom filter array for each hash.

2. *Unnecessary Internal Blocks in Binary Construction.* Since the width W of our window is small, $W_0 = \lceil \frac{\lambda W}{4} \rceil = 1$. Hence, each leaf node in the binary tree corresponds to one time step, and therefore, the binary construction does not help to save memory in this case. Moreover, since W is small, the cost of count reconstruction from W leaf nodes is still better than the overhead of keeping track of the binary construction.
3. *Relaxing Assumption on Item Arrival.* Since we no longer need the binary construction, we can further relax the assumption that exactly one item arrives at each stream per time step. In our dataset, at least one item arrives at each stream at each time step, and we do not need any upper bound on the number of arriving items. Within each time step, PMG for each stream serializes the arriving items and process them in any arbitrary order.
4. *Estimating the Total Number of Arriving Items.* With the assumption that only one item arrives at each time step for each stream, it is trivial to compute the total number items within a window; this quantity is required for estimating the fraction an item appears and also for deciding when to do lazy updates (this is the W in the conditions for **Up**, **Off** and **Down** in PDCH-LU). However, now each stream also needs to report to the aggregator the number of items that arrive each day, which has sensitivity 1 for neighboring streams. Hence, in order to achieve ϵ-differential privacy for the whole protocol, we can assign $\epsilon_1 := \frac{9\epsilon}{10}$ and $\epsilon_2 := \frac{\epsilon}{10}$ such that each stream runs PMG with ϵ_1-differential privacy (using $\mathsf{Geom}(e^{\epsilon_1})$ noise) and estimates the number of items each day with ϵ_2-differential privacy (using $\mathsf{Geom}(e^{\epsilon_2})$ noise).[1]

Performance Metric. We consider the following performance measures. For each performance measure, we plot its mean with an error bar of 2 standard deviations (hence each plot could go negative).

1. *Error in estimated fraction.* For each day, suppose H is the set of items whose true fraction in the current window is at least θ and $\widehat{H}$ is the set of items whose estimated fraction in the current window is at least $\theta - \lambda$. For each item $x \in H \cup \widehat{H}$, we calculate the error $\mathcal{E}(x)$ which is the absolute difference between the true and the estimated fractions of item x in the current window. We compute the mean and the standard deviation of $\mathcal{E}(x)$ over x in $H \cup \widehat{H}$. We evaluate these statistics over time for different protocols with various parameters.
2. *Communication Cost.* We measure the number of words each node sends to the aggregator each day. For each day, we compute the mean and the standard deviation of the communication cost over different nodes.

[1] We give more privacy budget to PMG as it is more complicated, and less privacy budget to the estimation of number of items each day as it is relatively simpler. It does not really affect the asymptotic error as long as each part get a constant fraction of the privacy budget, but experiment suggests that these parameters work well.

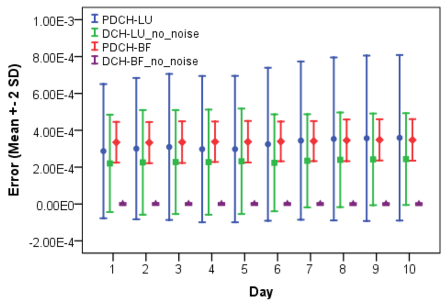

Fig. 3. Error under different protocols

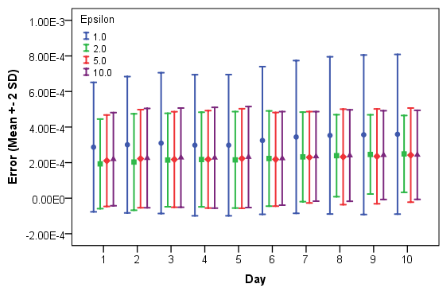

Fig. 4. Error under different privacy parameters for the PDCH-LU algorithm.

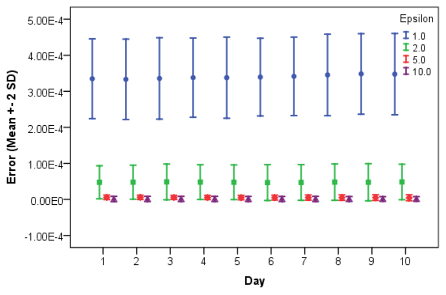

Fig. 5. Error under different privacy parameters for the PDCH-BF algorithm.

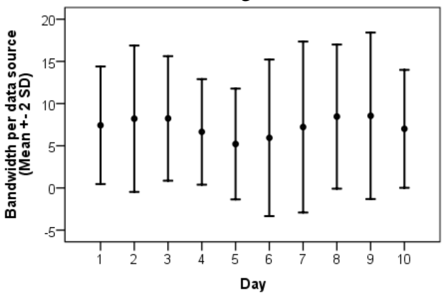

Fig. 6. The number of words sent per data source in the PDCH-LU protocol.

5.2 Results

Utility. In Figures 3, 4, and 5, we observe that the error in each case is well below the theoretic guarantee $\lambda = 0.001$, and we interpret each figure as follows.

Figure 3 compares the errors of the PDCH-LU and PDCH-BF protocols, and also demonstrates a breakdown of the error, i.e., how much error is introduced by compressing the bandwidth, and how much due to the noise necessary for differential privacy. With our choice of parameters, the Bloom filter should introduce almost no error and hence DCH-BF(no noise) forms the baseline for comparison. The plot for protocol PDCH-BF essentially reflects the error introduced when we wish to preserve differential privacy. The plot for DCH-LU(no noise) reflects the error introduced by lazy update in order to save communication bandwidth. The interesting unexpected result is that for PDCH-LU when we use lazy update together with noise to ensure differential privacy, the extra noise does not seem to increase the error by much. In fact, the effect of lazy updates seem to smooth out some of the error introduced by the added random noise.

Figures 4 and 5 plot the utility of PDCH-BF and PDCH-LU under different differential privacy parameter ϵ. As we decreases ϵ, the magnitude of noise increase and we can see in Figure 5 that as expected the error for PDCH-BF is increased as well. However, we can see that in Figure 4, that reducing ϵ has only a small effect on the performance of PDCH-LU.

Communication cost. Figure 6 shows that in the PDCH-LU protocol, the number of item updates sent by each node per day is around 5 and almost never above 20. Typically, each item update is under 10 bytes of data.

In comparison, we need to pay higher (but still reasonable) communication cost if we wish to ensure aggregator obliviousness. In our PDCH-BF experiment, given 8 hashes each having a filter of size $Q = \lceil e \cdot 3600 \rceil$, and assuming that each Diffie-Hellman ciphertext is 1024 bits, then the each sends about $10MB$ data per day to the aggregator.

6 Conclusion

We proposed novel privacy-preserving protocols allowing an untrusted aggregator to continually monitor heavy hitters across a set of distributed streams. Our protocols achieve two levels of privacy protection, namely, differential privacy and aggregator obliviousness. Both theoretic proofs and experimental results show that our protocols achieve good utility in realistic settings, and introduce low memory, computational, and bandwidth overhead at each data source.

References

[1] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, 2004.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[3] H.-L. Chan, T.-W. Lam, L.-K. Lee, and H.-F. Ting. Continuous monitoring of distributed data streams over a time-based sliding window. In *STACS*, 2010.

[4] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. In *ICALP (2)*, pages 405–417, 2010.

[5] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *ESA*, pages 348–360, 2002.

[6] C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.

[7] C. Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, 2011.

[8] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *STOC*, pages 715–724, 2010.

[9] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[10] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *KDD*, pages 265–273, 2008.

[11] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *STOC*, 2009.

[12] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[13] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28:51–55, 2003.

[14] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *PETS*, pages 175–191, 2011.

[15] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *PODS*, 2006.

[16] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD Conference*, pages 19–30, 2009.

[17] D. J. Mir, S. Muthukrishnan, A. Nikolov, and R. N. Wright. Pan-private algorithms via statistics on sketches. In *PODS*, pages 37–48, 2011.

[18] J. Misra and D. Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

[19] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD 2010*, pages 735–746, 2010.

[20] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.

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

A Proofs for PMG

Privacy Guarantee. The Misra-Gries Algorithm keeps $O(\frac{1}{\lambda})$ non-zero counters at any time. For an item x that does not have an explicit counter, its count is interpreted as zero. Hence, we can imagine that the algorithm maintains some vector that has at most β non-zero coordinates at any time. Hence, after reading the whole stream (after Line 10), we can view the resulting vector obtained as a function $f : \mathcal{U}^T \rightarrow \mathbb{Z}^{\mathcal{U}}$. If we can show that the sensitivity of f is at most $\beta + 1$, then by Fact 2, by adding independent random noise sampled from $\mathsf{Geom}(\exp(\frac{\epsilon}{\beta+1}))$ to coordinates of f, the resulting vector will be ϵ-differentially private. After that, rounding negative values to zero and picking the largest β counters are deterministic actions, and therefore releasing $\widehat{f}$ in Line 19 preserves ϵ-differential privacy.

The difficulty is that even if two streams differ only at one time step, the corresponding counters can be drastically different. In one case, the new item causes an existing counter to increase by one, while a different item could cause every non-zero counter to decrease by 1, possibly causing all counters to be zero! The following lemma analyzes the sensitivity of f and the technical challenge is to show that starting from two adjacent streams, the two corresponding vectors of counters at every time step satisfy one of four structural states, where in each case the ℓ_1-distance between the two vectors is at most $\beta + 1$.

Lemma 4. *In Algorithm 1 after Line 10, the function $f : \mathcal{U}^T \rightarrow \mathbb{Z}^{\mathcal{U}}$ has sensitivity at most $\beta + 1$.*

Proof. Let σ_1 and σ_2 be any two adjacent data streams of length T, for some time bound $T > 0$. For the execution of $\mathsf{PMG}(\epsilon, \lambda)$ on σ_1, we denote by $f_1^t(x)$ the value of the counter for item x at the end of iteration t in the for loop between Lines 3 and 10; we can view $f_1^t \in \mathbb{Z}^{\mathcal{U}}$ as a vector. Define $f_2^t \in \mathbb{Z}^{\mathcal{U}}$ similarly for the stream σ_2. The following claim states that for all time steps t, the vectors f_1^t and f_2^t follow a structural property. Its proof appears in the appendix.

Claim 1 (Structural Property between Adjacent Streams) *Let t_0 be the time step such that $\sigma_1(t_0) \neq \sigma_2(t_0)$ (it is obvious that for $t < t_0$, $f_1^t = f_2^t$). Then, for each $t \geq t_0$, the vectors f_1^t and f_2^t satisfy one of the following four structural states.*
1. $\boldsymbol{\mathcal{I}}$: $f_1^t = f_2^t$.
2. $\boldsymbol{\mathcal{S}(x_1, x_2)}$, for some $x_1 \neq x_2 \in \mathcal{U}$: $f_1^t(x_1) = f_2^t(x_1) + 1$ and $f_2^t(x_2) = f_1^t(x_2) + 1$. For $x \in \mathcal{U} \setminus \{x_1, x_2\}$, $f_1^t(x) = f_2^t(x)$.
3. $\boldsymbol{\mathcal{N}_1(x_1, X)}$, for some $x_1 \in X \subset \mathcal{U}$ and $|X| = \beta$: $f_1^t(x_1) = f_2^t(x_1) + 2$; for $x \in X \setminus \{x_1\}$, $f_1^t(x) = f_2^t(x) + 1$; for $x \in \mathcal{U} \setminus X$, $f_1^t(x) = f_2^t(x) = 0$.
4. $\boldsymbol{\mathcal{N}_2(x_2, X)}$, for some $x_2 \in X \subset \mathcal{U}$ and $|X| = \beta$: $f_2^t(x_2) = f_1^t(x_2) + 2$; for $x \in X \setminus \{x_2\}$, $f_2^t(x) = f_1^t(x) + 1$; for $x \in \mathcal{U} \setminus X$, $f_1^t(x) = f_2^t(x) = 0$.
In particular, for each of the four states, we have $||f_1^t - f_2^t||_1 \leq \beta + 1$.

Hence, from Claim 1, we have $||f_1^T - f_2^T||_1 \leq \beta + 1$, as required. □

Proof of Claim 1:

In this proof, we use $f_1(x)$ to denote the counter for item x during the execution of $\mathsf{PMG}(\epsilon, \lambda)$ on σ_1, and define f_2 similarly for σ_2. Note that $f_1(x)$ and $f_2(x)$ might change during the execution of $\mathsf{PMG}(\epsilon, \lambda)$. When we want to specify the values of the counters at specific times, we will use the superscripts.

We use induction on $t - t_0$.

Base case. Suppose $t = t_0$. Note that we have $f_1^{t_0-1}(x) = f_2^{t_0-1}(x)$ for all $x \in \mathcal{U}$. Let $A := \{x \in \mathcal{U} : f_1^{t_0-1}(x) > 0\}$, then A is also the set $\{x \in \mathcal{U} : f_2^{t_0-1}(x) > 0\}$. Suppose $\sigma_1(t_0) = x_1 \neq x_2 = \sigma_2(t_0)$.

According to the behavior of the algorithm, we have the following cases to consider.

1. $\underline{|A| < \beta \text{ or } \{x_1, x_2\} \subset A.}$ In time step t_0, both $f_1(x_1)$ and $f_2(x_2)$ increase by one, and so we have $f_1^{t_0}(x_1) = f_1^{t_0-1}(x_1) + 1$ and $f_2^{t_0}(x_1) = f_2^{t_0-1}(x_1) + 1$. Hence, at the end of time step t_0, PMG is in state $\mathcal{S}(x_1, x_2)$.
2. $\underline{x_1 \notin A \text{ and } x_2 \notin A \text{ and } |A| = \beta.}$ During time step t_0, for all $x \in A$, both $f_1(x)$ and $f_2(x)$ decrease by one. Also, both $f_1(x_1)$ and $f_2(x_2)$ firstly increase by one and then decrease by one, and thus remain 0's. Hence, at the end of time step t_0, PMG is in state $\mathcal{I}$.
3. $\underline{x_1 \in A \text{ and } x_2 \notin A \text{ and } |A| = \beta.}$ During time step t_0, $f_1(x_1)$ increases by one, and for all $x \in A$, $f_2(x)$ decreases by one. Also, $f_2(x_2)$ firstly increases by one and then decreases by one, and thus remains 0. Hence, at the end of time step t_0, PMG is in state $\mathcal{N}_1(x_1, A)$.
4. $\underline{x_2 \in A \text{ and } x_1 \notin A \text{ and } |A| = \beta.}$ Symmetric to the above case, at the end of time step t_0, PMG is in state $\mathcal{N}_2(x_2, A)$.

Induction step. Suppose $t > t_0$, and it is true that at the end of time step $t - 1$, PMG can only be in the four structural states.

Suppose $\sigma_1(t) = \sigma_2(t) = y$. Let $A_1 := \{x \in \mathcal{U} : f_1^{t-1}(x) > 0\}$ and $A_2 := \{x \in \mathcal{U} : f_2^{t-1}(x) > 0\}$. Note that for any $i \in \{1, 2\}$, $A_i \cup \{y\}$ is the set of items x such that $f_i(x) > 0$ after increasing $f_i(y)$ at Line 4 in the t-th iteration; hence, the decreasing step is excecuted if and only if $|A_i \cup \{y\}| > \beta$.

According to the state in which PMG is at the end of time step $t-1$, we have several cases to consider.

PMG is in $\mathcal{I}$. During time step t, for any $x \in \mathcal{U}$, $f_1(x)$ and $f_2(x)$ increase and decrease simultaneously. Thus, at the end of time step t, $f_1(x) = f_2(x)$ for all $x \in \mathcal{U}$. Hence, PMG is still in $\mathcal{I}$.

PMG is in $\mathcal{S}(x_1, x_2)$. Let $A := \{x \in \mathcal{U} \setminus \{x_1, x_2\} : f_1^{t-1}(x) > 0\}$. Observe that A is also the set $\{x \in \mathcal{U} \setminus \{x_1, x_2\} : f_2^{t-1}(x) > 0\}$. According to the behavior of the algorithm, we have the following subcases.

1. $\underline{|A_1 \cup \{y\}| \leq \beta \text{ and } |A_2 \cup \{y\}| \leq \beta.}$ During time step t, both $f_1(y)$ and $f_2(y)$ increase by one. Hence, at the end of time step t, PMG is still in $\mathcal{S}(x_1, x_2)$.
2. $\underline{|A_1 \cup \{y\}| \leq \beta \text{ and } |A_2 \cup \{y\}| > \beta.}$ It is easy to see that only the following two situations may lead to this subcase.
   (a) $|A| = \beta - 1$ and $f_1(x_2) = 0$ and $f_2(x_1) = 0$ and $y = x_1$. During time step t, $f_1(x_1)$ increases by one, and $f_2(x)$ decreases by one for all $x \in A \cup \{x_2\}$. Also, $f_2(x_1)$ firstly increases by one and then decreases by one, and thus remains 0. Hence, at the end of time step t, PMG is in state $\mathcal{N}_1(x_1, A \cup \{x_1\})$.
   (b) $|A| = \beta - 2$ and $f_1(x_2) = 0$ and $f_2(x_1) > 0$ and $y \notin A \cup \{x_1, x_2\}$. During time step t, $f_1(y)$ increases by one, and $f_2(x)$ decreases by one for all $x \in A \cup \{x_1, x_2\}$. Also, $f_2(y)$ firstly increases by one and then decreases by one, and thus remains 0. Note that $f_1(x_2) = 0$ implies $f_2(x_2) = 1$. Therefore,after the decreasing step, $f_2(x_2) = 0$. Hence, at the end of time step t, PMG is in state $\mathcal{N}_1(x_1, A \cup \{x_1, y\})$.
3. $\underline{|A_1 \cup \{y\}| > \beta \text{ and } |A_2 \cup \{y\}| \leq \beta.}$ Symmetric to the above subcase, at the end of time step t, PMG is in either $\mathcal{N}_2(x_2, A \cup \{x_2\})$ or $\mathcal{N}_2(x_2, A \cup \{x_2, y\})$.
4. $\underline{|A_1 \cup \{y\}| > \beta \text{ and } |A_2 \cup \{y\}| > \beta.}$ It is easy to see that only the following two situations may lead to this subcase.
   (a) $|A| = \beta - 1$ and $f_1(x_2) = 0$ and $f_2(x_1) = 0$ and $y \notin A \cup \{x_1, x_2\}$. During time step t, $f_1(y)$ and $f_2(y)$ firstly increase by one and then decrease by one, and thus remain 0's. Also, for all $x \in A$, $f_1(x)$ and $f_2(x)$ decrease by one. And, $f_1(x_1)$ and $f_2(x_2)$ decrease by one, and hence are both 0's at the end of time step t. Therefore, PMG is in state $\mathcal{I}$ at the end of time t.
   (b) $|A| = \beta - 2$ and $f_1(x_2) > 0$ and $f_2(x_1) > 0$ and $y \notin A \cup \{x_1, x_2\}$. During time step t, $f_1(x)$ and $f_2(x)$ decrease by one for all $x \in A \cup \{x_1, x_2\}$. Also, $f_1(y)$ and $f_2(y)$ firstly increase by one and then decrease by one, and thus remain 0's. Hence, at the end of time step t, PMG is still in $\mathcal{S}(x_1, x_2)$.

PMG is in $\mathcal{N}_1(x_1, X)$. Still, according to the behavior of PMG, we have several cases to consider.

1. $\underline{y \in X.}$ During time step t, both $f_1(y)$ and $f_2(y)$ increase by one. Hence, at the end time t, PMG is still in $\mathcal{N}_1(x_1, X)$.
2. $\underline{y \notin X \text{ and } |A_2 \cup \{y\}| \leq \beta.}$ Note that by the definition of state $\mathcal{N}_1(x_1, X)$, it holds that $|X| = \beta$, and thus $|X \cup \{y\}| > \beta$. Also, for all $x \in X$, $f_1^{t-1}(x) > 0$. Hence, during time step t, $f_1(x)$ decreases by one for all $x \in X$, and $f_1(y)$ firstly increases by one and then decreases by one, and hence remains 0. And, since $|A_2 \cup \{y\}| \leq \beta$, $f_2(y)$ increases by one during time t. Therefore, at the end of time step t, PMG is in $\mathcal{S}(x_1, y)$.

3. $\underline{y \notin X \text{ and } |A_2 \cup \{y\}| > \beta.}$ Note that $|A_2 \cup \{y\}| > \beta$ implies that $|A_2| = \beta$, which means $X = A_2$. Hence, during time step t, both $f_1(x)$ and $f_2(x)$ decrease by one, for all $x \in X$. Also, both $f_1(y)$ and $f_2(y)$ firstly increase by one and then decrease by one, and hence remain 0's. Therefore, at the end of time step t, PMG is still in $\mathcal{N}_1(x_1, X)$.

PMG is in $\boldsymbol{\mathcal{N}_2(x_2, X)}$. Symmetric to the above case, at the end of time step t, PMG can only be in state $\mathcal{N}_2(x_2, X)$ or $\mathcal{S}(y, x_2)$. □

Utility. Let $\mathcal{W} := \sigma([T])$ be the multiset of items in stream σ. We show that provided that T is large enough, with high probability, the algorithm $\mathsf{PMG}(\epsilon, \lambda)$ in Algorithm 1 outputs a vector $\widehat{f} \in \mathbb{Z}^{\mathcal{U}}$ such that for every item x, $\widehat{f}$ approximates $\mathsf{count}_x(\mathcal{W})$ with additive error at most λT. There are two sources of error in $\widehat{f}$: the first source comes from counters being decreased in the loop between loop between Lines 3 and 10; the second source comes from the randomness and the rounding between Lines 12 and 17.

Lemma 5 (Error for f). *In Algorithm 1 after Line 10, consider the counters at this point as a vector $f \in \mathbb{Z}^{\mathcal{U}}$. Let $\mathcal{W}$ be the multiset containing the items in the stream $\sigma \in \mathcal{U}^T$ of length T. Then, for each $x \in \mathcal{U}$, $|f(x) - \mathsf{count}_x(\mathcal{W})| \leq \frac{\lambda}{2} \cdot T$.*

Proof. Observe that in the for loop between Lines 3 and 10, if counters are decreased in an iteration (which we call an *decrease iteration*), then the counters for exactly $\beta + 1$ different items are decreased by 1. Hence, it follows that there can be at most $\frac{T}{\beta+1} \leq \frac{T}{2/\lambda+1} < \frac{\lambda}{2} \cdot T$ decrease iterations.

For any item $x \in \mathcal{U}$, the counter for x increases by one whenever one copy of x comes into the stream, and could only possibly decrease by one in a decrease iteration. Hence, it follows that $|f(x) - \mathsf{count}_x(\mathcal{W})| \leq \frac{\lambda}{2} \cdot T$. □

Lemma 6 (Error for $\widehat{f}$). *Let $f \in \mathbb{Z}^{\mathcal{U}}$ be as in Lemma 5 and $\widehat{f} \in \mathbb{Z}^{\mathcal{U}}$ be the vector in the output of* $\mathsf{PMG}(\epsilon, \lambda)$. *Suppose that for all $x \in \mathcal{U}$, the randomness r_x generated in Line 12 satisfies $|r_x| \leq \frac{\lambda}{4} \cdot T$. Then, for all $x \in \mathcal{U}$, $|\widehat{f}(x) - f(x)| \leq \frac{\lambda}{2} \cdot T$.*

Proof. We show the result by case analysis. Observe that for any item x, $\widehat{f}(x)$ is either $f(x) + r_x$ or 0.

Case 1: $\widehat{f}(x) = f(x) + r_x$. We have $|\widehat{f}(x) - f(x)| = |r_x| \leq \frac{\lambda T}{4} \leq \frac{\lambda T}{2}$.

Case 2: $\widehat{f}(x) = 0$ and $f(x) + r_x \leq 0$. We have $|\widehat{f}(x) - f(x)| \leq |r_x| \leq \frac{\lambda T}{4} \leq \frac{\lambda T}{2}$.

Case 3: $\widehat{f}(x) = 0$ and $f(x) + r_x > 0$. Since there are at most β non-zero coordinates in f, we conclude that there must be some item y such that $f(y) = 0$ and $f(y) + r_y \geq f(x) + r_x$. It follows that $f(x) \leq r_y - r_x \leq |r_y| + |r_x| \leq \frac{\lambda}{2} \cdot T$. Hence, $|\widehat{f}(x) - f(x)| = |0 - f(x)| \leq \frac{\lambda}{2} \cdot T$. □

Lemma 7 (Usefulness of $\mathsf{PMG}(\epsilon, \lambda)$). *Let $0 < \delta < 1$ and $T \geq \frac{32}{\lambda^2 \epsilon} \log \frac{n}{\delta}$. Suppose $\widehat{f} \in \mathbb{Z}^{\mathcal{U}}$ is the output of* $\mathsf{PMG}(\epsilon, \lambda)$ *on an input stream $\sigma \in \mathcal{U}^T$ of length T. Then, $\widehat{f}$ is $(\lambda T, \delta)$-useful with respect to the multiset $\sigma([T])$ of items in σ.*

Proof. By Fact 1, we know that for any $x \in \mathcal{U}$, $\Pr[|r_x| > \frac{\lambda}{4} \cdot T] = \Pr[|r_x| > \lfloor \frac{\lambda}{4} \cdot T \rfloor] \leq \exp(-\frac{\epsilon}{\beta+1} \cdot \lfloor \frac{\lambda}{4} \cdot T \rfloor)$, which is at most $\frac{\delta}{n}$ when $T \geq \frac{32}{\lambda^2 \epsilon} \log \frac{n}{\delta}$. By union bound on the items in $\mathcal{U}$, we conclude that with probability at least $1 - \delta$, for all $x \in \mathcal{U}$, $|r_x| \leq \frac{\lambda}{4} \cdot T$. From Lemmas 6 and 5, we can conclude the result immediately. □

B Proofs for PCC

Recall that at any point of time t, there exists a collection $\mathcal{C}_t$ of disjoint active blocks, such that $\mathcal{C}_t$ contains at most two blocks from each level, and that the union of blocks in $\mathcal{C}_t$ is the union of all active level-0 blocks. At any time, $\mathsf{PCC}(\epsilon, \lambda)$ maintains the dictionary $\mathcal{P}_t$, where $\mathcal{P}_t = \{(x, \sum_{\mathcal{B} \in \mathcal{C}_t} \widehat{f}_{\mathcal{B}}(x)) : x \in \mathcal{U}$ and $\sum_{\mathcal{B} \in \mathcal{C}_t} \widehat{f}_{\mathcal{B}}(x) > 0\}$.
Small $|\mathcal{P}_t|$. Let t be any fixed time step. Note that if $\mathcal{P}_t$ contains a pair of the form (x, c_x), then there exists at least one $\mathcal{B} \in \mathcal{C}_t$ such that $\widehat{f}_{\mathcal{B}}(x) > 0$. Since $\mathcal{C}_t$ contains at most two blocks at each level, and $\widehat{f}_{\mathcal{B}}$ for a block $\mathcal{B}$ at level i has at most $O(\frac{1}{\lambda_i})$ non-zero coordinates, it follows that $|\mathcal{P}_t| \le O(\sum_{i=0}^{\ell} 2 \cdot \frac{1}{\lambda_i}) = O(\frac{1}{\lambda} \log \frac{1}{\lambda})$.

Lemma 8 (Running Time). *The mechanism* $\mathsf{PCC}(\epsilon, \lambda)$ *takes* $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ *operations to process each item in the stream, and samples at most* $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ *random variables in each time step. Moreover, it takes* $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ *operations to produce* $\mathcal{P}_t$ *from the active blocks.*

Proof. Observe that at any time step, at most one block from each level is under construction. Hence, from Lemma 2, the total number operations to process each item is at most $\sum_{i=0}^{l} O(\frac{1}{\lambda_i}) = O(\frac{1}{\lambda} \log \frac{1}{\lambda})$, and the number of random variables sampled in each time step is at most $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$.

Next we analyze the number of operations to produce $\mathcal{P}_t$. Note that $\widehat{f}_{\mathcal{B}}$ may also be returned by PMG in the form of a dictionary. Hence, we can look at $\widehat{f}_{\mathcal{B}}$ for every $\mathcal{B} \in \mathcal{C}_t$ to add items into $\mathcal{P}_t$. Still, we have to check $\widehat{f}_{\mathcal{B}}$ for at most two level-i blocks $\mathcal{B}$, and checking one such $\widehat{f}_{\mathcal{B}}$ needs $O(\frac{1}{\lambda_i})$ operations. Thus, $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ operations are needed in total.

Lemma 9 (Memory Usage). *The mechanism* $\mathsf{PCC}(\epsilon, \lambda)$ *can be implemented using* $O(\frac{1}{\lambda} \log^2 \frac{1}{\lambda})$ *words of memory.*

Proof. At any time t, there are at most $\frac{W}{W_0 \cdot 2^i}$ active blocks, and at most one under-construction block at each level i. Since each active or under-construction block at level i uses $O(\frac{1}{\lambda_i})$ words of memory, the total memory usage is $O(\sum_{i=0}^{\ell} \frac{W}{W_0 \cdot 2^i} \frac{1}{\lambda_i}) = O(\sum_{i=0}^{\ell} \frac{i}{\lambda}) = O(\frac{\ell^2}{\lambda}) = O(\frac{1}{\lambda} \log^2 \frac{1}{\lambda})$ □

Utility. Let $\mathcal{W}_t := \mathcal{W}_t(\sigma)$. We show that provided W is large enough, with high probability, the dictionary maintained by $\mathsf{PCC}(\epsilon, \lambda)$ can be used to give approximate counts.

Lemma 10 (Error for $\mathcal{P}_t(x)$). *Let t be any fixed time step. Suppose for every level-i block* $\mathcal{B} \in \mathcal{C}_t$ *and every* $x \in \mathcal{U}$, $|\widehat{f}_{\mathcal{B}}(x) - \mathsf{count}_x(\mathcal{B})| \le \lambda_i W_i$. *Then, for all* $x \in \mathcal{U}$, $|\mathcal{P}_t(x) - \mathsf{count}_x(\mathcal{W}_t)| \le \lambda W$.

Proof. Recall that $\cup_{\mathcal{B} \in \mathcal{C}_t} \mathcal{B}$ is the union of all level-0 active blocks at time t. Hence, all items in $\mathcal{W}_t \setminus \cup_{\mathcal{B} \in \mathcal{C}_t} \mathcal{B}$ are contained in at most two level-0 blocks (one under-construction and one expired). Thus, for every $x \in \mathcal{U}$, $|\sum_{\mathcal{B} \in \mathcal{C}_t} \mathsf{count}_x(\mathcal{B}) - \mathsf{count}_x(\mathcal{W}_t)| \le 2W_0 \le \frac{1}{2} \cdot \lambda W$.

Also recall that for any $0 \le i \le \ell$, $\mathcal{C}_t$ contains at most two level-i blocks. Hence, for all $x \in \mathcal{U}$, $|\sum_{\mathcal{B}\in\mathcal{C}_t} \widehat{f}_{\mathcal{B}}(x) - \sum_{\mathcal{B}\in\mathcal{C}_t} \mathsf{count}_x(\mathcal{B})| \le \sum_{i=0}^{\ell} 2\lambda_i W_i \le \frac{1}{2}\lambda W$.

Observe that for any $x \in \mathcal{U}$, $\mathcal{P}_t(x) = \sum_{\mathcal{B}\in\mathcal{C}_t} \widehat{f}_{\mathcal{B}}(x)$. Hence, it follows that for any $x \in \mathcal{U}$, $|\mathcal{P}_t(x) - \mathsf{count}_x(\mathcal{W}_t)| \le |\sum_{\mathcal{B}\in\mathcal{C}_t} \widehat{f}_{\mathcal{B}}(x) - \sum_{\mathcal{B}\in\mathcal{C}_t} \mathsf{count}_x(\mathcal{B})| + |\sum_{\mathcal{B}\in\mathcal{C}_t} \mathsf{count}_x(\mathcal{B}) - \mathsf{count}_x(\mathcal{W}_t)| \le \lambda W$ □

Lemma 11 (Usefulness of $\mathsf{PCC}(\epsilon, \lambda)$). *Suppose* $0 < \delta < 1$ *and* $L > 0$.

(1) If $W \ge \Theta(\frac{1}{\lambda^2\epsilon}(\log^2 \frac{1}{\lambda}) \cdot \log(\frac{n}{\delta} \log \frac{1}{\lambda}))$. *Then, at every time step* $t \in \mathbb{N}$, $\mathcal{P}_t$ *is* $(\lambda W, \delta)$*-useful with respect to* $\mathcal{W}_t$.
(2) If $W \ge \Theta(\frac{1}{\lambda^2\epsilon}(\log^2 \frac{1}{\lambda}) \cdot \log(\frac{L+W}{\lambda W} \cdot \frac{n}{\delta}))$*, then for any* $T \ge L+1$, $\{\mathcal{P}_t\}_{t\in[T-L,T]}$ *is simultaneously* $(\lambda W, \delta)$*-useful with respect to* $\{\mathcal{W}_t\}_{t\in[T-L,T]}$.

Proof. (1) Let $t \in \mathbb{N}$ be any fixed time step and suppose $W \ge \frac{2048\cdot(\ell+1)^2}{\lambda^2\epsilon} \log \frac{2(\ell+1)n}{\delta} = \Theta(\frac{1}{\lambda^2\epsilon}(\log^2 \frac{1}{\lambda}) \cdot \log(\frac{n}{\delta} \log \frac{1}{\lambda}))$. Then, for any $0 \le i \le \ell$, $W_i = \lfloor \frac{\lambda W}{4} \rfloor \cdot 2^i \ge \frac{32}{\lambda_i^2 \epsilon_i} \log(\frac{2(\ell+1)n}{\delta})$. By Lemma 1, for every level-i block $\mathcal{B} \in \mathcal{C}_t$, $\widehat{f}_{\mathcal{B}}$ is $(\lambda_i W_i, \frac{\delta}{2(\ell+1)})$-useful with respect to $\mathcal{B}$. Recall that $\mathcal{C}_t$ contains at most two level-i blocks, and hence $|\mathcal{C}_t| \le 2(\ell+1)$. By union bound on the blocks in $\mathcal{C}_t$, we know that with probability at least $1-\delta$, for all level-i block $\mathcal{B} \in \mathcal{C}_t$ and all $x \in \mathcal{U}$, $|\widehat{f}_{\mathcal{B}}(x) - \mathsf{count}_x(\mathcal{B})| \le \lambda_i W_i$. By Lemma 10, we can immediately conclude that $\mathcal{P}_t$ is $(\lambda W, \delta)$-useful with respect to $\mathcal{W}_t$.

(2) Let $T \ge L+1$ be any fixed time step. Let $\mathcal{S} = \{\mathcal{B}_j^i : T-(L+W)+1 \le (j-1)W_i + 1 \le jW_i \le T\}$ be the set of all blocks that are totally within $\sigma([T-(L+W)+1, T])$. It is easy to see that $|\mathcal{S}| \le \sum_{i=0}^{\ell} \frac{L+W}{W_0 \cdot 2^i} \le \frac{16(L+W)}{\lambda W}$.

Suppose $W \ge \frac{2048\cdot(\ell+1)^2}{\lambda^2\epsilon} \log(\frac{16(L+W)}{\lambda W} \cdot \frac{n}{\delta}) = \Theta(\frac{1}{\lambda^2\epsilon}(\log^2 \frac{1}{\lambda}) \cdot \log(\frac{L+W}{\lambda W} \cdot \frac{n}{\delta}))$. Then, for any $0 \le i \le \ell$, $W_i = \lfloor \frac{\lambda W}{4} \rfloor \cdot 2^i \ge \frac{32}{\lambda_i^2 \epsilon_i} \log(\frac{16(L+W)}{\lambda W} \cdot \frac{n}{\delta})$. By Lemma 1, for every level-i block $\mathcal{B} \in \mathcal{S}$, $\widehat{f}_{\mathcal{B}}$ is $(\lambda_i W_i, \frac{\lambda W \delta}{16(L+W)})$-useful with respect to $\mathcal{B}$. Recall that $|\mathcal{S}| \le \frac{16(L+W)}{\lambda W}$. Hence, by union bound on all blocks in $\mathcal{S}$, we know that with probability at least $1-\delta$, for all level-i block $\mathcal{B} \in \mathcal{S}$, and all $x \in \mathcal{U}$, $|\widehat{f}_{\mathcal{B}}(x) - \mathsf{count}_x(\mathcal{B})| \le \lambda_i W_i$. Observe that $\cup_{t\in[T-L,T]} \mathcal{C}_t \subseteq \mathcal{S}$. Hence, by Lemma 10, with probability at least $1-\delta$, for all $t \in [T-L, T]$ and all $x \in \mathcal{U}$, $|\mathcal{P}_t(x) - \mathsf{count}_x(\mathcal{W}_t)| \le \lambda W$. □

C Proofs for PDCH-LU

Space and Time Requirement at Each Node. Note that for each $x \in \mathcal{U}$, if $\mathcal{P}_t(x) = 0$, then at the end of time step t, $\mathsf{Last}(x)$ must be 0. Hence, there are at most $|\mathcal{P}_t| = O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ items x with $\mathsf{Last}(x) > 0$ at the end of each time step t. Those non-zero values of $\mathsf{Last}(x)$ are the information we need to explicitly store. Together with the $O(\frac{1}{\lambda} \log^2 \frac{1}{\lambda})$ words of memory used by $\mathsf{PCC}(\epsilon, \frac{\lambda}{11})$, each node uses $O(\frac{1}{\lambda} \log^2 \frac{1}{\lambda})$ words of memory. Also, at each time t, an item x is checked by PDCH-LU only if $\mathsf{Last}(x) > 0$ in the beginning of time t or $\mathcal{P}_t(x) > 0$, and there are $O(\frac{1}{\lambda} \log \frac{1}{\lambda})$ such items. Together with the operations of $\mathsf{PCC}(\epsilon, \frac{\lambda}{11})$, totally $O(\frac{1}{\lambda}\log\frac{1}{\lambda})$ operations are needed to process an item.

Privacy Guarantee. Note that the transcript of node i, $\Pi(\sigma_i)$, is a deterministic function of $(\mathcal{P}_t)_{t\in\mathbb{N}}$, the dictionaries maintained by $\mathsf{PCC}(\epsilon, \frac{\lambda}{11})$ on σ_i. Since $(\mathcal{P}_t)_{t\in\mathbb{N}}$ maintains ϵ-differential privacy, it follows that the transcript of each node is ϵ-differentially private against the aggregator.

Utility Suppose $0 < \delta < 1$ and let $t \geq W$ be any fixed time step. We use $\mathsf{Last}_t(x)$ to denote $\mathsf{Last}(x)$'s value at the end of time step t. And let $\mathcal{P}_t^i$ and $\mathsf{Last}_t^i(x)$ be $\mathcal{P}_t$ and $\mathsf{Last}_t(x)$ at node i, respectively. From the description of $\mathsf{PDCH\text{-}LU}$, it is easy to see that for any $i \in [k]$, $|\mathcal{P}_t^i(x) - \mathsf{Last}_t^i(x)| \leq \frac{9}{11}\cdot\lambda W$. By Lemma 3 and union bound on the k streams, we know that if $W \geq \Theta(\frac{1}{\lambda^2\epsilon}(\log^2\frac{1}{\lambda})\cdot\log(\frac{kn}{\delta}\log\frac{1}{\lambda}))$, then with probability at least $1-\delta$, for any $i \in [k]$ and any $x \in \mathcal{U}$, $|\mathcal{P}_t^i(x) - \mathsf{count}_x(\mathcal{W}_t^{(i)})| \leq \frac{\lambda}{11}W$, and hence $\mathsf{Last}_t^i(x)$ is a λ-approximate count for x with respect to $\mathcal{W}_t^{(i)}$. Note that at the end of any time step t, $\mathsf{Last}(x)$ at node i is always identical to the counter $c_i(x)$ maintained by the aggregator. Hence, we conclude that if $W \geq \Theta(\frac{1}{\lambda^2\epsilon}(\log^2\frac{1}{\lambda})\cdot\log(\frac{kn}{\delta}\log\frac{1}{\lambda}))$, then at the end of any time step $t \geq W$, with probability at least $1-\delta$, for all $x \in \mathcal{U}$, the count $c(x)$ on the server is a λ-approximate count for x with respect to $\mathcal{W}_t^{[k]}$.

Communication Cost Suppose $L > 0$ and $0 < \delta < 1$. By Lemma 3 and union bound on the k streams, if $W \geq \Theta(\frac{1}{\lambda^2\epsilon}(\log^2\frac{1}{\lambda})\log(\frac{L+W}{\lambda W}\cdot\frac{kn}{\delta}))$, then for any $T \geq L+1$, with probability at least $1-\delta$, for any $i \in [k]$, any $t \in [T-L, T]$ and any $x \in \mathcal{U}$, $|\mathcal{P}_t^i(x) - \mathsf{count}_x(\mathcal{W}_t^{(i)})| \leq \frac{\lambda}{11}\cdot W$. By a characteristic-set argument similar to that in [3], this approximation guarantee implies that for any $T \geq L$, with probability at least $1-\delta$, the total communication cost in time interval $[T-L+1, T]$ is $O(\frac{k}{\lambda}\cdot\lceil\frac{L}{W}\rceil\log W)$ words.

D Generating Largest r among n Independently and Identically Distributed Random Variables

Given some distribution $\mathcal{D}$ on integers and $1 \leq r \leq n$, we wish to sample r (dependent) random variables that have the joint distribution of the r largest random variables among n independently and identically distributed random variables each having distribution $\mathcal{D}$. The naive way to achieve this is to sample n independent random variables from $\mathcal{D}$, sort them and output the r largest; however, this would take $\Omega(n)$ memory and time. We would like a sampling method that takes $O(r)$ memory and time independent of n.

Computational Model and Source of Randomness. We assume arithmetic such as exponentiation can be performed in $O(1)$ time, even when the exponent is n. Given distribution $\mathcal{D}$, we assume the probability mass function $k \mapsto p_k(\mathcal{D})$ and the cumulative distribution function $S_k(\mathcal{D}) := \sum_{i\leq k} p_i(\mathcal{D})$ can be retrieved in $O(1)$ time for each k. We assume that we can sample independent real numbers from the interval $[0, 1]$.

Notation. When there is no risk of ambiguity, we write $p_k := p_k(\mathcal{D})$ and $S_k := S_k(\mathcal{D})$ dropping the dependence on $\mathcal{D}$. We assume $X_1, X_2, \ldots, X_n$ are n independent random variables, each having distribution $\mathcal{D}$. For $1 \leq l \leq n$, let Z_l be the lth largest among the n random variables. We shall describe a procedure $\mathsf{SAMPLE}(r, n, \mathcal{D})$ that prints r numbers having the joint distribution $(Z_1, Z_2, \ldots, Z_r)$.

Input: An integer n and a distribution $\mathcal{D}$.
Output: An integer k having the distribution as the largest among n independent random variables each sampled from $\mathcal{D}$.

```
1 Sample $x \in [0,1]$ uniformly at random;
2 Find $k$ such that $S^n_{k-1} < x \leq S^n_k$;
3 Return $k$;
```

Algorithm 3: $\mathsf{MAX}(n, \mathcal{D})$

For each integer k, we denote by $\mathcal{D}_k$ the conditional distribution having probability mass function $i \mapsto \frac{p_i(\mathcal{D})}{S_k(\mathcal{D})}$, for $i \leq k$.

$\mathsf{MAX}(n, \mathcal{D})$**: Sampling the Largest.** Observe that $\Pr[Z_1 = k] = S^n_k - S^n_{k-1}$. Hence, the procedure $\mathsf{MAX}(n, D)$ that returns an integer having the same distribution as Z_1 is defined in Algorithm 3.

Conditioning on Z_1. Define $q_l := \Pr[Z_1 = k, Z_2 = k, \ldots, Z_l = k] = S^n_k - \sum_{i=0}^{l-1} \binom{n}{i} p^i_k S^{n-i}_{k-1}$. Then, we can compute $\Pr[Z_l = k | Z_1 = k, \ldots, Z_{l-1} = k] = \frac{q_l}{q_{l-1}}$. This means conditioning on $Z_1 = k$, we can sample $Z_2, Z_3, \ldots$ until we encounter the first l such that $Z_{l+1} \neq k$, at which point we know the remaining $n - l$ random variables each has distribution $\mathcal{D}_{k-1}$, and hence we can call $\mathsf{SAMPLE}(r - l, n - l, \mathcal{D}_{k-1})$ to print the remaining $r - l$ random variables. The details of $\mathsf{SAMPLE}(r, n, \mathcal{D})$ are given in Algorithm 4.

```
Input: Integers $1 \leq r \leq n$ and distribution $\mathcal{D}$.
Output: Print $r$ numbers having the joint distribution as the $r$ largest among $n$
        independent random variables each sampled from $\mathcal{D}$.
1  $k \leftarrow \mathsf{MAX}(n, \mathcal{D})$;
2  $\mathsf{Print}(k)$;
3  $q_1 \leftarrow S_k^n - S_{k-1}^n$;
4  $l \leftarrow 1$;
5  while $l < r$ do
6      $q_{l+1} \leftarrow q_l - \binom{n}{l} p_k^l S_{k-1}^{n-l}$;
7      Sample $x \in [0, 1]$ uniformly at random;
8      if $x \leq \frac{q_{l+1}}{q_l}$ then
           // $Z_{l+1} = k$
9          $\mathsf{Print}(k)$;
10         $l \leftarrow l + 1$;
11     end
12     else
           // $Z_{l+1} < k$, break while loop
13         break;
14     end
15 end
16 if $l < r$ then
17     $\mathsf{SAMPLE}(r - l, n - l, \mathcal{D}_{k-1})$;
18 end
```

Algorithm 4: $\mathsf{SAMPLE}(r, n, \mathcal{D})$