# Small Synopses for Group-By Query Verification on Outsourced Data Streams

KE YI
Hong Kong University of Science and Technology
FEIFEI LI
Florida State University
GRAHAM CORMODE and MARIOS HADJIELEFTHERIOU
AT&T Labs - Research
GEORGE KOLLIOS
Boston University
and
DIVESH SRIVASTAVA
AT&T Labs - Research

Due to the overwhelming flow of information in many data stream applications, data outsourcing is a natural and effective paradigm for individual businesses to address the issue of scale. In the standard data outsourcing model, the data owner outsources streaming data to one or more third-party servers, which answer queries posed by a potentially large number of clients on the data owner's behalf. Data outsourcing intrinsically raises issues of trust, making outsourced query assurance on data streams a problem with important practical implications. Existing solutions proposed in this model all build upon cryptographic primitives such as signatures and collision-resistant hash functions, which only work for certain types of queries, e.g., simple selection/aggregation queries.

In this paper, we consider another common type of queries, namely, "`GROUP BY, SUM`" queries, which previous techniques fail to support. Our new solutions are not based on cryptographic primitives, but instead use algebraic and probabilistic techniques to compute a small synopsis on the true query result, which is then communicated to the client so as to verify the correctness of the query result returned by the server. The synopsis uses a constant amount of space irrespective of the result size, has an extremely small probability of failure, and can be maintained using no extra space when the query result changes as elements stream by. We then generalize our synopsis to allow some tolerance on the number of erroneous groups, in order to support semantic load shedding on the server. When the number of erroneous groups is indeed tolerable, the synopsis can be strengthened so that we can locate and even correct these errors. Finally, we implement our techniques and perform an empirical evaluation using live network traffic.

Categories and Subject Descriptors: F.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; H.2 [**Database Management**]: Security and integrity

General Terms: Algorithms, Security, Verification

Additional Key Words and Phrases: Synopses, Data streams, Outsourcing

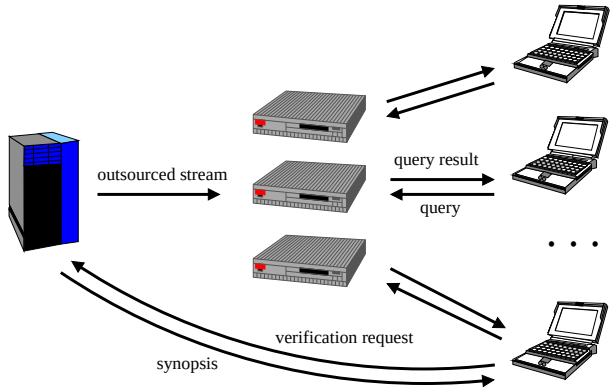



Fig. 1. System architecture.

## 1. INTRODUCTION

A large number of industrial and academic Data Stream Management Systems (DSMS) have been developed recently [Chandrasekaran et al. 2003; Hammad et al. 2004; Arasu et al. 2003; Cranor et al. 2003; Carney et al. 2003; Abadi et al. 2005]. The need for such DSMSs is mainly driven by the continuous nature of the data being generated by a variety of real-world applications, like telephony and networking. Providing fast and reliable querying services on the streaming data to clients is central to many businesses. However, due to the overwhelming data flow observed in most data streams, many companies do not possess the necessary resources for deploying a DSMS, and are not willing to acquire them. Hence, in these cases outsourcing the data stream and the desired computations to a third-party server is the only alternative. Outsourcing also solves the issue of scale: as there are more clients, the data owner can simply employ more mirroring servers. In addition, this can often lead to faster query responses, since these servers can be closer to the clients than a single centralized server. However, data outsourcing and remote computations intrinsically raise issues of trust. As a consequence, outsourced query verification on data streams is a problem with important practical implications.

Consider a setting where the data owner (e.g., a stock broker) with limited resources, such as memory and bandwidth, outsources its data stream to one or more remote, untrusted servers (that can be compromised, malicious, running faulty software, etc.). Clients register continuous queries on one of the servers and receive results upon request (Figure 1). Note that very often, the data is not private information, the data owner chooses to outsource mainly due to the high resource consumption associated with maintaining these continuous queries. Assuming that the server charges the data owner according to the computation resources consumed and the volume of traffic processed for answering the queries, the server then has an incentive to deceive the owner and the client for increased profit. Furthermore, the server might have a competing interest to provide fraudulent answers to a particular client. Hence, a passive malicious server could drop query results or provide random answers in order to reduce the computation resources required for answering queries, while a compromised or active malicious server might be willing to spend

additional computational resources to provide fraudulent results (by altering, dropping, or introducing spurious answers). In other cases, incorrect answers might simply be a result of faulty software, or due to load shedding strategies, which are essential tools for dealing with bursty streaming data [Tatbul and Zdonik 2006; Arasu and Manku 2004; Babcock et al. 2004; Tatbul et al. 2003].

Ideally, the data owner and the client should be able to verify the integrity of the computation performed by the server using significantly fewer resources than having the query answered directly, i.e., the data owner evaluates the query locally and then transmits the entire query result to the client. We aim at designing a *synopsis* or *certificate* which can verify correctness: the output of a simple function which can easily be computed on both the input and output of the computation in order to verify that the result is correct. Further, the client should have the capability to tolerate errors caused by load shedding algorithms or other non-malicious operations, while at the same time being able to identify mal-intended attacks which have a significant impact on the result.

The present work concentrates on "`GROUP BY, COUNT`" and "`GROUP BY, SUM`" queries. Such queries are especially common on data streams. For example, in network monitoring applications, often one would like to compute the total number of packets originated and destined to certain IP addresses. This problem has been studied extensively in the networking domain, and a variety of solutions, based on designing custom hardware or fast approximate counters, have been proposed; see e.g. [Zhao et al. 2006]. Also notice that the "`GROUP BY, COUNT`" query is equivalent to computing the frequencies of the tuples in a stream, and this problem has received a lot of attention in the data streaming literature. Indeed, most streaming algorithms deal with either the frequencies directly [Cormode and Muthukrishnan 2005] or their relatives, such as frequency moments [Alon et al. 1996; Indyk and Woodruff 2005], heavy hitters [Cormode and Muthukrishnan 2003; Karp et al. 2003; Metwally et al. 2006], quantiles [Gilbert et al. 2002; Greenwald and Khanna 2001], inverse frequencies [Cormode et al. 2005], etc. All theses works focus on computing the answers to these queries but not their verification.

In this article, we develop solutions for verifying "`GROUP BY, COUNT`" and "`GROUP BY, SUM`" queries on any type of grouping imposed on the input data. First, we provide a solution for verifying the absolute correctness of queries in the presence of any error, and second, an algorithm for supporting *semantic load shedding*, which allows the server to drop tuples in a selected small number of groups. In the latter case we design techniques that can tolerate a small number of inconsistent answers while guaranteeing that the rest are correct. Furthermore, we strengthen the scheme so that we can determine not only *whether* there are some errors but also *where* they are, and correct them if necessary, which is important in many critical applications. We also discuss the hardness of supporting *random load shedding*, where small errors are allowed for a wide range of answers.

There is considerable work on query authentication in an outsourced database [Hacigumus et al. 2002; Devanbu et al. 2003; Martel et al. 2004; Bertino et al. 2004; Pang and Tan 2004; Pang et al. 2005; Li et al. 2006] or an outsourced data stream [Li et al. 2007; Papadopoulos et al. 2007]. However, most of these works consider simple selection/aggregation queries, and to the best of our knowledge,

none of the existing techniques can support "`GROUP BY, COUNT`" and "`GROUP BY, SUM`" queries, in spite of their importance. Interestingly, unlike all the prior works that are based on cryptographic primitives such as digital signatures and collision-resistant hash functions, our solutions use algebraic and probabilistic techniques to compute a small synopsis on the true query result, which is then communicated to the client so as to verify the correctness of the query result returned by the server. Therefore we use the word "verification" as opposed to "authentication" to differentiate our techniques with the existing ones. In addition to the differences with respect to the techniques, there are also some subtle yet interesting differences between the security guarantees provided by "verification" and "authentication". We will defer the discussion on these issues to Section 8.

If a client wants to verify the query results with absolute confidence, the only solution is for the data owner to evaluate the query exactly and transmit the entire result to the client, which obviates the need of outsourcing. Hence, we provide high-confidence probabilistic solutions with arbitrarily minuscule probability of error, and develop verification algorithms that incur minimal resources, in terms of both the memory consumption of the data owner and the data owner-client network bandwidth.

Towards this goal the contributions of this work are:

(1) A randomized synopsis (Section 4.1), which we call PIRS, that raises an alarm with very high confidence if there exists at least one error in the query results. Specifically, the data owner maintains a constant-sized synopsis (three machine words) of the current query result, and transmits the synopsis to the client (via a secure channel) upon a verification request. The client then can verify the query result returned by the server using only this small synopsis (see Figure 1). The data owner can maintain the synopsis using constant space and low processing cost per tuple in the stream ($O(1)$ for count queries and $O(\log n)$ or $O(\log \mu)$ for sum queries, where $n$ is the number of possible groups and $\mu$ is the update amount per tuple). We also give a theoretical analysis of the algorithm that proves its space optimality on the bits level. In addition, we show the following.
   —A strong result stating that the same synopsis can be used for verifying multiple simultaneous queries with the same aggregate attribute but different group-by partitionings (Section 4.2). The size of the synopsis is the same as that for verifying a single query.
   —A rigorous analysis on the security guarantee of the proposed scheme under multiple attacks of the server with unlimited computing power (Section 4.3).
   —An adaption of the basic scheme to support queries on sliding windows (Section 4.4).

(2) A variety of generalizations of the PIRS synopsis, which use the basic synopsis as a building block to create new schemes including:
   —PIRS$^\gamma$ (Section 5.1), a generalization of PIRS for raising alarms when the number of errors exceeds a predefined threshold. This synopsis thus allows some room of error for the server (e.g. using semantic load shedding): As long as there are not too many errors (less than $\gamma$), the server is still considered to be trustworthy.

—PIRS$^{\gamma*}$ (Section 5.2), a strengthened version of PIRS$^{\gamma}$ so that when the number of errors is tolerable, we can also locate and even correct these errors. Therefore, PIRS$^{\gamma*}$ also acts as an error-correcting code, which guarantees that the complete and correct query results can be delivered to the client (provided the number of errors is less than $\gamma$).
—PIRS$^{\pm\gamma}$ (Section 5.3), an approximate version of PIRS$^{\gamma}$ that has a much reduced size.
—FM-PIRS (Section 5.4), a synopsis that can be used to estimate the actual number of errors accurately. FM-PIRS is a parameter-free version of PIRS$^{\pm\gamma}$ in the sense that it does not depend on $\gamma$. In particular, when the number of errors exceeds $\gamma$, PIRS$^{\pm\gamma}$ simply raises an alarm, while FM-PIRS also reports an estimate of the actual number of errors. FM-PIRS has a smaller size than PIRS$^{\pm\gamma}$ for large enough $\gamma$.

(3) Hardness results, including:
—A discussion of the difficulty behind supporting random load shedding and some simple heuristics (Section 6.1).
—Some lower bound results on the hardness of verifying some other related queries, such as joins (Section 6.2).

(4) Finally, an extensive empirical evaluation (Section 7) of the techniques designed in this paper using live network traffic, showing that our algorithms not only provide strong theoretical guarantees, but also work extremely well in practice and are very simple to implement.

## 2. PROBLEM FORMULATION

The queries examined in this work have the following structure:

```
SELECT G_1, ..., G_M, AGG(A_1), ..., AGG(A_N) FROM T
WHERE ... GROUP BY G_1, ..., G_M
```

Note also that `GROUP BY` aggregate queries have wide applications in monitoring and statistical analysis of data streams (e.g., in networking and telephony applications). Previous work has addressed exactly these types of queries from various aspects (see [Zhang et al. 2005] and related work therein) other than outsourcing. For example, a query that appears frequently in network monitoring applications is the following:

```
SELECT source_ip, destination_ip, SUM(packet_size) FROM IP_Trace
GROUP BY source_ip, destination_ip   (*)
```

In the rest of the paper we will use this query as our main motivating example and concentrate on sum and count. Other aggregates that can be converted to these two (e.g., average, standard deviation, etc.) can be easily supported, by verifying each component separately (i.e., verifying the sum and the count in the case of average). We will focus on verifying this query as a continuous query; adaptations to sliding windows are discussed in Section 4.4.

**Data stream model.** Following the example query (*), the "`GROUP BY`" predicate partitions the streaming tuples into a set of $n$ groups, computing one sum per group. The data stream can be viewed as a sequence of additions (or subtractions) over a set of items in $[n] = \{1, \ldots, n\}$. Denote this data stream as $\mathcal{S}$ and its

$\tau$-th tuple as $s^\tau = (i, u^\tau)$, an update of amount $u^\tau$ to the $i$th group. Formally, the query answer can be expressed as a dynamic vector of non-negative integers $\mathbf{v}^\tau = [v_1^\tau, \ldots, v_n^\tau] \in \mathbb{N}^n$, containing one component per group aggregate. Initially, $\mathbf{v}^0$ is the zero vector. A new tuple $s^\tau = (i, u^\tau)$ updates the corresponding group $i$ in $\mathbf{v}^\tau$ as $v_i^\tau = v_i^{\tau-1} + u^\tau$. We allow $u^\tau$ to be either positive or negative, but require $v_i^\tau \geq 0$ for all $\tau$ and $i$. When count queries are concerned, we have $u^\tau = 1$ for all $\tau$. We assume that the $L_1$ norm of $\mathbf{v}^\tau$ is always bounded by some large $m$, i.e., at any $\tau$, $\|\mathbf{v}^\tau\|_1 = \sum_{i=1}^n v_i^\tau \leq m$. Our streaming model is the same as the general Turnstile model of [Muthukrishnan 2003], and our algorithms are designed to work under this model. The readers are referred to two excellent papers [Muthukrishnan 2003; Babcock et al. 2002] for detailed discussions of data stream models.

**Problem definition.** The problem of *Continuous Query Verification on data streams (CQV)* is defined as follows:

*Definition* 2.1. Given a data stream $\mathcal{S}$, a continuous query $\mathcal{Q}$ and a user defined parameter $\delta \in (0, \frac{1}{2})$, design a synopsis $\mathcal{X}$ of $\mathbf{v}$ such that for any $\tau$, given any $\mathbf{w}^\tau$ and using $\mathcal{X}(\mathbf{v}^\tau)$, we: 1. raise an alarm with probability at least $1 - \delta$ if $\mathbf{w}^\tau \neq \mathbf{v}^\tau$; 2. shall not raise an alarm if $\mathbf{w}^\tau = \mathbf{v}^\tau$.

Here $\mathbf{w}^\tau$, for example, could be the answer provided by the server, while $\mathcal{X}(\mathbf{v}^\tau)$ is the synopsis communicated to the client from the data owner for verifying vector $\mathbf{v}^\tau$.

With this definition we raise an alarm with high probability if any component (or group answer) $v_i^\tau$ is inconsistent. Consider a server that is using semantic load shedding, i.e., dropping tuples from certain groups, on bursty stream updates. In this scenario the aggregate of a certain, small number of components will be inconsistent without malicious intent. We would like to design a technique that allows a certain degree of tolerance in the number of erroneous answers contained in the query results, rather than raising alarms indistinctly. The following definition captures the semantics of *Continuous Query Verification with Tolerance for a Limited Number of Errors (CQV$^\gamma$)*:

*Definition* 2.2. For any $\mathbf{w}, \mathbf{v} \in \mathbb{N}^n$, let $E(\mathbf{w}, \mathbf{v}) = \{i \mid w_i \neq v_i\}$. Then define $\mathbf{w} \neq_\gamma \mathbf{v}$ iff $|E(\mathbf{w}, \mathbf{v})| \geq \gamma$ and $\mathbf{w} =_\gamma \mathbf{v}$ iff $|E(\mathbf{w}, \mathbf{v})| < \gamma$. Given a data stream $\mathcal{S}$, a continuous query $\mathcal{Q}$, and user defined parameters $\gamma \in \{1, \ldots, n\}$ and $\delta \in (0, \frac{1}{2})$, design a synopsis $\mathcal{X}$ of $\mathbf{v}$ such that, for any $\tau$, given any $\mathbf{w}^\tau$ and using $\mathcal{X}(\mathbf{v}^\tau)$, we: 1. raise an alarm with probability at least $1 - \delta$, if $\mathbf{w}^\tau \neq_\gamma \mathbf{v}^\tau$; 2. shall not raise an alarm if $\mathbf{w}^\tau =_\gamma \mathbf{v}^\tau$.

Note that CQV is the special case of CQV$^\gamma$ with $\gamma = 1$. Similarly, we would like to design techniques that can support random load shedding, i.e., which can tolerate small absolute or relative errors on any component irrespective of the total number of inconsistent components. The following definition captures the semantics of *Continuous Query Verification with Tolerance for Small Errors (CQV$^\eta$)*:

*Definition* 2.3. For any $\mathbf{w}, \mathbf{v} \in \mathbb{N}^n$, let $\mathbf{w} \not\approx_\eta \mathbf{v}$ iff there is some $i$ such that $|w_i - v_i| > \eta$, and $\mathbf{w} \approx_\eta \mathbf{v}$ iff $|w_i - v_i| \leq \eta$ for all $i \in [n]$. Given a data stream $\mathcal{S}$, a continuous query $\mathcal{Q}$, and user defined parameters $\eta$ and $\delta \in (0, \frac{1}{2})$, design a synopsis $\mathcal{X}$ of $\mathbf{v}$ such that, for any $\tau$, given any $\mathbf{w}^\tau$ and using $\mathcal{X}(\mathbf{v}^\tau)$, we: 1. raise

an alarm with probability at least $1-\delta$, if $\mathbf{w}^\tau \not\approx_\eta \mathbf{v}^\tau$; 2. shall not raise an alarm if $\mathbf{w}^\tau \approx_\eta \mathbf{v}^\tau$.

Note that the definition above requires the *absolute* errors for each $v_i^\tau$ to be no larger than $\eta$. It is also possible to use *relative* errors, i.e., raise an alarm iff there is some $i$ such that $|w_i^\tau - v_i^\tau|/|v_i^\tau| > \eta$. Thus CQV is also a special case of CQV$^\eta$ with $\eta = 0$.

Related definitions are also possible. For example, one may wish to bound the *sum of the absolute errors*, or bound both the number and the size of the errors. We do not discuss these variations in detail, since they are for the most part similar to the ideas we outline subsequently.

We will work under the standard RAM model. Under this model, it is assumed that addition, subtraction, multiplication, division, or modular arithmetic operations involving two words take one unit of time. We also assume that $n/\delta$ and $m/\delta$ fit in a word. In the rest of the paper, we drop the superscript $\tau$ when there is no confusion.

## 3. POSSIBLE SOLUTIONS

This section presents some intuitive solutions and discusses why they are not sufficient for solving the CQV problem. We focus on count queries only; the discussion extends to sum queries since count is a special case of sum. Abusing notations, we use $|\mathbf{v}|$ to denote the number of non-zero entries of $\mathbf{v}$.

**A naïve solution.** A naïve solution to our problem is for the data owner to always maintain $\mathbf{v}$ exactly, using $\Theta(|\mathbf{v}| \log m)$ bits of space. When it receives a verification request from a client, it computes a collision-resistant hash function (e.g., SHA0 or SHA1) of $\mathbf{v}$ and transmits the hash to the client. Although this simple solution incurs a small network overhead, it has two major drawbacks. First, the owner's memory consumption is linear in $|\mathbf{v}|$, which is too large when there are a large number of groups or when there are multiple queries that need to be supported. In fact, all streaming algorithms strive to use space substantially smaller than linear. Second, whenever the data owner receives a verification request from some client, unless $\mathbf{v}$ has not changed since the last request, the hash of $\mathbf{v}$ needs to be recomputed, taking $O(|\mathbf{v}|)$ time. For many queries like the example query (*), $|\mathbf{v}|$ could easily go up to the order of millions.

Since space complexity is the most important measure for all streaming algorithms, in the rest of the paper we only consider algorithms that require sublinear space. One might think of the following two simple solutions to reduce the high space requirement of this naïve algorithm.

**Random sampling.** A first attempt is random sampling. Assuming a sampling rate $r$, the client randomly chooses $rn$ groups. If $\mathbf{w} \neq \mathbf{v}$, this method will raise an alarm if the error in $\mathbf{w}$ is one of the sampled groups, which happens with probability $r$. In order to satisfy the problem statement requirements we need to set $r = 1-\delta$. For CQV$^\gamma$, if the server modifies exactly $\gamma$ answers, then the probability of raising an alarm is only roughly $r^\gamma$, which is obviously too small for practical $r$'s and $\gamma$'s. Thus, random sampling that guarantees $\delta$ failure probability can at most reduce the space cost by a small fraction.

**Sketches.** Recent years have witnessed a large number of sketching techniques (e.g. [Alon et al. 1996; Cormode and Muthukrishnan 2005; Bar-Yossef et al. 2002; Flajolet and Martin 1985]) that are designed to summarize high-volume streaming data with small space. It is tempting to maintain such a sketch $\mathcal{K}(\mathbf{v})$ for the purpose of verification. When the server returns some $\mathbf{w}$, we compute $\mathcal{K}(\mathbf{w})$, which is possible since $\mathbf{w}$ exactly tells us what the elements have appeared in the stream and what their frequencies are. Then we check if $\mathcal{K}(\mathbf{v}) = \mathcal{K}(\mathbf{w})$. Below, let us consider the two well-known AMS sketches from the work of Alon et al. [1996] to see whether they work, and if so, how large they are.

The $F_0$ sketch uses a pairwise independent random hash function $r$ and computes the maximum number of trailing zeros in the binary form of $r(i)$ for all tuples in the stream. This sketch is oblivious to the number of times a tuple appears, so will not detect any errors as long as $\mathbf{w}$ and $\mathbf{v}$ have the same set of locations on the groups with nonzero entries. Simple extensions, such as counting the number of occurrences of each value of $r(i)$, blow up the space used by a logarithmic factor but give no useful guarantee.

The $F_2$ sketch does look like a promising approach. It computes the sum $Y(\mathbf{v}) = \sum_{i=1}^{n} h(i)v_i$, where $h : \{1, \ldots, n\} \to \{-1, 1\}$ is chosen randomly from a family of 4-wise independent hash functions. This process is repeated over independent choices of the hash functions to improve the accuracy. If $\mathbf{v} = \mathbf{w}$, we obviously have $Y(\mathbf{v}) = Y(\mathbf{w})$, so the sketch does not have any false positives. If $\mathbf{v} \neq \mathbf{w}$, then the $L_2$ norm of $\|\mathbf{v} - \mathbf{w}\|_2^2$ is nonzero. From [Alon et al. 1996] we know that $(Y(\mathbf{v} - \mathbf{w}))^2 = (Y(\mathbf{v}) - Y(\mathbf{w}))^2$ is an unbiased estimator of $\|\mathbf{v} - \mathbf{w}\|_2^2$. If $\mathbf{w} \neq \mathbf{v}$, $\|\mathbf{v} - \mathbf{w}\|_2^2 > 0$, and the sketch will raise an alarm unless $Y(\mathbf{v}) = Y(\mathbf{w})$, or this estimator is 0. This means that we will miss the alarm only if this estimator is off by $\|\mathbf{v} - \mathbf{w}\|_2^2$. By using 4 independent estimators and taking the average, we can bound this probability from above by 1/2. Therefore, in order to guarantee a $1 - \delta$ overall success probability, the data owner needs to maintain $4 \cdot \log \frac{1}{\delta}$ independent copies. Since each copy requires three words (one for $Y(\mathbf{v})$ and two for the 4-wise independent hash function $h$), the total size of the synopsis will be 384 words for $\delta = 2^{-32}$. Later we will see that our solution achieves the same security guarantee using merely three words, which can also be maintained in $O(1)$ time.

The analysis above shows that $4 \cdot \log \frac{1}{\delta}$ copies of the sketch are guaranteed to work, but one may ask if they are really necessary. Below we will present a concrete adversarial construction showing that the analysis above is almost tight and using less copies will indeed compromise the security guarantee.

For concreteness, suppose we adopt the BCH4 scheme (c.f. [Rusu and Dobra 2007]) to construct a 4-wise independent random hash function $f : [n] \to \{0, 1\}$, and then set $h(i) = 2f(i) - 1$. Note that $\sum_{i=1}^{n} h(i)(v_i - w_i) = 2\sum_{i=1}^{n} f(i)(v_i - w_i) - \sum_{i=1}^{n}(v_i - w_i)$. Below we will construct a $\mathbf{w} \neq \mathbf{v}$ such that $\sum_{i=1}^{n}(v_i - w_i) = 0$, but $\sum_{i=1}^{n} f(i)(v_i - w_i) = 0$ is likely to happen.

Without loss of generality we assume $n = 2^r - 1$. Let $S_0$ and $S_1$ be two random $r$-bit integers. The BCH4 scheme computes $f(i)$ as $f(i) = (S_0 \odot i) \oplus (S_1 \odot i^3)$, where $\oplus$ is the vector dot product over the last $r$ bits evaluated on $\mathbb{Z}_2$, i.e., assuming the last $r$ bits of $x$ (resp. $y$) is $x_1, \ldots, x_r$ (resp. $y_1, \ldots, y_r$), then $x \oplus y = (\sum_{i=1}^{r} x_i y_i) \bmod 2$. We construct $\mathbf{w}$ as follows. For all odd $i$ and for $i = 2^{r-1}$, we set $w_i = v_i$; for even

$i \neq 2^{r-1}$, we set $v_i - w_i = -1$ if $i < 2^{r-1}$, and $v_i - w_i = 1$ if $i > 2^{r-1}$. It is clear that $\sum_{i=1}^{n}(v_i - w_i) = 0$. We will show that if $S_0 < 2^{r-1}$, then $\sum_{i=1}^{n} f(i)(v_i - w_i) = 0$. Consider any odd $i < 2^{r-1}$, and $j = i + 2^{r-1}$. We have

$$\begin{aligned} f(j) &= (S_0 \odot j) \oplus (S_1 \odot j^3) \\ &= (S_0 \odot (i + 2^{r-1})) \oplus (S_1 \odot (i + 2^{r-1})^3) \\ &= (S_0 \odot i) \oplus (S_1 \odot (i + 2^{r-1})^3), \end{aligned}$$

where the last equality is due to the fact that the first bit of $S_0$ is zero. On the other hand, for even $i$, since

$$\begin{aligned} (i + 2^{r-1})^3 &= i^3 + 3 \cdot i^2 \cdot 2^{r-1} + 3 \cdot i \cdot 2^{2r-2} + 2^{3r-3} \\ &\equiv i^3 \qquad (\bmod 2^r), \end{aligned}$$

we have $f(i) = f(j)$. Thus, the pair $f(i)(v_i - w_i)$ and $f(j)(v_j - w_j)$ cancel out, and we have $\sum_{i=1}^{n} f(i)(v_i - w_i) = 0$. So when $S_0 < 2^{r-1}$, which happens with probability $1/2$, the sketch will miss this erroneous $\mathbf{w}$. This means that in order to guarantee a $1 - \delta$ overall success probability, indeed $\Theta(\log \frac{1}{\delta})$ copies are necessary.

One can evaluate other possible sketching techniques. In general, they have the common feature that in order to give a suitably small probability of failure, a significant amount of space is required, asymptotically worse than the $O(1)$ words of space our solution uses.

## 4. PIRS: POLYNOMIAL IDENTITY RANDOM SYNOPSIS

### 4.1 The Basic Synopses

This section presents our basic synopsis, called *Polynomial Identity Random Synopses* (PIRS) and denoted by $\mathcal{X}(\mathbf{v})$, for solving the CQV problem (Definition 2.1). The synopsis, as the name suggests, is based on testing the identity of polynomials by evaluating them at a randomly chosen point. The technique of verifying polynomial identities can be traced back to the seventies [Freivalds 1979]. It has found applications in e.g. verifying matrix multiplications and pattern matching [Motwani and Raghavan 1995]. PIRS has two variants, named PIRS-1 and PIRS-2, respectively.

**PIRS-1.** Let $p$ be some prime such that $\max\{m/\delta, n\} < p$. For the space analysis, let $p \leq 2\max\{m/\delta, n\}$ — according to Bertrand's Postulate [Nagell 1981] such a $p$ always exists. We will work in the field $\mathbb{Z}_p$, i.e., all additions, subtractions, and multiplications are done modulo $p$. For the first PIRS, denoted PIRS-1, we choose $\alpha$ from $\mathbb{Z}_p$ uniformly at random and compute $\mathcal{X}(\mathbf{v})$ incrementally from $\mathcal{X}(\mathbf{v}^{\tau-1})$ and $s^\tau = (i, u^\tau)$ as:

$$\mathcal{X}(\mathbf{v}^\tau) = \mathcal{X}(\mathbf{v}^{\tau-1})(\alpha - i)^{u^\tau},$$

where we define $\mathcal{X}(0) = \mathcal{X}(\mathbf{v}^0) = 1$. Consequently, we have

$$\mathcal{X}(\mathbf{v}) = (\alpha - 1)^{v_1} \cdot (\alpha - 2)^{v_2} \cdot \cdots \cdot (\alpha - n)^{v_n}.$$

We assume that $n, m, \delta, p$ are publicly known to the data owner and all the clients. Then the data owner picks $\alpha$ secretly and maintains $\mathcal{X}(\mathbf{v})$. Upon a verification request, the data owner returns PIRS to the client, which consists of only two

words: $\alpha$ and $\mathcal{X}(\mathbf{v})$. Given any $\mathbf{w}$ returned by the server, the client can use PIRS to check if $\mathbf{w} = \mathbf{v}$ with high probability, by computing

$$\mathcal{X}(\mathbf{w}) = (\alpha - 1)^{w_1} \cdot (\alpha - 2)^{w_2} \cdot \cdots \cdot (\alpha - n)^{w_n}.$$

We first check if $\sum_{i=1}^{n} w_i > m$, if so we reject $\mathbf{w}$ immediately. Otherwise, if $\mathcal{X}(\mathbf{w}) = \mathcal{X}(\mathbf{v})$, then we declare that $\mathbf{w} = \mathbf{v}$; else we raise an alarm. It is easy to see that we never raise a false alarm. Therefore we only need to show that we miss a true alarm with probability at most $\delta$.

THEOREM 4.1. *Given any $\mathbf{w} \neq \mathbf{v}$, PIRS raises an alarm with probability at least $1 - \delta$.*

PROOF. Consider the polynomials $f_{\mathbf{v}}(x) = (x-1)^{v_1}(x-2)^{v_2} \cdots (x-n)^{v_n}$ and $f_{\mathbf{w}}(x) = (x-1)^{w_1}(x-2)^{w_2} \cdots (x-n)^{w_n}$. Since a polynomial with 1 as its leading coefficient, i.e., the coefficient of the term with the largest degree, is completely determined by its zeroes (with multiplicities), we have $f_{\mathbf{v}}(x) \equiv f_{\mathbf{w}}(x)$ iff $\mathbf{v} = \mathbf{w}$. If $\mathbf{v} \neq \mathbf{w}$, since both $f_{\mathbf{v}}(x)$ and $f_{\mathbf{w}}(x)$ have degree at most $m$, $f_{\mathbf{v}}(x) = f_{\mathbf{w}}(x)$ happens at no more than $m$ values of $x$, due to the fundamental theorem of algebra. Since we have $p \geq m/\delta$ choices for $\alpha$, the probability that $\mathcal{X}(\mathbf{v}) = \mathcal{X}(\mathbf{w})$ happens is at most $\delta$ over the random choice of $\alpha$. □

We now analyze the update time to maintain $\mathcal{X}(\mathbf{v})$ as new updates are observed. For count queries, each tuple increments one of the $v_i$'s by one, so the update cost is constant (one subtraction and one multiplication). For sum queries, a tuple $s = (i, u)$ increases $v_i$ by $u$, so we need to compute $(\alpha - i)^u$, which can be done in $O(\log u)$ (exponentiation by repeated squaring) time. To perform a verification with $\mathbf{w}$, we need to compute $(x - i)^{w_i}$ for each nonzero entry $w_i$ of $\mathbf{w}$, which takes $O(\log w_i)$ time, so the time needed for a verification is $O(\sum \log w_i) = O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$. Since both $\mathcal{X}(\mathbf{v})$ and $\alpha$ are smaller than $p$, the space complexity of the synopsis is $O(\log \frac{m}{\delta} + \log n)$ bits, or O(1) words.

THEOREM 4.2. *PIRS-1 occupies $O(\log \frac{m}{\delta} + \log n)$ bits of space, spends $O(1)$ (resp. $O(\log u)$) time to process a tuple for count (resp. sum) queries, and $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$ time to perform a verification.*

Some special care is needed when $u$ is negative (or handling deletions for count queries), as the field $\mathbb{Z}_p$ is not equipped with division. We need first to compute $(\alpha - i)^{-1}$, the multiplicative inverse of $(\alpha - i)$ in $\mathbb{Z}_p$, in $O(\log p)$ time (using Euclid's gcd algorithm [Knuth 1997]), and then compute $(\alpha - i)^{-1 \cdot |u|}$.

**PIRS-2.** When $n \ll m$ we can actually do slightly better in terms of the space usage, in a scheme we refer to as PIRS-2. Now we choose the prime $p$ between $\max\{m, n/\delta\}$ and $2\max\{m, n/\delta\}$. For $\alpha$ chosen uniformly at random from $\mathbb{Z}_p$, we compute

$$\mathcal{X}(\mathbf{v}) = v_1\alpha + v_2\alpha^2 + \cdots + v_n\alpha^n.$$

Note that this is also straightforward to maintain over a stream of updates, by adding on $u\alpha^i$ in response to update $s = (i, u)$. By considering the polynomial $f_{\mathbf{v}}(x) = v_1 x + v_2 x^2 + \cdots + v_n x^n$, the proof outline of Theorem 4.1 holds in this case, and the above choice of $p$ ensures that the desired $\delta$ guarantee is maintained.

But now PIRS-2 has an $O(\log n)$ update cost for both count and sum queries, since we need to compute $u\alpha^i$ for a tuple $(i, u)$ in the stream. Without repeating the details, we conclude with the following.

THEOREM 4.3. *PIRS-2 occupies $O(\log m + \log \frac{n}{\delta})$ bits of space, spends $O(\log n)$ time to process a tuple, and $O(|\mathbf{w}| \log n)$ time to perform a verification.*

Since the space complexities of PIRS-1 and PIRS-2 are comparable and PIRS-1 has a better update time for count queries, we recommend using PIRS-1 unless $n$ is small compared to $m$ and typical $u$.

An important property of (either variant of) PIRS is that the verification can also be performed in one pass of $\mathbf{w}$ using a constant number of words of memory. This is especially useful when $|\mathbf{w}|$ is large. The client will be able to receive $\mathbf{w}$ in a streaming fashion, verifies it online, and either forward it to a dedicated server for further processing, or a network storage device for offline use.

**Space optimality.** Below we give a lower bound showing that PIRS is space-optimal on the bits level for almost all values of $m$ and $n$.

THEOREM 4.4. *Any synopsis solving the CQV problem with error probability at most $\delta$ has to keep $\Omega(\log \frac{\min\{m,n\}}{\delta})$ bits.*

PROOF. We will take an information-theoretic approach. Assume that $\mathbf{v}$ and $\mathbf{w}$ are both taken from a universe $\mathcal{U}$, and let $\mathcal{M}$ be the set of all possible memory states the synopsis might keep. Any synopsis $\mathcal{X}$ can be seen as a function $f : \mathcal{U} \to \mathcal{M}$; and if $\mathcal{X}$ is randomized, it can be seen as a function randomly chosen from a family of such functions $\mathcal{F} = \{f_1, f_2, \ldots\}$, where $f_i$ is chosen with probability $p(f_i)$. Without loss of generality, we assume that $p(f_1) \geq p(f_2) \geq \cdots$. Note that $\mathcal{X}$ needs at least $\log |\mathcal{M}|$ bits to record the output of the function and $\log |\mathcal{F}|$ bits to describe the function chosen randomly from $\mathcal{F}$.

For any $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$, let $\mathcal{F}_{\mathbf{w},\mathbf{v}} = \{f \in \mathcal{F} \mid f(\mathbf{w}) = f(\mathbf{v})\}$. For a randomized synopsis $\mathcal{X}$ to solve CQV with error probability at most $\delta$, the following must hold for all $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$:

$$\sum_{f \in \mathcal{F}_{\mathbf{w},\mathbf{v}}} p(f) \leq \delta. \tag{1}$$

Let us focus on the first $k = \lceil \delta \cdot |\mathcal{F}| \rceil + 1$ functions $f_1, \ldots, f_k$. It is easy to see that $\sum_{i=1}^{k} p(f_i) > \delta$. Since there are a total of $|\mathcal{M}|^k$ possible combinations for the outputs of these $k$ functions, by the pigeon-hole principle, we must have

$$|\mathcal{U}| \leq |\mathcal{M}|^k \tag{2}$$

so that no two $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$ have $f_i(\mathbf{w}) = f_i(\mathbf{v})$ for all $i = 1, \ldots, k$; otherwise we would find $\mathbf{w}, \mathbf{v}$ that violate (1).

Taking log on both sides of (2), we have

$$\log |\mathcal{U}| \leq (\lceil \delta \cdot |\mathcal{F}| \rceil + 1) \log |\mathcal{M}|.$$

Since $\mathbf{v}$ has $n$ entries whose sum is at most $m$, by simple combinatorics, we have $|\mathcal{U}| \geq \binom{m+n}{n}$, or $\log |\mathcal{U}| \geq \min\{m, n\}$. We thus obtain the following tradeoff:

$$|\mathcal{F}| \cdot \log |\mathcal{M}| = \Omega(\min\{m, n\}/\delta).$$

If $\log|\mathcal{F}| \leq (1-\epsilon)\log(\min\{m,n\}/\delta)$ (i.e., $|F| \leq (\min\{m,n\}/\delta)^{1-\epsilon}$) for any constant $\epsilon \in (0,1)$, then $\mathcal{X}$ has to use super-polylogarithmic space $\log|\mathcal{M}| = \Omega((\min\{m,n\}/\delta)^{\epsilon})$; else $\mathcal{X}$ has to keep $\log|\mathcal{F}| \geq \log(\min\{m,n\}/\delta)$ random bits. □

Therefore, when $m \leq n$, PIRS-1 is optimal as long as $\log n = O(\log \frac{m}{\delta})$; when $m > n$, PIRS-2 is optimal as long as $\log m = O(\log \frac{n}{\delta})$. Our bounds are not tight when $\log \frac{m}{\delta} = o(\log n)$ or $\log \frac{n}{\delta} = o(\log m)$.

**Practical issues.** The theoretical analysis above focuses on the bit-level space complexity. When implemented, however, both PIRS-1 and PIRS-2 use three words ($p$, $\alpha$, and $\chi(\mathbf{v})$), and thus do not seem to have any difference. Nevertheless, there are some technical issues to be considered in practice.

First, we shall choose $p$ to be the maximum prime that fits in a word, so as to minimize $\delta$. Note that $\delta = m/p$ for PIRS-1 and $\delta = n/p$ for PIRS-2. For instance if we use 64-bit words and $m < 2^{32}$, then $\delta$ is at most $2^{-32}$ for PIRS-1, which makes any error highly unlikely (1 in four billion). If speed is a key consideration, careful choice of $p$ can allow faster implementation: for example, choosing $p$ to be a Mersenne prime ($p$ is of the form $p = 2^{\ell} - 1$ for some $\ell$) allows the modulo arithmetic to be performed using simple addition and subtraction operations. Such tricks are well known in the literature (see [Thorup 2000]), so we do not discuss them further.

Second, since we need to extract the group id $i$ from each incoming tuple directly, without the use of a dictionary (which would blow up the memory cost), the size of the group space, $n$, needs to be large for certain queries. For example, the query (*) of Section 2 has a group space of $n = 2^{64}$ (the combination of two IP addresses), although the actual number of nonzero entries $|\mathbf{v}|$ may be much less than $n$. In this case, since $m$ is typically much smaller, PIRS-1 would be the better choice.

**Synchronization.** In the discussion above we have assumed that the data owner and the servers have a synchronized view of the stream in order to apply PIRS correctly. Such a synchronization is hard to maintain without substantial overhead. Below we present an idea to get around this issue. The observation is that the client does not need to, and in fact unable to, verify the query results at every time instance. Typically one would just like to verify the results every now and then. We can use the following scheme to accomplish such a task.

When the data owner outsources the data stream, he/she attaches a timestamp to each tuple accordingly to his/her own clock. When the client wants to verify the query results at a certain time $t$, he/she will send such a request to the owner shortly before time $t$. Meanwhile he/she also asks the server to provide the query results at time $t$. When the server receives the first tuple with a timestamp after $t$, it will then return to the client the up-to-date query results. Note that the server never uses its own clock. The data owner, on the other hand, also returns to the client the PIRS synopsis when his/her clock reaches $t$. Now the query results and the synopsis are synchronized (unless the server cheats and returns the results for a different $t$), and the client can perform the verification when both arrive.

### 4.2 Handling Multiple Queries

The discussion so far focused on handling a single query per PIRS synopsis. Our techniques though can be used for handling multiple queries simultaneously. Consider a client who registers at the data owner a number of aggregate queries on a single attribute (e.g., packet size) but with different partitioning on the input tuples (e.g., source/destination IP and source/destination port), and wishes to verify all of them. Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ be $k$ such queries, and let the $i$-th query partition the incoming tuples into $n_i$ groups for a total of $n = \sum_{i=1}^{k} n_i$ groups. A simple solution for this problem would be to apply the PIRS algorithm once per query, using space linear in $k$. But by treating all the queries as one unified query of $n$ groups we can use one PIRS synopsis to verify the combined vector $\mathbf{v}$. The time cost for processing one update increases linearly in $k$, since each incoming tuple is updating $k$ components of $\mathbf{v}$ at once (one group for every query in the worst case):

COROLLARY 4.5. *PIRS-1 for $k$ queries occupies $O(\log \frac{m}{\delta} + \log n)$ bits of space, spends $O(k)$ (resp. $O(k \log u)$) time to process a tuple for count (resp. sum) queries, and $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$ time to perform a verification.*

Clearly, this is a strong result, since we can effectively verify multiple queries with a few words of memory and communication.

### 4.3 Information Disclosure on Multiple Attacks

Theorem 4.1 bounds the success rate for detecting a single attack attempted by the server. After an error has been detected, the client can choose to disclose this information to the server. If the error is not reported, then Theorem 4.1 will continue to hold. However, errors can occur due to faulty software or bad communication links, and may not be intentional. In this case we would like to give a warning to the server. Since an adversary can extract knowledge from this warning (e.g., it knows at least that the same response on the same data will always fail), the guarantee of Theorem 4.1 does not strictly hold. In order to restore the $1 - \delta$ success rate after a reported attack, the synopsis has to be recomputed from scratch, which is impossible in a streaming setting. Hence, it is important to rigorously quantify the loss of guarantee after a series of warnings have been sent out without resetting the synopsis.

Let $e_k = 1$ if the $k$-th attack goes undetected and $e_k = 0$ otherwise. Let $p_k$ be the probability that the server succeeds in its $k$-th attack after $k-1$ failed attempts, i.e., $p_k = \Pr[e_k = 1 \mid e_1 = 0, \ldots, e_{k-1} = 0]$. From Theorem 4.1 we know that $p_1 \leq \delta$. In what follows we upper bound $p_k$ with respect to the most powerful server, denoted as *Alice*, to demonstrate the strength of PIRS. We assume that Alice: 1. Knows how PIRS works except its random *seed*, $\alpha$; 2. Maximally explores the knowledge that could be gained from one failed attack; and 3. Possesses unbounded computational power.

Next, we precisely quantify the best Alice could do to improve $p_k$ over multiple attacks. Denote by $\mathcal{R}$ the space of seeds used by PIRS. For any $\mathbf{w}, \mathbf{v}$ denote the set of *witnesses* $\mathcal{W}(\mathbf{w}, \mathbf{v}) = \{r \in \mathcal{R} \mid \text{PIRS raises an alarm on } r\}$ and the set of *non-witnesses* $\overline{\mathcal{W}}(\mathbf{w}, \mathbf{v}) = \mathcal{R} - \mathcal{W}(\mathbf{w}, \mathbf{v})$. Note that $|\overline{\mathcal{W}}(\mathbf{w}, \mathbf{v})| \leq \delta|\mathcal{R}|$ if $\mathbf{w} \neq \mathbf{v}$, and $\overline{\mathcal{W}}(\mathbf{w}, \mathbf{v}) = \mathcal{R}$ if $\mathbf{w} = \mathbf{v}$. Suppose the seed PIRS uses is $r$. If Alice returns a correct

answer $\mathbf{w} = \mathbf{v}$, she cannot infer anything about $r$. If she returns some $\mathbf{w} \neq \mathbf{v}$ and gets a warning, it is possible that Alice can determine $r \notin \overline{\mathcal{W}}(\mathbf{w}, \mathbf{v})$. However, even if we assume that Alice has enough computational power to compute both the sets of witnesses and non-witnesses, it is impossible for her to infer which witness PIRS is using as $r$. After $k-1$ failed attacks using $\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}$, the set of seeds that Alice has ruled out is $\bigcup_{i=1}^{k-1} \overline{\mathcal{W}}(\mathbf{w}_i, \mathbf{v}_i)$, whose cardinality is at most $(k-1)\delta|\mathcal{R}|$. Thus, we have:

LEMMA 4.6. $p_k \leq \frac{\delta}{1-(k-1)\delta}$.

PROOF.

$$p_k = \frac{|\text{set of non-witnesses}|}{|\text{set of remaining seeds}|} = \frac{|\overline{\mathcal{W}}(\mathbf{w}_k, \mathbf{v}_k)|}{|\mathcal{R} - \bigcup_{i=1}^{k-1} \overline{\mathcal{W}}(\mathbf{w}_i, \mathbf{v}_i)|} \leq \frac{\delta}{1-(k-1)\delta}.$$

□

THEOREM 4.7. *Assuming that Alice has made a total of $k$ attacks to PIRS for any $k$, the probability that none of them succeeds is at least $1 - k\delta$.*

PROOF. This probability is

$$\begin{aligned}
&\Pr[e_1 = 0 \wedge \cdots \wedge e_k = 0] \\
&= \prod_{i=1}^{k} (1 - \Pr[e_i = 1 \mid e_1 = 0, \ldots, e_{i-1} = 0]) \\
&\geq \prod_{i=1}^{k} \left(1 - \frac{\delta}{1-(i-1)\delta}\right) = \prod_{i=1}^{k} \frac{1-i\delta}{1-(i-1)\delta} \\
&= \frac{1-\delta}{1} \cdot \frac{1-2\delta}{1-\delta} \cdot \cdots \cdot \frac{1-k\delta}{1-(k-1)\delta} = 1 - k\delta.
\end{aligned}$$

□

Theorem 4.7 shows that PIRS is very resistant towards coordinated multiple attacks even against an adversary with *unlimited computational power*. For a typical value of $\delta = 2^{-32}$, PIRS could tolerate millions of attacks before the probability of success becomes noticeably less than 1. Most importantly, the drop in the detection rate to $1 - k\delta$ occurs only if the client chooses to disclose the attacks to the server. Such disclosure is not required in many applications.

### 4.4 Handling Sliding Windows

In this section we discuss how to extend PIRS to support sliding windows. We will focus on PIRS-1 for count queries only; the same arguments apply to sum queries, as well as to PIRS-2, PIRS$^{\gamma}$, and PIRS$^{\pm\gamma}$.

An important property of PIRS-1 is that it is decomposable, i.e., for any $\mathbf{v}_1, \mathbf{v}_2, \mathcal{X}(\mathbf{v}_1 + \mathbf{v}_2) = \mathcal{X}(\mathbf{v}_1) \cdot \mathcal{X}(\mathbf{v}_2)$. (For PIRS-2, we have $\mathcal{X}(\mathbf{v}_1 + \mathbf{v}_2) = \mathcal{X}(\mathbf{v}_1) + \mathcal{X}(\mathbf{v}_2)$) This property allows us to extend PIRS for periodically sliding windows using standard techniques [Datar et al. 2002]. Using our earlier example, one such sliding window query might be the following.

```
SELECT SUM(packet_size) FROM IP_Trace
```

```
GROUP BY source_ip, destination_ip
WITHIN LAST 1 hour SLIDE EVERY 5 minutes
```

In this case, we can build a PIRS-1 for every 5-minute period, and keep it in memory until it expires from the sliding window. Assume that there are $k$ such periods in the window, and let $\mathcal{X}(\mathbf{v}_1), \ldots, \mathcal{X}(\mathbf{v}_k)$ be the PIRS for these periods. In addition, the data owner maintains the overall PIRS $\mathcal{X}(\mathbf{v}) = \prod_{i=1}^{k} \mathcal{X}(\mathbf{v}_i)$. When a new PIRS $\mathcal{X}(\mathbf{v}_{k+1})$ completes, we update $\mathcal{X}(\mathbf{v})$ as $\mathcal{X}(\mathbf{v}) := \mathcal{X}(\mathbf{v}) \cdot \mathcal{X}(\mathbf{v}_{k+1}) \cdot (\mathcal{X}(\mathbf{v}_1))^{-1}$. The following result is immediate.

COROLLARY 4.8. *For a periodically sliding window query with $k$ periods, our synopsis uses $O(k(\log \frac{m}{\delta} + \log n))$ bits of space, spends $O(1)$ time to process an update, and $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$ time to perform a verification.*

If various window sizes consisting of between 1 to $k$ periods are to be supported, we decompose the $k$ periods into a number of *dyadic* intervals. For simplicity assume that $k$ is a power of 2. We organize these intervals into $\ell = \log k$ levels. On level 0, there are $k$ intervals each consisting one period; on level $i, 1 \leq i \leq \ell - 1$, there are $k/2^i$ intervals, each spanning $2^i$ periods. Note that there are a total of $2k - 1$ such dyadic intervals. We build one PIRS for each interval, so the total size of the entire synopsis is still $O(k(\log \frac{m}{\delta} + \log n))$. Since a PIRS at level $i+1$ can be computed in constant time from two PIRS's at level $i$, the amortized update cost remains $O(1)$. Upon a verification request with a window size of $q$ periods, we can decompose the window into at most $O(\log k)$ dyadic intervals, and combine those corresponding PIRS's together to form the correct synopsis for the query window.

COROLLARY 4.9. *To support sliding window queries with various window sizes of up to $k$ periods, our synopsis uses $O(k(\log \frac{m}{\delta} + \log n))$ bits of space, spends $O(1)$ time to process an update, and $O(\log k)$ time to assemble the required synopsis upon a verification request. The client spends $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$ time to perform a verification.*

## 5. TOLERANCE FOR FEW ERRORS

This section presents a synopsis for solving the CQV$^\gamma$ problem (Definition 2.2). Let $\gamma$ be the number of components in $\mathbf{v}$ that are allowed to be inconsistent. First, we present a construction that gives an exact solution that satisfies the requirements of CQV$^\gamma$, and requires $O(\gamma^2 \log \frac{1}{\delta} \log n)$ bits of space. This synopsis can be strengthened so that we can also locate and even correct these errors. But this exact solution uses space quadratic in $\gamma$, so we next provide an *approximate* solution which uses only $O(\gamma \log \frac{1}{\delta}(\log m + \log n))$ bits. Finally, we present another synopsis that can estimate the number of errors. This estimator uses polylogarithmic space and does not depend on $\gamma$. All the solutions use PIRS as a black box, and therefore can choose either PIRS-1 or PIRS-2. We state all the results using PIRS-1 for count queries. The corresponding results for sum queries and PIRS-2 can be obtained similarly.

### 5.1 PIRS$^\gamma$: An Exact Solution

By using PIRS as a building block we can construct a synopsis that satisfies the requirements of CQV$^\gamma$. This synopsis, referred to as PIRS$^\gamma$, consists of multiple

*layers*, where each layer contains $k = c_1\gamma^2$ *buckets* ($c_1 \geq 1$ is a constant to be determined shortly). Each component of $\mathbf{v}$ is assigned to one bucket per layer, and each bucket is represented using only its PIRS synopsis (see Figure 2). PIRS$^\gamma$ raises an alarm if at least $\gamma$ buckets in any layer raise an alarm. The intuition is that if there are fewer than $\gamma$ errors, no layer will raise an alarm, and if there are more than $\gamma$ errors, at least one of the layers will raise an alarm with high probability (when the $\gamma$ inconsistent components do not collide on any bucket for this layer). By choosing the probability of failure of the individual PIRS synopsis carefully, we can guarantee that PIRS$^\gamma$ achieves the requirements of Definition 2.2.

$$\begin{bmatrix} \underset{\{v_3\}}{\mathcal{X}_{11}} & \underset{\{v_2,v_5\}}{\mathcal{X}_{12}} & \underset{\{v_1,v_4,v_6\}}{\mathcal{X}_{13}} \\ \underset{\{v_1,v_3\}}{\mathcal{X}_{21}} & \underset{\{v_2,v_5\}}{\mathcal{X}_{22}} & \underset{\{v_4,v_6\}}{\mathcal{X}_{23}} \end{bmatrix}$$

Fig. 2. The PIRS$^\gamma$ synopsis.

**Algorithm 1**: PIRS$^\gamma$-INITIALIZE(Prime p, Threshold $\gamma$)

```
1 c = 4.819, k = ⌈cγ²⌉
2 Generate x and y uniformly at random from ℤ_p
3 for ℓ = 1, ..., ⌈log 1/δ⌉ do
4 |   Layer L_ℓ = [X_1(v) := 0, ··· , X_k(v) := 0]
  |   // X_j(v) is a PIRS synopsis with δ' = 1/cγ
```

**Algorithm 2**: PIRS$^\gamma$-UPDATE(Tuple $s = (i, u)$)

```
1 for ℓ = 1, ..., ⌈log 1/δ⌉ do
2 |   b_ℓ(i) = xi + y mod k + 1
3 |   Update L_ℓ.X_{b_ℓ(i)}(v) using s
```

**Algorithm 3**: PIRS$^\gamma$-VERIFY(Vector $\mathbf{w}$)

```
1 for ℓ = 1, ..., ⌈log 1/δ⌉ do
2 |   Layer M_ℓ = [X_1(w) := 0, ··· , X_k(w) := 0]
  |   // X_j(w) is a PIRS synopsis with δ' = 1/cγ
3 |   for i = 1, ..., n do
4 |   |   b_ℓ(i) = xi + y mod k + 1
5 |   |   Update M_ℓ.X_{b_ℓ(i)}(w) by s = (i, w_i)
6 |   if |{j | L_i.X_j(v) ≠ M_i.X_j(w), 1 ≤ j ≤ k}| ≥ γ then Raise an alarm
```

Concentrating on one layer only, let $b$ be a pairwise independent hash function which maps the range $\{1,\ldots,n\}$ uniformly onto $\{1,\ldots,k\}$. PIRS$^\gamma$ assigns $v_i$ to the $b(i)$-th bucket, and for each bucket computes the PIRS synopsis of the assigned subset of $v_i$'s with probability of failure $\delta' = 1/(c_2\gamma)$ ($c_2 \geq 1$ is a constant to be determined shortly). According to Theorem 4.2 each of these $k$ synopses occupies $O(\log \frac{m}{\delta'} + \log n) = O(\log m + \log n)$ bits. Given some $\mathbf{w} =_\gamma \mathbf{v}$, since there are fewer than $\gamma$ errors, the algorithm will not raise an alarm. We can choose constants $c_1$ and $c_2$ such that if $\mathbf{w} \neq_\gamma \mathbf{v}$, then the algorithm will raise an alarm with probability at least $1/2$ for this layer. In this case there are two cases when the algorithm will fail to raise an alarm: 1. There are fewer than $\gamma$ buckets that contain erroneous components of $\mathbf{w}$; 2. There are at least $\gamma$ buckets containing erroneous components but at least one of them fails due to the failure probability of PIRS. We show that by setting constants $c_1, c_2 = 4.819$ either case occurs with probability at most $1/4$. Consider the first case. Since the $v_i$'s are assigned to the buckets in a pairwise independent fashion, we can guarantee that the mapping of the $\gamma$ erroneous components onto the $k$ buckets is injective with probability

$$\begin{aligned} & 1 - \left(1 - \frac{1}{k}\right)^{\frac{\gamma(\gamma-1)}{2}} \\ \leq\ & 1 - \left(1 - \frac{1}{c_1\gamma^2}\right)^{\gamma^2/2} \leq 1 - 2^{-1/c_1} \leq \frac{1}{4}, \end{aligned} \tag{3}$$

where the last inequality holds by our choice of $c_1$. Next, consider the second case. The probability that some of the $\gamma$ buckets that are supposed to raise an alarm fail is:

$$1 - (1-\delta')^\gamma = 1 - \left(1 - \frac{1}{c_2\gamma}\right)^{c_2\gamma/c_2} \leq 1 - 2^{-\frac{2}{c_2}} < \frac{1}{4}, \tag{4}$$

which holds as long as $c_2 \geq 4.819$.

Therefore, using one layer PIRS$^\gamma$ will raise an alarm with probability at least $1/2$ on some $\mathbf{w} \neq_\gamma \mathbf{v}$, and will not raise an alarm if $\mathbf{w} =_\gamma \mathbf{v}$. By using $\log \frac{1}{\delta}$ layers and reporting an alarm if at least one of these layers raises an alarm, the probability is boosted to $1-\delta$.

THEOREM 5.1. *For any $\mathbf{w} \neq_\gamma \mathbf{v}$, PIRS$^\gamma$ raises an alarm with probability at least $1-\delta$. For any $\mathbf{w} =_\gamma \mathbf{v}$, PIRS$^\gamma$ will not raise an alarm.*

In addition to the $k \log \frac{1}{\delta}$ PIRS synopses, we also need to generate the hash function $b$ mapping updates to buckets. This can be done by picking $x$ and $y$ uniformly at random from $\mathbb{Z}_p$, and computing $b(i) = xi + y \bmod p \bmod k$. This generates a function that is pairwise-independent over the random choices of $x$ and $y$ [Motwani and Raghavan 1995]. To perform a verification, we can compute for all the layers in parallel while making one pass over $\mathbf{w}$. The detailed initialization, update and verification algorithms for PIRS$^\gamma$ appear in Algorithms 1, 2, and 3. The next theorem bounds both the space and time complexity of PIRS$^\gamma$.

THEOREM 5.2. *PIRS$^\gamma$ requires $O(\gamma^2 \log \frac{1}{\delta}(\log m + \log n))$ bits, spends $O(\log \frac{1}{\delta})$ time to process a tuple in the stream, and $O(|\mathbf{w}|(\gamma + \log \frac{m}{|\mathbf{w}|}) \log \frac{1}{\delta})$ time to perform*

*a verification.*

With careful analysis a smaller constant in the big-O above can be achieved in practice. For a given $\gamma$, we choose the minimum $k$ such that (3) is at most $1/2$, and choose $1/\delta'$ very large (close to the maximum allowed integer) so that (4) is almost zero. For instance if $\gamma = 2$ and 3, then $2\log\frac{1}{\delta}$ and $6\log\frac{1}{\delta}$ words suffice, respectively. For arbitrary $\gamma$, the storage requirement is $2\gamma^2\log\frac{1}{\delta}$ words in the worst case.

### 5.2 PIRS$^{\gamma*}$: Locating and Correcting Errors

When there are a small number of errors (at most $\gamma$), PIRS$^\gamma$ will not raise an alarm, which gives some leeway to the server. This is often necessary so that the server can cope with large volumes of incoming data using some semantic load shedding strategies. However, in some critical applications, if the client demands complete correctness, PIRS$^\gamma$ is not sufficient, since it only tells the client *if* there are $< \gamma$ errors but not *where* they are. In this subsection, we present PIRS$^{\gamma*}$, a strengthened version of PIRS$^\gamma$ that is able to identify which groups are affected by errors, and even compute the correct sums for the affected groups. The idea is to take advantage of a technique based on the *binary decomposition* of the group identifier: this idea has been used in different contexts, such as finding frequent items in data streams [Cormode and Muthukrishnan 2003; Cormode et al. 2005]. Here, we need to embed the PIRS summary into this decomposition.

Applying the binary decomposition to PIRS$^\gamma$, we increase the amount of information kept about each bucket. In addition to keeping a PIRS synopsis of all items which fall into a given bucket, we additionally keep $2\lceil\log n\rceil$ PIRS synopses, arranged as a two-dimensional array $A$ of size $\lceil\log n\rceil \times 2$. When an update to group $i$ is placed into bucket $b(i)$, we also update the PIRS in $A[j, bit(i,j)]$, for all $1 \le j \le \lceil\log n\rceil$, where $bit(i,j)$ denotes the $j$th bit in the binary representation of $i$.

To perform a query verification, we compare the array $A$ of PIRS synopses computed for both $\mathbf{v}$ and $\mathbf{w}$ for each bucket. If all corresponding entries match, then (with high probability) there is no erroneous components in the bucket. If, for any $j$, the PIRS in both $A[j,0]$ and $A[j,1]$ do not match, then this indicates that there is more than one erroneous component in this bucket, because a single erroneous $i$ cannot contaminate both $A[j,0]$ and $A[j,1]$. Otherwise, there must be exactly one erroneous component falling in this bucket. Our above analysis indicates that this will indeed be the case for all erroneous components with high probability, providing that there are at most $\gamma$ such components. In this case, for each $j$, exactly one of $A[j,0]$ and $A[j,1]$ will not match. If it is $A[j,1]$, this indicates that the $j$th bit of the identifier $i$ of the erroneous group is 1; else it is 0. Using all $\lceil\log n\rceil$ pairs of PIRS, the identifier can therefore be recovered exactly.

Now the client can locate all the erroneous components $w_i$ in the result $\mathbf{w}$ returned by the server. Moreover, we have enough information to recover each true $v_i$ for each wrong result. Consider each bucket at a certain layer $\ell$ that contains exactly one error. Suppose the error is $v_i$. From the process above we can identify each such bucket and also the index $i$. Note that the data owner will return $\mathcal{X}_{b_\ell(i)}(\mathbf{v}) = \prod_j (\alpha - j)^{v_j}$. Since only $v_i$ is unknown in this equation, we can in principle find $v_i$ by solving the equation, although this requires computing the discrete logarithm,

for which efficient algorithms are not known. Nevertheless, if we plug in PIRS-2, the equation becomes as $\mathcal{X}_{b_\ell(i)}(\mathbf{v}) = \sum_j v_j \alpha^j$. Thus, $v_i$ can be found efficiently using field subtractions and divisions.

In PIRS$^{\gamma*}$, we replace each PIRS in PIRS$^{\gamma}$ with an array of $O(\log n)$ PIRS, so the space and time increases by an $O(\log n)$ factor.

THEOREM 5.3. *PIRS$^{\gamma*}$ requires $O(\gamma^2 \log \frac{1}{\delta} \log n(\log m + \log n))$ bits, spends $O(\log \frac{1}{\delta} \log n)$ time to process a tuple in the stream, and $O(|\mathbf{w}|(\gamma + \log \frac{m}{|\mathbf{w}|}) \log \frac{1}{\delta} \log n)$ time to perform a verification. For any $\mathbf{w} \neq_\gamma \mathbf{v}$, PIRS$^{\gamma*}$ raises an alarm with probability $1-\delta$; for any $\mathbf{w} =_\gamma \mathbf{v}$, PIRS$^{\gamma*}$ will not raise an alarm but correctly identify and recover all the errors in $\mathbf{w}$ with probability $1-\delta$.*

Note that when the number of errors, say $\lambda$, is no more than $\gamma$, PIRS$^{\gamma*}$ can recover *all* the errors with high probability, which is a very strong guarantee. When $\lambda > \gamma$, there are too many errors to expect a complete recovery of all the query results (like any error-correcting code cannot recover the data when there are too many errors). Nevertheless, we show that PIRS$^{\gamma*}$ can still recover a good proportion of the results. For this analysis, we use *precision* and *recall* to measure the performance of the synopsis. Precision refers to the probability that an identified error is truly an actual error. Since PIRS does not have false positives, precision is always 1. Recall, on the other hand, is the percentage of the actual errors that have been recovered, or equivalently, the probability that any one error has been captured by the synopsis. From the previous discussions, we know that for any given error $\mathcal{E}$, if it falls into a bucket by itself in *any* of the layers, then PIRS$^{\gamma*}$ can correctly recover it. For a particular layer, because the errors are distributed into the buckets pairwise-independently and there are $c_1\gamma^2$ buckets, the probability that the bucket containing $\mathcal{E}$ is the same as the bucket for any of the other $\lambda - 1$ errors is at most $\lambda/(c_1\gamma^2)$ following the union bound. Since the $\log \frac{1}{\delta}$ layers are mutually independent, the probability that this collision happens in all layers is $\left(\frac{\lambda}{c_1\gamma^2}\right)^{\log \frac{1}{\delta}} = \delta^{\Omega(\log(\gamma^2/\lambda))}$.

THEOREM 5.4. *When there are $\lambda > \gamma$ errors, PIRS$^{\gamma*}$ raises an alarm with probability $1-\delta$ and recovers the errors with a recall of $1 - \delta^{\Omega(\log(\gamma^2/\lambda))}$.*

### 5.3 PIRS$^{\pm\gamma}$: An Approximate Solution

The exact solution works well when only a small number of errors can be tolerated. In applications where $\gamma$ is large, the quadratic space requirement is prohibitive. If we relax the definition of CQV$^\gamma$ to allow raising alarms when *approximately* $\gamma$ errors have been observed, we can design more space-efficient algorithms. This approximation is often acceptable since when $\gamma$ is large, users probably will not concern too much if the number of errors detected deviates from $\gamma$ by a small amount. This section presents such an approximate solution, denoted with PIRS$^{\pm\gamma}$, that guarantees the following:

THEOREM 5.5. *PIRS$^{\pm\gamma}$: 1. raises no alarm with probability at least $1-\delta$ on any $\mathbf{w} =_{\gamma^-} \mathbf{v}$ where $\gamma^- = (1 - \frac{c}{\ln \gamma})\gamma$; and 2. raises an alarm with probability at least $1-\delta$ on any $\mathbf{w} \neq_{\gamma^+} \mathbf{v}$ where $\gamma^+ = (1 + \frac{c}{\ln \gamma})\gamma$, for any constant $c > -\ln \ln 2 \approx 0.367$.*

Note that this is a very sharp approximation; the multiplicative approximation ratio $1 \pm \frac{c}{\ln \gamma}$ is close to 1 for large $\gamma$.

PIRS$^{\pm\gamma}$ also contains multiple layers of buckets, where each bucket is assigned a subset of the components of $\mathbf{v}$ and summarized using PIRS (Figure 2). Focusing on one layer only, our goal is, for any $\mathbf{w} =_{\gamma^-} \mathbf{v}$, to not to raise an alarm with probability at least $1/2 + \epsilon$ for some constant $\epsilon \in (0, 1/2)$, and on any $\mathbf{w} \neq_{\gamma^+} \mathbf{v}$ to raise an alarm with probability at least $1/2 + \epsilon$. By using $O(\log \frac{1}{\delta})$ independent layers and reporting the majority of the results, the probabilistic guarantee will be boosted to $1 - \delta$ using Chernoff bounds [Motwani and Raghavan 1995].

Let $k$ be the number of buckets per layer. The components of $\mathbf{v}$ are distributed into the $k$ buckets in a $\gamma^+$-wise independent fashion, and for each bucket the PIRS summary of those components is computed using $\delta' = 1/\gamma^2$. Given some $\mathbf{w}$, let this layer raise an alarm only if all the $k$ buckets report alarms. The intuition is that if $\mathbf{w}$ contains more than $\gamma^+$ erroneous members, then the probability that every bucket gets at least one such component is high; and if $\mathbf{w}$ contains fewer than $\gamma^-$ erroneous members, then the probability that there exists some bucket that is not assigned any erroneous members is also high.

The crucial factor that determines whether a layer could possibly raise an alarm is the distribution of erroneous components into buckets. The event that all buckets raise alarms is only possible if each bucket contains at least one inconsistent component. Let us consider all the inconsistent components in $\mathbf{w}$ in some order, say $w_1, w_2, \ldots$, and think of each of them as a collector that randomly picks a bucket to "collect". Assume for now that we have enough inconsistent elements, and let the random variable $Y$ denote the number of inconsistent components required to collect all the buckets, i.e., $Y$ is the smallest $i$ such that $w_1, \ldots, w_i$ have collected all the buckets. Then the problem becomes an instantiation of the *coupon collector's problem* [Motwani and Raghavan 1995] (viewing buckets as coupons and erroneous components as trials). With $k$ buckets, it is known that $E(Y) = k \ln k + O(k)$, therefore we set $k$ such that $\gamma = \lceil k \ln k \rceil$. It is easy to see that $k = O(\gamma / \ln \gamma)$, hence the desired storage requirement.

We need the following sharp bounds showing that $Y$ cannot deviate too much from its mean.

Lemma 5.6 [Motwani and Raghavan 1995]. *For any constant $c'$,*

$$\Pr[Y \leq k(\ln k - c')] \leq e^{-e^{c'}} + o(1),$$
$$\Pr[Y \geq k(\ln k + c')] \leq 1 - e^{-e^{-c'}} + o(1),$$

*where $o(1)$ depends on $k$.*

Notice that $\ln \gamma \leq 2 \ln k$ for any $k \geq 2$, so Lemma 5.6 also implies that for any real constant c:

$$\Pr[Y \leq \gamma - c\frac{\gamma}{\ln \gamma} = \gamma^-] \leq e^{-e^{c}} + o(1), \tag{5}$$
$$\Pr[Y \geq \gamma + c\frac{\gamma}{\ln \gamma} = \gamma^+] \leq 1 - e^{-e^{-c}} + o(1). \tag{6}$$

Now, consider the following two cases. If $\mathbf{w} =_{\gamma^-} \mathbf{v}$, then the probability that these fewer than $\gamma^-$ independent erroneous components cover all buckets is bounded by (5), which is also the upper bound for the probability that the layer raises an

alarm. Thus, for any $c \geq 0$, the probability of raising a false alarm is (for large enough $\gamma$) at most

$$e^{-e^c} \leq 1/e.$$

If $\mathbf{w} \neq_{\gamma^+} \mathbf{v}$, then considering only $\gamma^+$ of the inconsistent components which are independently distributed to the buckets, there are two cases in which a true alarm is not raised: 1. These $\gamma^+$ components do not cover all buckets; and 2. All the buckets are covered but at least one of them fails to report an alarm. The probability that the first case occurs is bounded by (6); while the probability that the second case happens is at most $1-(1-\delta')^k$. By the union bound, the total probability that we produce a false negative is at most

$$1 - e^{-e^{-c}} + o(1) + 1 - (1-\delta')^k \leq 2 - e^{e^{-c}} - 2^{-\frac{2}{\gamma}} + o(1).$$

For $\gamma$ large enough, there exists a constant $\epsilon > 0$ such that this probability is at most $1/2 - \epsilon$ for any $c > -\ln\ln 2$.

To summarize, if $c > -\ln\ln 2 \approx 0.367$, then both the false positive and false negative probabilities are at most $1/2-\epsilon$ for some constant $\epsilon$ at one layer with $k = O(\gamma/\log\gamma)$ buckets. Below we analyze the error probabilities of using $\ell = O(\log\frac{1}{\delta})$ independent layers.

To drive down the error probabilities for both false positives and false negatives to $\delta$, we use $\ell = O(\log\frac{1}{\delta})$ layers and report the simple majority of their "votes". We quantify this probability for false negatives; the other case is symmetric.

Each layer can be viewed as a coin flip that raises a true alarm with probability at least $1/2+\epsilon$. Let the random variable $Z$ denote the number of layers that raise alarms. This process is a sequence of independent Bernoulli trials, hence $Z$ follows the binomial distribution. For $\ell$ independent layers, the expectation of $Z$ is at least $\mu = (1/2+\epsilon)\ell$. By the Chernoff bound, the probability that a majority of layers raise alarms is

$$\Pr[Z < \frac{1}{2}\ell] = \Pr[Z < \left(1 - \frac{2\epsilon}{1+2\epsilon}\right)\mu] < e^{-\frac{\mu}{2}(\frac{2\epsilon}{1+2\epsilon})^2}. \tag{7}$$

Therefore, we need to ensure that $e^{-\frac{\mu}{2}(\frac{2\epsilon}{1+2\epsilon})^2} \leq \delta$, which can be satisfied by taking $\ell = \lceil \frac{1+2\epsilon}{\epsilon^2}\ln\frac{1}{\delta}\rceil$.

Finally, we need to generate a $\gamma^+$-wise independent random hash function to map groups to buckets. Using standard techniques we can generate such a function using $O(\gamma\log n)$ truly random bits. Specifically, the technique of [Wegman and Carter 1981] for constructing *t-universal hash families* can be used. Let $p$ be some prime between $n$ and $2n$, and $\alpha_0, \ldots, \alpha_{\gamma-1}$ be $\gamma$ random numbers chosen uniformly and independently from $\mathbb{Z}_p$. Then we set

$$b(i) = \sum_{j=0}^{t-1} \alpha_j i^j \bmod k + 1.$$

This function is guaranteed to be drawn from a $t$-wise independent family of functions (so that, over the random choice of the function, the probability of $t$ items colliding under the hash function is $1/k^{t-1}$). For an incoming tuple $s = (i, u)$, we compute $b(i)$ using the $\alpha_j$'s in $O(\gamma)$ time (using Horner's rule), and then perform

the update to the corresponding PIRS. This requires the storage of $O(\gamma^+) = O(\gamma)$ truly random numbers per layer. We have thus obtained the desired results:

THEOREM 5.7. *PIRS$^{\pm\gamma}$ uses $O(\gamma \log \frac{1}{\delta}(\log m + \log n))$ bits of space, spends $O(\gamma \log \frac{1}{\delta})$ time to process an update and $O(|\mathbf{w}|(\gamma + \log \frac{m}{|\mathbf{w}|}) \log \frac{1}{\delta})$ time to perform a verification.*

### 5.4 FM-PIRS: Estimating the Number of Errors

From the previous section we see that by allowing two-sided errors, we can reduce the size of the synopsis from quadratic in $\gamma$ to linear. However, for large $\gamma$, even a linear size is too large. Further, the update cost of PIRS$^{\pm\gamma}$ is also linear in $\gamma$, making these synopses very expensive to maintain when $\gamma$ is large. In this section, we present an improved solution for the CQV$^{\gamma}$ problem, whose size and update cost only depend on the degree of approximation, but not $\gamma$, thus allowing it to scale well with $\gamma$. Unlike PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$, FM-PIRS tries to directly estimate the number of errors in the result provided by the server, and then compare with $\gamma$, so it is a stronger version in some sense. As a result, FM-PIRS can also support a wider range of values of $\gamma$, which can be given only at verification time. For small values of $\gamma$, the bounds and guarantees of PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$ are preferred, but for larger values of $\gamma$, the cost of FM-PIRS is preferable.

As the name suggest, FM-PIRS is a combination of PIRS and the *FM sketch* [Flajolet and Martin 1985], which is used to estimate the number of distinct elements in a stream.

**The FM sketch.** We first briefly describe the FM sketch. Suppose that the universe is $[n] = \{1, \ldots, n\}$. We pick a random hash function $h : [n] \to [2^L - 1]$ such that any $h(i)$ is uniformly distributed over $[2^L - 1]$, where $L = O(\log n)$. For each element $i$ in the stream, we compute $h(i)$ and denote by $r(i)$ the number of trailing zeros in the binary representation of $h(i)$. The FM sketch simply computes $R = \max\{r(i), \text{for all } i \text{ in the stream}\}$ and then outputs $1/\varphi \cdot 2^R$, where $\varphi \approx 0.7735$. This simple estimate has a large variance. To improve accuracy, the universe is divided into $k$ partitions using another random uniform hash function, and an $R_j$ is computed for each partition. Finally the output is $k/\varphi \cdot 2^{(R_1+\cdots+R_k)/k}$. It was shown in [Flajolet and Martin 1985] that this estimator has a bias bounded by $1 + 0.31/k$ and a standard error of $0.78/\sqrt{k}$.

**The FM-PIRS synopsis.** We will focus on the basic FM sketch with $k = 1$; generalization to larger $k$ will be straightforward. Our idea is to treat each "wrong" group $i$ such that $v_i \neq w_i$ as a distinct element in the universe $[n]$, and then compute $R = \max\{r(i), \text{for all wrong groups } i\}$. However, the data owner has no idea whether $i$ is a wrong group, so we cannot compute $r(i)$ directly. Instead, we create $L$ PIRS's $\mathcal{X}_1, \ldots, \mathcal{X}_L$ with $\delta' = \delta/L$. For any $i$, group $i$ is put into $\mathcal{X}_j$ if $j \leq r(i)$. Thus $\mathcal{X}_1$ gets half of the groups, $\mathcal{X}_2$ gets a quarter of the groups, etc. We can thus compute $R$ as follows.

LEMMA 5.8. *When all of $\mathcal{X}_1, \ldots, \mathcal{X}_L$ correctly captures the errors in them, which happens with probability at least $1-\delta' \cdot L = 1-\delta$, we have $R = \arg\max_j\{\mathcal{X}_j \text{ raises an alarm}\}$.*

PROOF. First, for any $j$, if $\mathcal{X}_j$ raises an alarm, there must be a wrong group $i$ that is distributed into $\mathcal{X}_j$, i.e., such that $r(i) \geq j$. So $R \geq r(i) \geq \arg\max_j\{\mathcal{X}_j \text{ raises an alarm}\}$.

On the other hand, consider the group $i$ with the maximum $r(i)$. $\mathcal{X}_{r(i)}$ must raise an alarm, so $R = r(i) \leq \arg\max_j\{\mathcal{X}_j \text{ raises an alarm}\}$. □

*Example:* Consider the following example with $n = 8$, and we use $L = 3$ PIRS's $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3$. Suppose the random hash function $h$ maps each index in $[n]$ as follows (in binary): $h(1) = 10, h(2) = 10, h(3) = 1, h(4) = 111, h(5) = 101, h(6) = 0, h(7) = 100$. Then we will allocate $v_1, v_2, v_6, v_7$ to $\mathcal{X}_1$, $v_6, v_7$ to $\mathcal{X}_2$, and $v_6$ to $\mathcal{X}_3$. If $v_2, v_3, v_5$, and $v_7$ later become erroneous, they will cause $\mathcal{X}_1$ and $\mathcal{X}_2$ to raise alarms (with high probability). Now we have $R = 2$ and the estimated number of errors is $1/\varphi \cdot 2^R = 5.17$.

It is straightforward to generalize the basic scheme to $k$ partitions. Thus we have the following.

THEOREM 5.9. *Fix any $k$, FM-PIRS has a size of $O(k \log n(\log m + \log n))$ bits, processes a tuple in expected time $O(1)$, and computes an estimate of the number of errors in the result in expected time $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$. With probability at least $1 - \delta$, the estimate has a bias bounded by $1 + 0.31/k$ and a standard error of $0.78/\sqrt{k}$.*

PROOF. Since each partition keeps $L = O(\log n)$ PIRS's, the overall size of FM-PIRS is $O(k \log n(\log m + \log n))$ bits. For an incoming tuple, only one partition gets affected, but 0 to $L$ PIRS's in this partition might get updated. Since the hash function $h$ is uniform, the expected number of PIRS's updated is $O(1)$. Upon receiving the FM-PIRS synopses of $\mathbf{v}$ and a result $\mathbf{w}$ from the server, we need to spend $O(\log w_i)$ expected time per non-zero entry of $\mathbf{w}$ to compute the FM-PIRS synopses of $\mathbf{w}$. So the expected time needed for an estimation is $O(\sum_i \log w_i) = O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$. Finally, the bias and standard error of the estimate follow from [Flajolet and Martin 1985]. □

**An analytical comparison of PIRS$^{\pm\gamma}$ and FM-PIRS.** Since FM-PIRS computes an estimate of the number of errors in $\mathbf{w}$, we can use FM-PIRS to do the same task PIRS$^{\pm\gamma}$ is designed for. For a fair comparison, we need to set $k$ such that FM-PIRS provides the same probabilistic guarantee that PIRS$^{\pm\gamma}$ does. Since the standard error of FM-PIRS is $O(1/\sqrt{k})$ and PIRS$^{\pm\gamma}$ allows a deviation of $O(1/\ln\gamma)$. By setting $k = O(\log^2\gamma)$, we can guarantee that FM-PIRS captures both false positives and false negatives with good probabilities (e.g., greater than 3/4). Finally, by using $O(\log\frac{1}{\delta})$ independent copies of FM-PIRS and take the median, the success probability can be boosted to $1 - \delta$, the same as what PIRS$^{\pm\gamma}$ guarantees. Finally, we only need $L = O(\log\gamma)$ since we are not interested in estimating the number of errors when there are over, say $2\gamma$ of them.

Under this configuration, FM-PIRS uses $O(\log^3\gamma(\log m + \log n)\log\frac{1}{\delta})$ bits of space. Thus asymptotically (as $\gamma$ grows) FM-PIRS is better than PIRS$^{\pm\gamma}$. However, for small $\gamma$ PIRS$^{\pm\gamma}$ should be better in terms of size, while FM-PIRS becomes better when $\gamma$ exceeds some large threshold. Nevertheless, FM-PIRS should always be much better in terms of update time. We further compare these two synopses empirically in Section 7.

## 6. HARDNESS RESULTS

### 6.1 Tolerance for Small Errors

In this subsection we prove the hardness of solving CQV$^\eta$ (Definition 2.3) using sublinear space, even if approximations are allowed. This problem can be interpreted as detecting if there is any component of $\mathbf{w}$ that has an absolute error exceeding a specified threshold $\eta$. We show that this problem requires at least $\Omega(n)$ bits of space.

THEOREM 6.1. *Let $\eta$ and $\delta \in (0, 1/2)$ be user specified parameters. Given a data stream $\mathcal{S}$, let $\mathcal{X}$ be any synopsis built on $\mathbf{v}$ that given $\mathbf{w}$: 1. raises an alarm with probability at most $\delta$ if $\mathbf{w} \approx_\eta \mathbf{v}$; and 2. raises an alarm with probability at least $1-\delta$ if $\mathbf{w} \not\approx_{(2-\epsilon)\eta} \mathbf{v}$ for any $\epsilon > 0$. Then $\mathcal{X}$ has to use $\Omega(n)$ bits.*

PROOF. We will reduce from the problem of approximating the infinite frequency moment, defined as follows. Let $A = (a_1, a_2, \ldots)$ be a sequence of elements from the set $\{1, \ldots n\}$. The *infinite frequency moment*, denoted by $F_\infty$, is the number of occurrences of the most frequent element. Alon et al. [1996] showed that any randomized algorithm that makes one pass over $A$ and computes $F_\infty$ with a relative error of at most $1/3$ and a success probability greater than $1-\delta$ for any $\delta < 1/2$, has to use $\Omega(n)$ memory bits. In particular, they proved that even if each element appears at most twice, it requires $\Omega(n)$ bits in order to decide if $F_\infty$ is 1 or 2 with probability at least $1-\delta$.

Let $\mathcal{X}$ by a synopsis solving the problem stated in Theorem 6.1. We will show how to use $\mathcal{X}$ to compute the infinite frequency moment for any $A$ in which each element appears at most twice. We will make one pass over $A$. For any element $i$ that we encounter, we update $\mathcal{X}$ with the tuple $s = (i, \eta)$. In the end, we verify $\mathbf{w} = 0$ using $\mathcal{X}(\mathbf{v})$. If $\mathcal{X}$ asserts that $\mathbf{w} \approx_\eta \mathbf{v}$, we return $F_\infty = 1$; if $\mathcal{X}$ asserts that $\mathbf{w} \not\approx_{(2-\epsilon)\eta} \mathbf{v}$, we return $F_\infty = 2$. It is not difficult to see that we have thus computed the correct $F_\infty$ with probability at least $1-\delta$. □

If we allow relative errors instead of absolute errors, the problem is still difficult, as can be shown by setting $s = (i, n)$ for element $i$, and doing the verification with $\mathbf{w} = (n/(1+\eta), \cdots, n/(1+\eta))$ in the proof above.

Given the hardness of solving CQV$^\eta$, we are interested in seeking alternative methods that might be able to give guarantees under different notions of approximation using less space than the exact solution. Here we briefly discuss one such method.

**The CM sketch.** The CM sketch [Cormode and Muthukrishnan 2005] uses $O(\frac{1}{\epsilon}\log\frac{1}{\delta'})$ words of space and provides an approximate answer $\widetilde{v}_i$ for any $i \in [n]$, that satisfies $v_i - 3\epsilon||\mathbf{v}||_1 \leq \widetilde{v}_i \leq v_i + 3\epsilon||\mathbf{v}||_1$ with probability $1-\delta'$, for any $\epsilon \in (0,1)$. However, this does not make it applicable for solving CQV$^\eta$ as: 1. The estimation depends on $||\mathbf{v}||_1$ and it only works well for skewed distributions. Even in that case, in practice the estimation works well only for the large $v_i$'s; and 2. $||\mathbf{v}||_1$ is not known in advance. However, if we can estimate an upper bound on $||\mathbf{v}||_1$, say $||\mathbf{v}||_1 \leq \Gamma$, then by setting $\epsilon = \frac{1}{3}\frac{\eta}{\Gamma}$ and $\delta' = \delta/n$, we can use the CM sketch to get approximate answers $\widetilde{v}_i$ such that $|\widetilde{v}_i - v_i| \leq \eta$ holds for all $i$ *simultaneously* with probability at least $1-\delta$. Now, given some $\mathbf{w}$, we generate an alarm iff there exists

some $i$ such that $|w_i - \widetilde{v}_i| \geq 2\eta$. This way, we give out a false alarm with probability at most $\delta$ if $\mathbf{w} \approx_\eta \mathbf{v}$, and generate an alarm with probability $1-\delta$ if $\mathbf{w} \not\approx_{3\eta} \mathbf{v}$. For other $\mathbf{w}$'s, no guarantee can be made. In particular, some false negatives may be observed for some range of $\mathbf{w}$ (see Figure 3). This solution uses $O(\frac{1}{\epsilon} \log \frac{n}{\delta} \log W)$ bits of space and $O(\log \frac{n}{\delta})$ time per update (where $W$ is the largest expressible integer in one word of the RAM model). The space dependence on $\frac{1}{\epsilon}$ is expensive, as $\frac{1}{\epsilon} = \frac{\Gamma}{\eta}$ in this case and the upper bound on $||\mathbf{v}||_1$ in practice might be large.

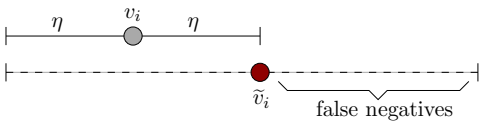



Fig. 3. False negatives for the CM sketch approach.

### 6.2 Lower Bounds for Other Queries

Our discussion of methods to verify queries has focused so far on `GROUP BY, SUM` and `GROUP BY, COUNT` queries. In this subsection, we study other natural query forms, and show that in contrast to the results so far, giving strong guarantees on answering these queries either exactly or approximately is not possible with a small synopsis. Our approach is to encode problems from communication complexity, which require a large amount of communication to solve, and argue that this entails a large synopsis in our setting.

**`GROUP BY, MIN` and `GROUP BY, MAX`.** Consider a query of the same form as that described in Section 2, except that the SQL aggregate over the $n$ groups is `MIN`. Now the semantics in the streaming setting are that $v_i^{\min}$ should be $\min_\tau\{u^\tau | s^\tau = (i, u^\tau)\}$ and $v_i^{\max}$ should be $\max_\tau\{u^\tau | s^\tau = (i, u^\tau)\}$.

LEMMA 6.2. *Any synopsis that guarantees with probability at least* $1-\delta$ *whether* $\forall i. v_i^{\min} = w_i^{\min}$ *requires space at least* $n$.

PROOF. We show a reduction to the communication complexity problem of INDEX. In this problem there are two players: Alice, who holds a bit string $x$ of $n$ bits in length, and Bob, who holds an index $1 \leq y \leq n$. Alice must send a single message to Bob, who must then compute the $y$th bit of string $x$. It has been shown that to solve this problem, even allowing a small probability $\delta$ of failure, requires Alice's message to be $\Omega(n)$ bits in size [Kushilevitz and Nisan 1997].

We now show that if a summary with the desired properties did exist, then it could be used to solve the INDEX problem. Alice creates a summary for her bit string $x$, by setting the value of group $i$ to 0 if the $i$th bit of $x$ is zero, and 1 otherwise. She then sends the summary $\chi(x)$ to Bob. Bob then records $\chi(x)$, and updates the summary with an item in group $y$ with weight 0. He then uses the original summary and the updated summary to test whether there is a difference between them. If there is, then he concludes that bit $y$ of $x$ is a 1; else, it is a 0. Therefore, if the summary succeeds with constant probability (say, at least 2/3), then it allows the INDEX problem to be solved, and so must require $\Omega(n)$ bits. □

LEMMA 6.3. *Any synopsis which guarantees with probability at least* $1-\delta$ *whether* $\forall i. v_i^{\max} = w_i^{\max}$ *requires space at least* $n$.

PROOF. The same approach works as in the previous lemma. The only difference is that Bob inserts an item in group $y$ with weight 1, and tests whether there has been any change in the summarized data. Again, this argument allows the INDEX problem to be solved, implying the lower bound on the size of the summary. □

We comment that the same approach can show the hardness of the `AVERAGE` aggregate. But, as remarked in Section 2, we can compute `AVERAGE` indirectly, by verifying the `SUM` and `COUNT` separately. Only if we ask the third party to directly compute `AVERAGE` without returning the `SUM` and `COUNT` does this hardness result hold.

**Join queries.** We now consider a join query, and show that this too is hard. In particular, consider a query of the form

```
SELECT L.a, SUM(L.c) FROM L, R
WHERE L.a = R.b
GROUP BY L.a
```

This can be represented as a product of two vectors, say $x$ and $y$, and the desired output is $v_i = x[i] \cdot y[i]$. However, again by communication complexity arguments, we can show that such a query requires a summary whose size is linear in the number of groups. In this case, we analyze the effect of the "communication" happening mid-way through processing the input data stream.

LEMMA 6.4. *Any synopsis which allows the query verification of the above query with probability at least* $1 - \delta$ *requires space* $\Omega(n)$.

PROOF. The DISJOINTNESS problem in Communication Complexity is when Alice holds a binary vector $x$ of length $n$, Bob holds a vector string $y$ of length $n$, and their goal is to determine whether $x \cdot y > 0$. It is known that any communication protocol to solve this problem must exchange $\Omega(n)$ bits between Alice and Bob, even allowing a small constant probability of failure.

To show the hardness of verifying the join query, we show that Alice and Bob could use a summary to solve DISJOINTNESS. Alice takes her string $x$, and builds a summary $\chi(x)$ by inserting every 1 bit as an item from $L$ into the summary with weight 1. She then sends the summary to Bob, who follows the same procedure, inserting every 1 bit from $y$ as an item from $R$ into the summary with weight 1. Bob then builds a new summary of a join between two empty relations, and compares the two summaries. If the join of $L$ and $R$ is empty, then the two summaries should report that the results are identical; however, if there is anything in the join of $L$ and $R$, then it should report that they are different. But this corresponds exactly to the cases $x \cdot y = 0$ and $x \cdot y > 0$. Therefore, any summary to solve this problem must have size $\Omega(n)$. □

This shows that even if we wish to verify a simplified version of the query, to determine whether the join is empty or not, it still requires a summary whose size is linear in the number of groups.

## 7. EMPIRICAL EVALUATION

In this section we evaluate the performance of the proposed synopses over two real data streams. The experimental study demonstrates that our synopses: 1. use very small space; 2. support fast updates; 3. have very high accuracy; 4. support multiple queries; and 5. are easy to implement.

### 7.1 Experimental Setup

Our synopses are implemented using GNU C++ and the GNU GMP extension which provides arbitrary precision arithmetic, useful for operating on numbers with an arbitrary number of bits. This allows us to support long group id's. For instance, the example query (*) has group id's with 64 bits, and the prime $p$ needs to be chosen such that we need integer arithmetic over a field with 65 bits. For queries involving shorter id's, one can alternatively use native 32-bit or 64-bit integers, which will make the implementation even simpler and more efficient. The experiments were run on an Intel Pentium 2.8GHz CPU with 512KB L2 cache and 512MB of main memory.

We used two real data sets for our experiments. The *IP traces (IPs)* data stream is collected over the AT&T backbone network; each tuple is a TCP/IP packet header. Here, we are interested in analyzing the source IP, destination IP, and packet size header fields. The data set consists of a segment of one day traffic and has 100 million packets. The *World Cup (WC)* data stream [Arlitt and Jin ] consists of web server logs for the 1998 Soccer World Cup. Each record in the log contains several attributes such as a timestamp, a client id, a requested object id, a response size, etc. We used the request streams of days 46 and 47 that have about 100 millions records. Without loss of generality, unless otherwise stated, we perform the following default queries: For the IPs data set, we perform the example query (*) where the aggregate is either `COUNT` or `SUM`. For the WC data set, we perform `COUNT` or `SUM` queries on the packet size group-by client id/object id. Each client id, object id, IP address, the response size, or the packet size is a 32-bit integer. Thus, the group id is 64-bit long (by concatenating the two grouping attributes), meaning a potential group space of $n = 2^{64}$. The number of nonzero groups is of course far lower than $n$: WC has a total of 50 million nonzero groups and IPs has 7 million nonzero groups.

### 7.2 PIRS

We first present the experimental results on the basic PIRS synopsis. A very conservative upper bound for the total response size and packet size is $m = 10^{10} \ll n \approx 2 \times 10^{19}$ for all cases in our experiments. So from our analysis in Section 4, PIRS-1 is the better choice, and is thus used in our experiments. We pre-computed $p$ as the smallest prime above $2^{64}$ (which is 18446744073709551629) and used the same $p$ throughout this section. Thus, each word (storing $p, \alpha$, or $\mathcal{X}(\mathbf{v})$) occupies 9 bytes.

**Synopsis size.** As our analysis has pointed out, PIRS uses only 3 words, or 27 bytes for our queries. This is in contrast to the naïve solution of keeping the exact value for each nonzero group, which would require 600MB and 84MB for the WC and IPs data sets, respectively. Also keep in mind that this is the space usage

for a single query; much more space will be needed if multiple queries are to be supported. The small size of PIRS is particularly important because it implies not only a small memory requirement of the data owner, but also a much reduced network bandwidth consumption since the synopsis needs to be sent from the data owner to the client upon each verification request. Even if the data owner has a large memory to store all the groups and chooses to send a hash (e.g. SHA0 or SHA1) of the query result, which would solve the bandwidth problem, but still it faces a serious computation problem: Upon each request for verification from some client, all the groups need to be scanned once to compute the hash.

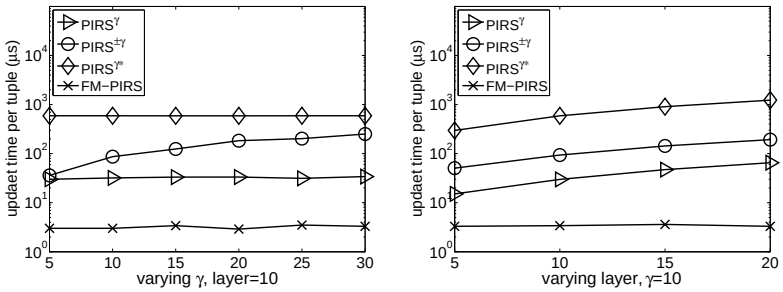
A related question is, if the query result size is so large (600MB and 84MB in our experiments), how does the server transmit the results to the client? Here we would like to remind the reader that queries posed on a data stream are mostly continuous, long-standing queries. Each client could register his/her queries with a server with potentially different quality-of-service agreements, requiring the server to send over the updated results, say, every 10 minutes or every time the results have changed by a certain amount. This way, the server only needs to send the deltas of the query results compared with the last transmission, reducing the bandwidth consumption significantly. To do so, however, the server needs to maintain a registration record on a per-client basis remembering the client's query specification, quality-of-service agreement, and all the changes to the query results since the last transmission. But all these components are necessary in any infrastructure that aims to provide continuous query services on data streams. Our techniques enable the migration of all these costly maintenance from the data owner to the server, which is exactly the goal of data outsourcing.

**Update cost.** PIRS has a low update cost which is crucial to any streaming application. The average per-tuple update cost is shown in Table I for count and sum queries on both WC and IPs. The update time for the two count queries stays the same regardless of the data set, since an update always incurs one addition, one multiplication, and one modulo. The update cost for sum queries is higher, since we need $O(\log u)$ time for exponentiation. The cost on WC is slightly larger as its average $u$ is larger than that of IPs. Nevertheless, PIRS is still extremely fast in all cases, and is able to process more than $10^5$ tuples ($10^6$ tuples for count queries) per second.

| | WC | IPs |
|---|---|---|
| Count | 0.98 $\mu$s | 0.98 $\mu$s |
| Sum | 8.01 $\mu$s | 6.69 $\mu$s |

Table I. Average update time per tuple.

**Detection accuracy.** As guaranteed by the theoretical analysis, the probability of failure of PIRS-1 is $\delta \leq m/p$, which is at most $0.5 \times 10^{-9}$. This is practically zero. Note that our estimate of $m$ is very conservative; the actual $\delta$ is much smaller. We generated $100,000$ random attacks and, not surprisingly, PIRS identified all of them.




(a) Update cost per tuple vs. $\gamma$. (b) Update cost per tuple vs. layer.

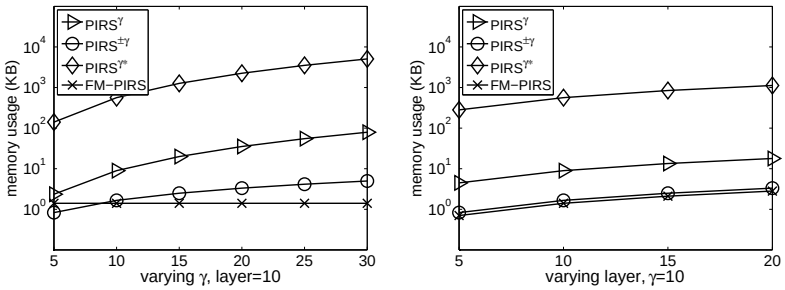
Fig. 4. PIRS$^{\gamma}$, PIRS$^{\pm\gamma}$, PIRS$^{\gamma*}$, FM-PIRS: update cost.

### 7.3 PIRS$^{\gamma}$, PIRS$^{\gamma*}$, PIRS$^{\pm\gamma}$, and FM-PIRS

Next, we present the experimental results on the four extended synopses: PIRS$^{\gamma}$, PIRS$^{\pm\gamma}$, PIRS$^{\gamma*}$, and FM-PIRS. These synopses are designed to check if the query results contain less than a specified number of errors. Specifically, PIRS$^{\gamma}$ raises an alarm when there are $\gamma$ or more errors; PIRS$^{\gamma*}$ is a strengthened version of PIRS$^{\gamma}$ that in addition to the alarm, also identifies and corrects the errors; PIRS$^{\pm\gamma}$ allows some approximation in terms of $\gamma$ but has a smaller size; and FM-PIRS can be used to estimate the number of errors regardless of $\gamma$. Note that the naïve solution of keeping the exact $\mathbf{v}$ and sending its hash to the client does not even work for the problems that these PIRS variants are designed to solve. To be able to do so, the naïve algorithm has to transmit the entire $\mathbf{v}$ to the client, which is certainly not a viable solution. In the following we only present the experimental results on the IPs data set with the count query. Similar patterns have been observed on the WC data set.

Recall that all these synopses exhibit a similar structure with multiple layers and each layer consists of multiple buckets, but how the buckets are configured is different for different synopses. In PIRS$^{\gamma}$, each layer contains $O(\gamma^2)$ buckets, whereas PIRS$^{\pm\gamma}$ has only $O(\gamma)$ buckets. For PIRS$^{\gamma*}$, the number of buckets in each layer is the same as in PIRS$^{\gamma}$, but each bucket in addition contains a two dimensional array of PIRS for identifying the id's of the erroneous groups. For FM-PIRS, the number of buckets (say $b$) is determined by the potential number ($|E(\mathbf{w}, \mathbf{v})|$) of erroneous groups, i.e., we need to ensure that $2^b > |E(\mathbf{w}, \mathbf{v})|$. For our purpose, setting $b = 16$ could handle up to $2^{16}$ faulty groups.

**Update cost.** In this set of experiments we study the update costs of PIRS$^{\gamma}$, PIRS$^{\pm\gamma}$, PIRS$^{\gamma*}$ and FM-PIRS. Except PIRS$^{\pm\gamma}$, all other synopses have an update cost that is independent of $\gamma$, as illustrated in Figure 4(a). PIRS$^{\gamma}$ maps an incoming tuple to a bucket in each layer based on its group id and updates that bucket accordingly. The mapping is performed via a pairwise independent hash function, hence the whole process does not depend on $\gamma$. PIRS$^{\gamma*}$ follows a similar step, with an additional overhead of updating the associated array of PIRS for the selected bucket. FM-PIRS, as confirmed in the experiments, is very cheap to maintain. Remember that it randomly selects a single layer and then updates one bucket in




(a) Memory usage vs. $\gamma$. (b) Memory usage vs. layer.

Fig. 5. PIRS$^{\gamma}$, PIRS$^{\pm\gamma}$, PIRS$^{\gamma*}$, FM-PIRS: memory usage.

this layer following a geometric distribution. Hence, it is independent of both $\gamma$ and $\ell$, the number of layers (see also Figure 4(b)). Finally, PIRS$^{\pm\gamma}$ updates one bucket in each layer. However, the selection of the bucket is achieved with the help of a $\gamma$-wise independent hash function, which has an update cost linear in $\gamma$. All these trends have been demonstrated in Figure 4(a). As far as the number of layers $\ell$ is concerned, all synopses except the FM-PIRS have a linear update cost w.r.t $\ell$ as shown in Figure 4(b). Nevertheless, all synopses are able to process each tuple in the order of ms. FM-PIRS in fact only takes approximately 1 $\mu$s to finish an update. Therefore, these synopses could process from thousands to hundreds of thousands tuples per second.

**Synopsis size.** Figure 5 shows the space usage of these synopses. Not surprisingly, PIRS$^{\gamma*}$ is the largest one as it offers the most powerful functionality of actually identifying and correcting the faulty groups. Following our discussion on configuring various synopses, it is not hard to explain the trends exhibited from Figure 5(a) and 5(b). Essentially, FM-PIRS is independent of $\gamma$. PIRS$^{\gamma}$ has a quadratic space dependence on $\gamma$ and PIRS$^{\pm\gamma}$'s space cost is linear to $\gamma$. As for the number of layers, all synopses follow a linear relationship with $\ell$. Finally, it is interesting to observe from Figure 5(a) that FM-PIRS has larger space usage than PIRS$^{\pm\gamma}$ when $\gamma$ is small, but is preferred for large $\gamma$'s.

**Space/time trade-offs: exploiting locality.** In many practical situations, data streams tend to exhibit a large degree of locality [Li et al. 2006]. Simply put, updates to $\mathbf{v}$ tend to cluster to the same components. In this case, it is possible to exploit space/time trade-offs. We allocate a small buffer used for storing exact aggregate results for a small number of groups. With data locality, a large portion of updates will hit the buffer. Whenever the buffer is full and a new group needs to be inserted, a victim is evicted from the buffer using the simple *least recently used (LRU)* policy. Only then does the evicted group update PIRS, using the overall aggregate value computed within the buffer. We flush the buffer to update PIRS whenever a verification is required. Since we are aggregating the incoming updates in the buffer and updating the synopsis in bulk, we incur a smaller amortized update processing cost per tuple. A simple LRU buffer has been added to the system and its effect on the update costs for the four synopses is reported in Figure 6 with

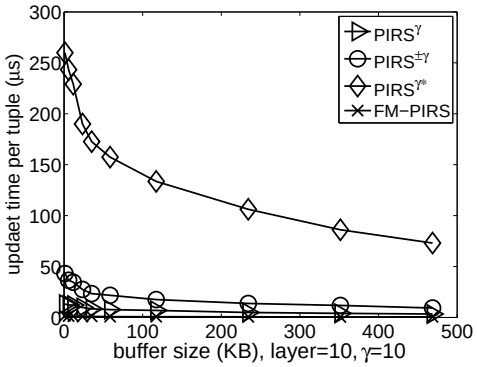
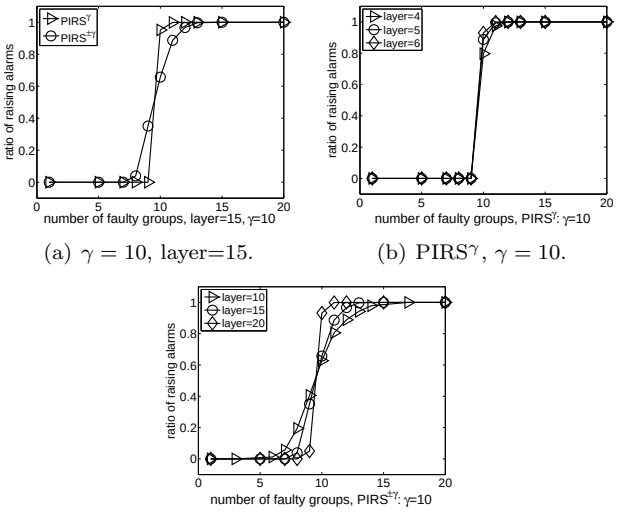



Fig. 6. Space-time tradeoff.



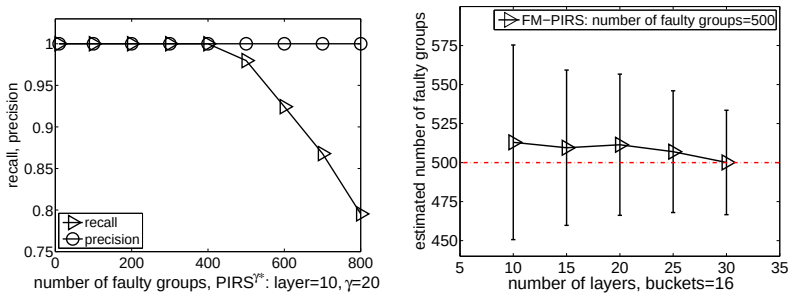

(a) $\gamma = 10$, layer=15. (b) PIRS$^{\gamma}$, $\gamma = 10$.

(c) PIRS$^{\pm\gamma}$, $\gamma = 10$.

Fig. 7. Detection with tolerance for limited number of errors.

$\ell = 10$ and $\gamma = 10$. All synopses demonstrate very similar trends. As the figure indicates, a very small buffer (roughly 500 KB) that fits into the cache is able to reduce the update cost by an order of magnitude.

**Detection accuracy.** We first concentrate on PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$ for the simple purpose of raising alarms when the number of faulty groups exceeds the threshold $\gamma$. We observed that both synopses can achieve excellent detection accuracy as the theoretical analysis suggests. All results reported here are the average ratios obtained from $100,000$ rounds of attacks. Since the detection mechanism of the synopses does not depend on the data characteristics, both data sets give similar results, so again we show plots for the IPs data only. Figure 7(a) shows the ratios of raising alarms versus the number of actual inconsistent groups, with $\gamma = 10$ and




(a) Recall and Precision of PIRS$^{\gamma*}$. (b) Accuracy of FM-PIRS.

Fig. 8. Effectiveness of PIRS$^{\gamma*}$ and FM-PIRS.

10 layers. As expected, PIRS$^{\gamma}$ has no false positives and almost no false negatives; only very few false negatives are observed with 10 and 11 actual inconsistent groups. On the other hand, PIRS$^{\pm\gamma}$ has a transition region around $\gamma$ and it does have false positives. Nevertheless, the transition region is sharp and once the actual number of inconsistent groups is slightly away from $\gamma$, both false positives and negatives reduce to zero. We have also studied the impact of the number of layers on the detection accuracy. Our theoretical analysis gives provable bounds. For example with PIRS$^{\gamma}$ the probability of missing an alarm is at most $1/2^{\ell}$ (for $\ell$ layers). In practice, the probability is expected to be even smaller. We repeated the same experiments using different layers, and Figure 7(b) reports the result for PIRS$^{\gamma}$. With fewer layers (4–6) it still achieves excellent detection accuracy. Only when the number of inconsistent groups is close to $\gamma$, a small drop in the detection ratio is observed. Figure 7(c) reports the same experiment for PIRS$^{\pm\gamma}$ with layers from 10 to 20. Having a smaller number of layers enlarges the transition region and larger number of layers sharpens it. Outside this region, 100% detection ratio is always guaranteed. Other experiments performed over different values of $\gamma$ led to similar behavior.

There are two limitations with the above synopses. They could not identify the exact set of erroneous groups in the presence of errors. Also, their update cost and space usage both depend on (at least linear to) $\gamma$, which prevents the application of them for large $\gamma$ values. To that end, we have designed PIRS$^{\gamma*}$ and FM-PIRS respectively. To measure the effectiveness of identifying the faulty groups, we utilize two common metrics, namely, *recall* and *precision*. Recall refers to the percentage of truly faulty groups being successfully retrieved. Precision refers to the percentage of true faulty groups identified among the reported ones. Following our analysis in Section 5.2, the precision of PIRS$^{\gamma*}$ is always 1. The recall of PIRS$^{\gamma*}$ depends on the probability that the same faulty group has a collision with at least one other faulty group in every layer. Due to the independence among layers, this event happens with really low chances in practice. Hence, we expect a high recall rate as well. This has been confirmed in Figure 8(a) where PIRS$^{\gamma*}$ is configured to raise alarms when there are more than 20 faulty groups. Even with such a small $\gamma$ value, PIRS$^{\gamma*}$ can perfectly identify the IDs of faulty groups for up to 500. Only after then, the recall starts to drop. This generally agrees with our analysis in

| # queries | 5 | 10 | 15 | 20 |
|---|---|---|---|---|
| update time ($\mu$s) | 5.0 | 9.9 | 14.9 | 19.8 |
| memory usage (bytes) | 27 | 27 | 27 | 27 |

Table II. Update time and memory usage of PIRS for multiple queries.

Theorem 5.4.

To alleviate the relatively high update cost of PIRS$^{\gamma*}$, FM-PIRS could be used to estimate the number of faulty groups from the server's answer. Previous experiments have already shown convincing evidence that this is an extremely cheap structure to maintain and it is independent of $\gamma$. We tested it with 500 number of faulty groups and report the average result over 100 runs in Figure 8(b). The error bars represent the standard deviation of these runs. Clearly, increasing the number of layers improves the accuracy of the estimation (reducing the standard deviation and closing the gap between the average value and the true value). With a small number of layers, say $\ell = 20$, FM-PIRS delivers a satisfactory estimation of the number of faulty groups.

### 7.4 Multiple Queries

Our final set of experiments investigates the effect of multiple, simultaneous queries. Without loss of generality, we simply execute the same query a number of times. Note that the same grouping attributes with different query ids are considered as different groups. We tested with 5, 10, 15, and 20 queries in the experiments. Note that on the WC data set, the exact solution would use 600MB for each query, hence 12GB if there are 20 queries. Following the analysis in Section 4.4, our synopses naturally support multiple queries and still have the same memory usage as if there were only one query. Nevertheless, the update costs of all synopses increase linearly with the number of queries. In Table II we report the update time and memory usage for PIRS; similar trends were observed for PIRS$^{\gamma}$, PIRS$^{\pm\gamma}$, PIRS$^{\gamma*}$ and FM-PIRS.

In terms of verification power, all of the synopses are exactly the same as previously reported, by treating these multiple queries as one single combined query. More precisely, PIRS will raise an alarm as long as one of the queries contains an error, the $\gamma$ in PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$ will become the threshold on the total number of errors in all the queried combined, and FM-PIRS will estimate the total number of errors. Note that such a combination does increase the verification granularity, as we do not distinguish errors from different queries. If a finer granularity is desired, a separate synopsis is still required for each query. The only exception is PIRS$^{\gamma*}$. Since it finds and corrects all the errors (provided that there are no more than $\gamma$ errors in total), we can of course also identify the queries that are wrong, by using only one PIRS$^{\gamma*}$ on the combined query.

## 8. RELATED WORK

**Authentication in outsourced data.** There is considerable work on authenticating query execution in an outsourced database setting [Hacigumus et al. 2002; Devanbu et al. 2003; Martel et al. 2004; Bertino et al. 2004; Pang and Tan 2004;

Pang et al. 2005; Li et al. 2006]. Here, the client queries the publisher's data through a third party, namely the server, and the goal is to design efficient solutions to enable the client to authenticate the query results. Most of these works rely on cryptographic primitives such as digital signatures and Merkle hash trees to design efficient index structures built by the data publisher to enable authentication of query results. All these techniques apply only to offline settings and do not scale to online, one-pass streaming scenarios. More recently, there have been a few works dealing with query authentication on streaming data [Li et al. 2007; Papadopoulos et al. 2007]. However, these papers only consider selection and aggregation queries, and cannot support `GROUP BY` queries that we study in this paper. Moreover, all the techniques in these papers are still basically variants and combinations of digital signatures and Merkle hash trees, so technique-wise, they are very similar to the previous works that deal with non-streaming settings.

**Security guarantees.** PIRS deploys completely different techniques than all previous works on query authentication, and there are also some subtle yet interesting differences in the security guarantees provided by PIRS and the authentication techniques. Since all the authentication techniques rely on cryptographic primitives, the security is thus built upon the *computational infeasibility* for the attacker (the third-party server in our case) to break the system. Theoretically speaking, if the server has infinite computing power, she/he can always in principle successfully return some wrong query results to the client without being detected. The security guarantee of PIRS, on the other hand, is *probabilistic impossibility*. More precisely, if the server returns *any* wrong query answer, the probability that the client catches the attack is almost 1, say $1 - 10^{-9}$ in the setting of our experiments, i.e., it is virtually impossible for the server to attack successfully. Furthermore, the guarantee holds even assuming that the server possesses infinite computing power. So theoretically speaking, the security guarantee provided by PIRS is in this sense stronger than that provided by the authentication techniques. This is also the reason why we choose the term "verification" instead of "authentication", so as to differentiate PIRS from the authentication techniques.

**Comparison with sketches.** PIRS is a way of summarizing the underlying data streams. In that respect our work is related to the line of work on sketching techniques [Alon et al. 1996; Manku and Motwani 2002; Babcock et al. 2003; Cormode and Muthukrishnan 2005; Flajolet and Martin 1985; Ganguly et al. 2004]. Indeed, many of the verification problems we study have a similar formulation to problems studied in the streaming world. However, the precise specification is different, and so (as discussed in Section 3), applying sketching techniques to our problems yields results which either do not give comparable guarantees, or else require asymptotically and practically more space in order to give similar guarantees for the verification problem.

**Cryptographic approaches.** Another possible way for solving the CQV problem is to *incremental cryptography* [Bellare et al. 1994]. The idea is that by viewing $\mathbf{v}$ as a message, the client could compute an authenticated signature $\sigma(\mathbf{v})$ and any alteration to $\mathbf{v}$ will be detected. Now the challenge is how to perform incremental updates using $\sigma(\mathbf{v})$ alone, without storing $\mathbf{v}$, i.e., in the present setting the mes-

sage $\mathbf{v}$ is constantly updated. Cryptography researchers have devoted considerable effort for this problem, resulting in techniques such as incremental signature and incremental MAC [Bellare et al. 1994; Bellare et al. 1995]. However, these techniques only support updates for *block edit operations* such as insert and delete, i.e., by viewing $\mathbf{v}$ as blocks of bits, they are able to compute $\sigma(\mathbf{v}')$ using $\sigma(\mathbf{v})$ alone if $\mathbf{v}'$ is obtained by inserting a new block (or deleting an old block) into (from) $\mathbf{v}$. However, in our setting the update operation is arithmetic: $v_i^\tau = v_i^{\tau-1} + u^\tau$, which cannot be handled by simply deleting the old entry followed by inserting the new one, since we only have $u^\tau$ as input, and no access to either $v_i^{\tau-1}$ or $v_i^\tau$. Hence, such cryptographic approaches are inapplicable. Moreover, the techniques we propose are much more lightweight and simple to implement than complex schemes based on cryptographic primitives.

**Fingerprinting techniques.** Verifying the identity of polynomials is a fingerprinting technique [Motwani and Raghavan 1995]. Fingerprinting is a method for efficient, probabilistic checking of equality between two elements $x, y$ from a large universe $U$. Instead of testing the equality using $x, y$ deterministically with complexity at least $\log |U|$, a probabilistic approach is to pick a random mapping from $U$ to a significantly smaller universe $V$ such that with high probability $x, y$ are identical if and only if their images in $V$ are identical. The images of $x$ and $y$ are their fingerprints and their equality can be verified in $\log |V|$ time. Fingerprint techniques generally employ algebraic techniques combined with randomization. Classical examples include verifying univariate polynomial multiplication [Freivalds 1979], multivariate polynomial identities [Schwartz 1980], and verifying equality of strings [Motwani and Raghavan 1995]. We refer readers for an excellent discussion on these problems to [Motwani and Raghavan 1995]. Although the general technique of polynomial identity verification is known, our use of it in the setting of query verification on data streams appears to be new.

**Other related works.** Other security issues for secure computation/querying over streaming and sensor data have also started to receive attention recently. For example, orthogonal to our problem, [Chan et al. 2006; Garofalakis et al. 2007] have studied the problem of secure in-network aggregation for aggregation queries in sensor networks. Both works utilize cryptographic tools, such as digital signatures, as building blocks for their algorithms and assume the man-in-middle attack model. Hence, they are fundamentally different from our work. Nevertheless, they attest to the fact that secure computation of aggregation queries has a profound impact in many real applications.

**Comparison with prior publication.** This article is based on an earlier conference paper [Anonymous ]. In addition to providing the full analysis of the techniques proposed, this article also makes several new nontrivial contributions extending both the scope and techniques of [Anonymous ]. (1) In [Anonymous ] the model consists of only two parties: the server and the client. In this article we extend all our techniques to the more popular three-party outsourcing model that has been adopted by most existing works in this area. (2) We provide a rigorous analysis on the worst-case compromise of the security guarantee when PIRS is under multiple attacks. (3) We strengthen PIRS$^\gamma$ to a new synopsis, PIRS$^{\gamma*}$, which not only tells

the client *whether* the number of errors is below some threshold, but also *where* they are and *what* their true values should be. This is crucial in many critical applications, in which the clients demand complete correctness of the query results. With PIRS$^{\gamma*}$, we can identify and correct these errors (provided that there are no more than $\gamma$ errors), giving the client a complete and untainted query result. At the same time, we also allow some slack for the server so that it can employ load shedding strategies to cope with high load. (4) We design a new synopsis, PIRS-FM, that is able to estimate the number of errors in the query result. The space and time complexities of PIRS-FM are both much better than PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$. Our experimental results show that PIRS-FM significantly improves upon PIRS$^{\gamma}$ and PIRS$^{\pm\gamma}$ in terms of space and time, while maintaining roughly the same security guarantee. (5) We also have considered many related queries such as joins, and given lower bounds showing that verifying their correctness with small synopses is provably difficult.

## 9. CONCLUSION

The present work studies the problem of verifying "`GROUP BY, COUNT`" and "`GROUP BY, SUM`" queries on outsourced data streams. All the existing works on query authentication of outsourced databases are based on cryptographic primitives, and it seems inherently difficult to apply these techniques to such group-by queries. Our solutions, on the other hand, are based on verifying the identity of polynomials, hence fundamentally different from the existing query authentication framework. It is imaginable that our techniques can be applied to other query verification problems on outsourced databases that cannot be solved using existing techniques.

## REFERENCES


Abadi, D., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., and Zdonik, S. 2005. The Design of the Borealis Stream Processing Engine. In *Proc. of Biennial Conference on Innovative Data Systems Research*.

Alon, N., Matias, Y., and Szegedy, M. 1996. The space complexity of approximating the frequency moments. In *Proc. ACM Symposium on Theory of Computation*. 20–29.

Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., and Widom, J. 2003. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin 26,* 1, 19–26.

Arasu, A. and Manku, G. S. 2004. Approximate counts and quantiles over sliding windows. In *Proc. ACM Symposium on Principles of Database Systems*. 286–296.

Arlitt, M. and Jin, T. http:// www.acm.org/ sigcomm/ITA/. ITA, 1998 World Cup Web Site Access Logs.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. 2002. Models and issues in data stream systems. In *Proc. ACM Symposium on Principles of Database Systems*.

Babcock, B., Datar, M., and Motwani, R. 2004. Load shedding for aggregation queries over data streams. In *Proc. IEEE International Conference on Data Engineering*. 350–361.

Babcock, B., M.Datar, Motwani, R., and O'Callaghan, L. 2003. Maintaining variance and k-medians over data stream windows. In *Proc. ACM Symposium on Principles of Database Systems*. 234–243.

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

Bellare, M., Goldreich, O., and Goldwasser, S. 1994. Incremental cryptography: The case of hashing and signing. In *Proc. of Advances in Cryptology (CRYPTO)*. 216–233.

Bellare, M., Guerin, R., and Rogaway, P. 1995. Xor macs: New methods for message authentication using finite pseudorandom functions. In *Proc. of Advances in Cryptology (CRYPTO)*. 15–28.

Bertino, E., Carminati, B., Ferrari, E., Thuraisingham, B., and Gupta, A. 2004. Selective and authentic third-party distribution of XML documents. *IEEE Transactions on Knowledge and Data Engineering 16,* 10, 1263–1278.

Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., and Zdonik, S. 2003. Monitoring streams—a new class of data management applications. In *Proc. International Conference on Very Large Databases*. 215–226.

Chan, H., Perrig, A., and Song, D. 2006. Secure hierarchical in-network aggregation in sensor networks. In *Proc. of the ACM Conference on Computer and Communications Security*. 278–287.

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, D., Raman, V., Reiss, F., and Shah, M. A. 2003. TelegraphCQ: continuous dataflow processing for an uncertain world. In *Proc. of Biennial Conference on Innovative Data Systems Research*.

Cormode, G. and Muthukrishnan, S. 2003. What's hot and what's not: tracking most frequent items dynamically. In *Proc. ACM Symposium on Principles of Database Systems*. 296–306.

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

Cormode, G., Muthukrishnan, S., and Rozenbaum, I. 2005. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proc. International Conference on Very Large Databases*. 25–36.

Cranor, C., Johnson, T., Spatscheck, O., and Shkapenyuk, V. 2003. Gigascope: A stream database for internet databases. In *Proc. ACM SIGMOD International Conference on Management of Data*. 647–651.

Datar, M., Gionis, A., Indyk, P., and Motwani, R. 2002. Maintaining stream statistics over sliding windows. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*. 635–644.

Devanbu, P., Gertz, M., Martel, C., and Stubblebine, S. 2003. Authentic data publication over the internet. *Journal of Computer Security 11,* 3, 291–314.

Flajolet, P. and Martin, G. N. 1985. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences 31,* 2, 182–209.

Freivalds, R. 1979. Fast probabilistic algorithms. In *Proc. International Symposium on Mathematical Foundations of Computer Science*. 57–69.

Ganguly, S., Garofalakis, M., and Rastogi, R. 2004. Tracking set-expression cardinalities over continuous update streams. *The VLDB Journal 13,* 4, 354–369.

Garofalakis, M., Hellerstein, J. M., and Maniatis, P. 2007. Proof sketches: Verifiable in-network aggregation. In *Proc. IEEE International Conference on Data Engineering*.

Gilbert, A. C., Kotidis, Y., Muthukrishnan, S., and Strauss, M. 2002. How to summarize the universe: Dynamic maintenance of quantiles. In *Proc. International Conference on Very Large Databases*. 454–465.

Greenwald, M. and Khanna, S. 2001. Space-efficient online computation of quantile summaries. In *Proc. ACM SIGMOD International Conference on Management of Data*. 58–66.

Hacigumus, H., Iyer, B. R., and Mehrotra, S. 2002. Providing database as a service. In *Proc. IEEE International Conference on Data Engineering*. 29–40.

Hammad, M. A., Mokbel, M. F., Ali, M. H., Aref, W. G., Catlin, A. C., Elmagarmid, A. K., Eltabakh, M., Elfeky, M. G., Ghanem, T. M., Gwadera, R., Ilyas, I. F., Marzouk, M., and Xiong, X. 2004. Nile: A query processing engine for data streams. In *Proc. IEEE International Conference on Data Engineering*. 851.

Indyk, P. and Woodruff, D. 2005. Optimal approximations of the frequency moments of data streams. In *Proc. ACM Symposium on Theory of Computation*.

Karp, R. M., Shenker, S., and Papadimitriou, C. H. 2003. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems 28,* 1, 51–55.

Knuth, D. E. 1997. *The Art of Computer Programming*. Addison-Wesley.

Kushilevitz, E. and Nisan, N. 1997. *Communication Complexity*. Cambridge University Press.

Li, F., Chang, Q., Kollios, G., and Bestavros, A. 2006. Characterizing and exploiting reference locality in data stream applications. In *Proc. IEEE International Conference on Data Engineering*.

Li, F., Hadjieleftheriou, M., Kollios, G., and Reyzin, L. 2006. Dynamic authenticated index structures for outsourced databases. In *Proc. ACM SIGMOD International Conference on Management of Data*. 121–132.

Li, F., Yi, K., Hadjieleftheriou, M., and Kollios, G. 2007. Proof-infused streams: Enabling authentication of sliding window queries on streams. In *Proc. International Conference on Very Large Databases*.

Manku, G. S. and Motwani, R. 2002. Approximate Frequency Counts over Data Streams. In *Proc. International Conference on Very Large Databases*. 346–357.

Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., and Stubblebine, S. 2004. A general model for authenticated data structures. *Algorithmica 39,* 1, 21–41.

Metwally, A., Agrawal, D., and Abbadi, A. E. 2006. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems 31,* 3, 1095–1133.

Motwani, R. and Raghavan, P. 1995. *Randomized Algorithms*. Cambridge University Press.

Muthukrishnan, S. 2003. Data streams: algorithms and applications.

Nagell, T. 1981. *Introduction to Number Theory*, Second ed. Chelsea Publishing Company.

Pang, H., Jain, A., Ramamritham, K., and Tan, K.-L. 2005. Verifying completeness of relational query results in data publishing. In *Proc. ACM SIGMOD International Conference on Management of Data*. 407–418.

Pang, H. and Tan, K.-L. 2004. Authenticating query results in edge computing. In *Proc. IEEE International Conference on Data Engineering*. 560–571.

Papadopoulos, S., Yang, Y., and Papadias, D. 2007. CADS: Continuous authentication on data streams. In *Proc. International Conference on Very Large Databases*. 135–146.

Rusu, F. and Dobra, A. 2007. Pseudo-random number generation for sketch-based estimations. *ACM Transactions on Database Systems 32,* 2.

Schwartz, J. T. 1980. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM 27,* 4, 701–717.

Tatbul, N., Cetintemel, U., Zdonik, S., Cherniack, M., and Stonebraker, M. 2003. Load shedding in a data stream manager. In *Proc. International Conference on Very Large Databases*. 309–320.

Tatbul, N. and Zdonik, S. 2006. Window-aware load shedding for aggregation queries over data streams. In *Proc. International Conference on Very Large Databases*. 799–810.

Thorup, M. 2000. Even strongly universal hashing is pretty fast. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*.

Wegman, M. N. and Carter, J. L. 1981. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences 22,* 3.

Zhang, R., Koudas, N., Ooi, B. C., and Srivastava, D. 2005. Multiple aggregations over data streams. In *Proc. ACM SIGMOD International Conference on Management of Data*. 299–310.

Zhao, Q., Xu, J., and Liu, Z. 2006. Design of a novel statistics counter architecture with optimal space and time efficiency. In *ACM SIGMETRICS*. 323–334.

Anonymous. Omitted due to double-blind reviewing.