

# Randomized Synopses for Query Assurance on Data Streams

Ke Yi<sup>†</sup>    Feifei Li<sup>‡</sup>    Marios Hadjieleftheriou<sup>†</sup>    Divesh Srivastava<sup>†</sup>    George Kollios<sup>‡</sup>

<sup>†</sup>AT&T Labs, Inc.

{yike, marioh, divesh}@research.att.com

<sup>‡</sup>Department of Computer Science

Boston University

{lifeifei, gkollios}@cs.bu.edu

## Abstract

Due to the overwhelming flow of information in many data stream applications, many companies may not be willing to acquire the necessary resources for deploying a Data Stream Management System (DSMS), choosing, alternatively, to outsource the data stream and the desired computations to a third-party. But data outsourcing and remote computations intrinsically raise issues of trust, making outsourced query assurance on data streams a problem with important practical implications. Consider a setting where a continuous “GROUP BY, SUM” query is processed using a remote, untrusted server. A client with limited processing capabilities observing exactly the same stream as the server, registers the query on the server’s DSMS and receives results upon request. The client wants to verify the integrity of the results using significantly fewer resources than evaluating the query locally. Towards that goal, we propose a probabilistic verification algorithm for selection and aggregate/group-by queries, that uses constant space irrespective of the result-set size, has low update cost per stream element, and can have arbitrarily small probability of failure. We generalize this algorithm to allow some tolerance on the number of erroneous groups detected, in order to support semantic load shedding on the server. We also discuss the hardness of supporting random load shedding. Finally, we implement our techniques and perform an empirical evaluation using live network traffic.

## 1 Introduction

A large number of commercial Data Stream Management Systems (DSMS) have been developed recently [16, 25, 4, 19, 14, 2], mainly driven by the continuous nature of the data being generated by a variety of real-world applications, like telephony and networking. Many companies deploy such DSMSs for the purpose of gathering inval-

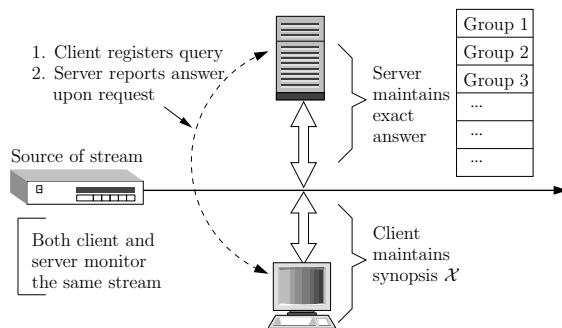


Figure 1. System architecture.

able statistics about their day-to-day operations. Not surprisingly, due to the overwhelming data flow observed in most data streams, some companies do not possess and are not willing to acquire the necessary resources for deploying a DSMS. Hence, in these cases outsourcing the data stream and the desired computations to a third-party is the only alternative. For example, an Internet Service Provider (e.g. Verizon) outsources the task of performing essential network traffic analysis to another company (e.g. AT&T) that already poses the appropriate tools for that purpose (e.g. Gigascope). Clearly, data outsourcing and remote computations intrinsically raise issues of trust. As a consequence, outsourced query assurance on data streams is a problem with important practical implications. This problem has been studied before in the context of static outsourced data [37]. To the best of our knowledge, this is the first work to address query assurance on data streams.

Consider a setting where continuous queries on a data stream are processed using a remote, untrusted server (that can be compromised, malicious, running faulty software, etc). A client with limited processing capabilities observing exactly the same input as the server, registers queries on the DSMS of the server and receives results upon request (Figure 1). Assuming that the server charges clients according

to the computation resources consumed and the volume of traffic processed for answering the queries, the former has an incentive to deceive the latter for increased profit. Furthermore, the server might have a competing interest to provide fraudulent answers to a particular client. Hence, a passive malicious server can 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 [39, 5, 8, 38].

Ideally, the client should be able to verify the integrity of the computations performed by the server using significantly fewer resources than evaluating the queries locally. Moreover, 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. To the best of our knowledge, this is the first work to address outsourced query assurance on streams, introducing a novel problem and proposing efficient solutions for a wide range of queries and failure models.

The present work concentrates on selection queries and aggregate queries, like sum and count. We develop solutions for verifying such aggregates on any type of grouping imposed on the input data (e.g., as a `GROUP BY` clause in standard SQL). 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 need to design techniques that can tolerate a small number of inconsistent answers while guaranteeing that the rest are correct. We also discuss the hardness of supporting *random load shedding*, where *small* errors are allowed for a *wide range* of answers.

Clearly, if a client wants to verify the query results with absolute confidence the only solution is to compute the answers exactly, which obviates the need of outsourcing. Hence, we investigate high-confidence probabilistic solutions and develop verification algorithms that significantly reduce resource consumption.

Towards that goal the contributions of this work are: 1. A probabilistic synopsis that raises an alarm with very high confidence if there exists at least one error in the query results. This algorithm has constant space cost (three words) and low processing cost per update ( $O(1)$  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), for an arbitrarily small probability of failure  $\delta$  (as long as  $n/\delta$  fits in one word); 2. A theoretical analysis of the algorithm that proves its space optimality on the bits level; 3. A strong result stating that the same synopsis can be used for verifying multiple simultaneous queries using space equal to that for a single query, for queries with the same aggregate attribute but different selections/groupings; 4. A generalization of this algorithm for raising alarms when the number of errors exceeds a predefined threshold, e.g., when semantic load shedding needs to be supported; 5. A discussion of the difficulty behind supporting random load shedding and some simple heuristics; 6. Finally, an extensive empirical evaluation of the techniques 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 AGG(A_1), ..., AGG(A_N) FROM T
WHERE ... GROUP BY G_1, ..., G_M
```

This is a rather general form of SQL queries, as any “SELECT, FROM, WHERE” query can be written in a `GROUP BY` aggregate form by using the primary key of the input relation  $T$  as the `GROUP BY` attribute  $G$  (a `GROUP BY` clause that groups every tuple by itself). For example “SELECT A,B FROM T WHERE B>10” can be written as “SELECT SUM(A),SUM(B) FROM T WHERE B>10 GROUP BY PK”. 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 numerous times ([43] and related work therein). For example, a query that appears frequently in network monitoring applications is the following:

```
SELECT 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. Any solution to the above problem naturally supports the tumbling window semantics. Our proposed scheme, as we will see shortly, has the appealing feature of being easily extended to the sliding window semantics, which will be discussed in Section 7.

**Data Stream Model.** Following the example query of the previous section, 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  $[n] = \{1, \dots, n\}$ . Denote this data stream as  $\mathcal{S}$  and its  $\tau$ -th tuple as  $s^\tau = (i, u^\tau)$ , an update of amount  $u$  to the  $i$ th group. Formally, the query answer can be expressed as a dynamic vector of non-negative integers  $\mathbf{v}^\tau = [v_1^\tau, \dots, v_n^\tau] \in \mathbb{N}^n$ , 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 [31], and our algorithms are designed to work under this model. The readers are referred to two excellent papers [31, 7] for detailed discussions of data stream models.

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

**Definition 1** Given a data stream  $\mathcal{S}$ , a continuous query  $\mathcal{Q}$  and a user defined parameter  $\delta \in (0, \frac{1}{2})$ , build 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 maintained by the client for verifying vector  $\mathbf{v}$ .

With this definition the synopsis raises 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** For any  $\mathbf{w}, \mathbf{v} \in \mathbb{Z}^n$ , let  $E(\mathbf{w}, \mathbf{v}) = \{i \mid w_i \neq v_i\}$ . Then  $\mathbf{w} \neq_\gamma \mathbf{v}$  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  $\gamma \in \{1, \dots, n\}$  and  $\delta \in (0, \frac{1}{2})$ , build 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 sup-

port 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 3** For any  $\mathbf{w}, \mathbf{v} \in \mathbb{Z}^n$ , 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  $\delta \in (0, \frac{1}{2})$ , build 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  $\eta = 0$ .

We will work under the standard RAM model. Under this model, it is assumed that an addition, subtraction, multiplication, division, or taking mod involving two words takes 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.

**Paper outline.** First we discuss some simple intuitive solutions and show why they do not work (Section 3). Then, we present our solutions for the CQV (Section 4) and CQV $^\gamma$  (Section 5) problems and a discussion about the hardness of the CQV $^\eta$  problem (Section 6). We continue by presenting some useful extensions (Section 7), and an empirical evaluation of our techniques (Section 8). Finally we review some related work (Section 9) before concluding the paper.

### 3 Possible Solutions

This section presents some intuitive solutions and discusses why they do not solve 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 algorithm that always maintains  $\mathbf{v}$  exactly would use  $\Theta(|\mathbf{v}| \log m)$  bits of space. One might think of the following two simple solutions in order to reduce the high space requirement.

**Random sampling.** A first attempt is random sampling. Assuming a sampling rate  $r$ , the client randomly chooses  $rn$  groups. Clearly, with probability  $r$  this method will raise an alarm if  $\mathbf{w} \neq \mathbf{v}$ . 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 can at most reduce the space cost by a tiny fraction.

**Sketches.** Recent years have witnessed a large number of sketching techniques (e.g. [3, 18, 10, 21]) 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 their frequencies. Then we check if  $\mathcal{K}(\mathbf{v}) = \mathcal{K}(\mathbf{w})$ .

It is imaginable that such an approach would likely to catch most unintentional errors such as malfunctions of the server or package drops during communication. However, the fact that they are not designed for verification leaves them vulnerable under certain attacks. For instance, let us consider the two AMS sketches from the seminal work of Alon et al. [3]. Their  $F_0$  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 in 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.

Their  $F_2$  sketch computes the sum  $\sum_{i=1}^n h(i)v_i$ , where  $h : \{1, \dots, n\} \rightarrow \{-1, 1\}$  is chosen randomly from a family of 4-wise independent hash functions, and then repeat the process for a certain number of times independently. Below we will argue that for certain  $\mathbf{w} \neq \mathbf{v}$ , the chance that  $\sum_{i=1}^n h(i)v_i = \sum_{i=1}^n h(i)w_i$  is high, thus the sketch will miss  $\mathbf{w}$  unless many repetitions are used. This AMS sketch uses the BCH4 scheme (c.f. [34]) to construct a 4-wise independent random hash function  $f : [n] \rightarrow \{0, 1\}$ , and then set  $h(i) = 2f(i) - 1$ . Since  $\sum_{i=1}^n h(i)(v_i - w_i) = 2 \sum_{i=1}^n f(i)(v_i - w_i) - \sum_{i=1}^n (v_i - w_i)$ , it is sufficient to construct a  $\mathbf{w} \neq \mathbf{v}$  where  $\sum_{i=1}^n (v_i - w_i) = 0$ , such that  $\sum_{i=1}^n f(i)(v_i - w_i) = 0$  are 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  $x_1, \dots, x_r$  (resp.  $y_1, \dots, y_r$ ), 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  $i = 2^{r-1}$ ,  $w_i = v_i$ ; for even  $i \neq 2^{r-1}$ ,  $v_i - w_i = -1$  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

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

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 cannot catch this erroneous  $\mathbf{w}$ . This implies that at least  $\Omega(\log \frac{1}{\delta})$  independent copies of sketches is needed to drive down the overall failure probability to the  $\delta$ , giving a space complexity of  $\Omega(\log \frac{1}{\delta} \log n)$  and update time  $\Omega(\log \frac{1}{\delta})$ , with potentially large hidden constants. As the typical  $\delta$  values are very small in applications that require query assurance, these bounds are not satisfying. More importantly,  $1/2$  is merely lower bound on its failure probability as we do not know the exact probability it fails when  $S_0 \geq 2^{r-1}$ . For some specific values of  $n = 2^r - 1$ , we enumerated all values of  $S_0$  and  $S_1$  to compute the exact failure probabilities, shown in the following table.

$r$	3	4	5	6	7	8	9	10
fail prob	.75	.86	.84	.90	.94	.92	.93	.94

From the table we can see that the actual failure probability can be much larger than the lower bound of  $1/2$ , and it is not clear if it is bounded away from 1, so it is not safe at all to just use  $O(\log \frac{1}{\delta})$  copies. To summarize, although we have not theoretically ruled out the possibility that the AMS sketch may work, we can find cases where it is very likely to fail. Even if it could work (by proving an upper bound on the failure probability), it has to use at least  $\Omega(\log \frac{1}{\delta} \log n)$  bits of space with a large hidden constant.

Finally, the AMS sketches have many variants developed in recent years, but to our knowledge, none of them has the guarantee of assurance required by the CQV problem. In the next section, we present our solution, which does not only solve the CQV problem, but also uses much less space than all the known sketches.

## 4 PIRS: Polynomial Identity Random Synopsis

This section presents two synopses, called *Polynomial Identity Random Synopses* (PIRS) and denote by  $\mathcal{X}(\mathbf{v})$ , for solving the CQV problem (Definition 1). The synopses, as the name suggests, are 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 [22]. It has found applications in e.g. verifying matrix multiplications and pattern matching [30].

**PIRS-1.** Let  $p$  be some prime such that  $\max\{m/\delta, n\} <$

$p \leq 2 \max\{m/\delta, n\}$ . According to Bertrand's Postulate [32] such a  $p$  always exists. We will work in the field  $\mathbb{Z}_p$ , i.e., all additions 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

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

Having computed  $\mathcal{X}(\mathbf{v})$  and given any input  $\mathbf{w}$ , PIRS is able to check if  $\mathbf{w} = \mathbf{v}$  with high probability and without explicitly storing  $\mathbf{v}$ : We first check if  $\sum_{i=1}^n w_i > m$ , if so we reject  $\mathbf{w}$  immediately; otherwise we compute  $\mathcal{X}(\mathbf{w})$  as:

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

If  $\mathcal{X}(\mathbf{w}) = \mathcal{X}(\mathbf{v})$ , then we declare that  $\mathbf{w} = \mathbf{v}$ ; otherwise 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 1** *Given any  $\mathbf{w} \neq \mathbf{v}$ , PIRS raises an alarm with probability at least  $1 - \delta$ .*

*Proof:* Consider the polynomials  $f_{\mathbf{v}}(x) = (x - 1)^{v_1} (x - 2)^{v_2} \dots (x - n)^{v_n}$  and  $f_{\mathbf{w}}(x) = (x - 1)^{w_1} (x - 2)^{w_2} \dots (x - n)^{w_n}$ . 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  $\delta$ .  $\square$

Note that once we have chosen  $\alpha$ ,  $\mathcal{X}(\mathbf{v})$  can be incrementally maintained easily. For count queries, each tuple increments one of the  $v_i$ 's by one, so the update cost is constant (one addition 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 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.

**Theorem 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 modulo  $p$ , in  $O(\log p)$  time (using Euclid's gcd algorithm [26]), and then compute  $(\alpha - i)^{-1|u|}$ .

**PIRS-2.** When  $n \ll m$  we can actually do slightly better with 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

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

By considering the polynomial  $f_{\mathbf{v}}(x) = v_1 x + v_2 x^2 + \dots + v_n x^n$ , we can use the same proof to show that Theorem 1 still holds. Nevertheless, 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 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, while PIRS-1 has a better update time, we recommend using PIRS-1 unless  $n$  is small compared to  $m$  and typical  $u$ .

Another nice property 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** *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  $f : \mathcal{U} \rightarrow \mathcal{M}$ ; and if  $\mathcal{X}$  is randomized, it can be seen as a function randomly chosen from a family of such functions  $\mathcal{F} = \{f_1, f_2, \dots\}$ , where  $f_i$  is chosen with probability  $p(f_i)$ . Without loss of generality, we assume that  $p(f_1) \geq p(f_2) \geq \dots$ . 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  $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$ :

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

Let us focus on the first  $k = \lceil \delta \cdot |\mathcal{F}| \rceil + 1$  functions  $f_1, \dots, f_k$ . 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

$$|\mathcal{U}| \leq |\mathcal{M}|^k \quad (2)$$

so that no two  $\mathbf{w} \neq \mathbf{v} \in \mathcal{U}$  have  $f_i(\mathbf{w}) = f_i(\mathbf{v})$  for all  $i = 1, \dots, k$ ; 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.  $\square$

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  $\log m = O(\log \frac{m}{\delta})$ . 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 practically means no error at all. 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 nowhere near  $n$ . In this case, since  $m$  is typically much smaller, PIRS-1 would be the better choice.

**Information Disclosure on Multiple Attacks.** Theorem

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 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 a compromised, smart server can extract knowledge from this warning (e.g., it knows at least that the same attack will always fail), the guarantee of Theorem 1 is not applicable any more. 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.,  $p_k = \Pr[e_k = 1 \mid e_1 = 0, \dots, e_{k-1} = 0]$ . From Theorem 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*; 2. Maximally explores the knowledge that could be gained from one failed attack; and 3. Possesses infinite 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  $\mathbf{w}_1, \dots, \mathbf{w}_{k-1}$ , 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:

$$\mathbf{Lemma 1} \quad p_k \leq \frac{\delta}{1 - (k-1)\delta}.$$

*Proof:*

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

$\square$

**Theorem 5** *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

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

□

Theorem 5 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. Of course, such disclosure is not required in most applications.

## 5 Tolerance for Few Errors

This section presents a synopsis for solving the  $\text{CQV}^\gamma$  problem (Definition 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  $\text{CQV}^\gamma$ , and requires  $O(\gamma^2 \log \frac{1}{\delta} \log n)$  bits of space, which is practicable only for small  $\gamma$ 's. Then, we provide an approximate solution which uses only  $O(\gamma \log \frac{1}{\delta} (\log m + \log n))$  bits. Both 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  $\text{CQV}^\gamma$ . 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 less 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.

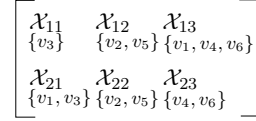


Figure 2. The PIRS $^\gamma$  synopsis.

---

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

---

```

1  $c = 4.819, k = \lceil c\gamma^2 \rceil$ 
2 Generate  $\beta_{\ell,j}$  uniformly at random from  $\mathbb{Z}_p$ , for
    $1 \leq \ell \leq \log 1/\delta, 1 \leq j \leq k$ 
3 for  $\ell = 1, \dots, \lceil \log 1/\delta \rceil$  do
4   Layer  $L_\ell = [\mathcal{X}_1(\mathbf{v}) := 0, \dots, \mathcal{X}_k(\mathbf{v}) := 0]$ 
   //  $\mathcal{X}_j(\mathbf{v})$  is a PIRS synopsis with
    $\delta' = 1/c\gamma$ 

```

---



---

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

---

```

1 for  $\ell = 1, \dots, \lceil \log 1/\delta \rceil$  do
2    $b_{\ell,i} = (\beta_{\ell,\gamma}i^{\gamma-1} + \dots + \beta_{\ell,2}i + \beta_{\ell,1}) \bmod k + 1$ 
3   Update  $L_\ell.\mathcal{X}_{b_{\ell,i}}(\mathbf{v})$  using  $s$ 

```

---



---

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

---

```

1 for  $\ell = 1, \dots, \lceil \log 1/\delta \rceil$  do
2   Layer  $M_\ell = [\mathcal{X}_1(\mathbf{w}) := 0, \dots, \mathcal{X}_k(\mathbf{w}) := 0]$ 
   //  $\mathcal{X}_j(\mathbf{w})$  is a PIRS synopsis with
    $\delta' = 1/c\gamma$ 
3   for  $i = 1, \dots, n$  do
4     Generate  $b_{\ell,i}$  as line 2, Algorithm 2
5     Update  $M_\ell.\mathcal{X}_{b_{\ell,i}}(\mathbf{w})$  by  $s = (i, w_i)$ 
6   if  $|\{j \mid L_\ell.\mathcal{X}_j(\mathbf{v}) \neq M_\ell.\mathcal{X}_j(\mathbf{w}), 1 \leq j \leq k\}| \geq \gamma$ 
     then Raise an alarm

```

---

Concentrating on one layer only, let  $b_1, \dots, b_n$  be  $n$   $\gamma$ -wise independent random numbers, uniformly distributed over  $\{1, \dots, k\}$ . 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 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 less 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 less 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  $\gamma$ -wise independent fashion, by considering just  $\gamma$  of them, the probability that less than  $\gamma$  buckets get at least one of the erroneous  $v_i$ 's is at most

$$\begin{aligned} & 1 - \frac{k}{k} \cdot \frac{k-1}{k} \cdots \frac{k-\gamma+1}{k} \\ & \leq 1 - \left(\frac{k-\gamma}{k}\right)^{\gamma} = 1 - \left(1 - \frac{\gamma}{k}\right)^{\frac{k}{\gamma} \cdot \frac{\gamma^2}{k}} \leq 1 - 2^{-\frac{2}{c_1}} \leq \frac{1}{4}, \end{aligned} \quad (3)$$

where the last inequality holds as long as  $c_1 \geq 2/\log \frac{4}{3} = 4.819$ .

Next, consider the second case. The probability that some of the  $\gamma$  buckets that are supposed to raise an alarm fail is:

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

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 6** *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  $\gamma$ -wise independent random numbers. Using standard techniques we can generate them on-the-fly using  $O(\gamma \log n)$  truly random bits. Specifically, the technique of [41] for constructing  $k$ -universal hash families can be used. Let  $p$  be some prime between  $n$  and  $2n$ , and  $\alpha_0, \dots, \alpha_{\gamma-1}$  be  $\gamma$  random numbers chosen uniformly and independently from  $\mathbb{Z}_p$ . Then we set

$$b_i = ((\alpha_{\gamma-1}i^{\gamma-1} + \alpha_{\gamma-2}i^{\gamma-2} + \dots + \alpha_0) \bmod p) \bmod k+1,$$

for  $i = 1, \dots, n$ . For an incoming tuple  $s = (i, u)$ , we compute  $b_i$  using the  $\alpha_j$ 's in  $O(\gamma)$  time, and then perform the update to the corresponding PIRS. To perform a verification, we can compute in parallel for all the layers 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 7** *PIRS $^{\gamma}$  requires  $O(\gamma^2 \log \frac{1}{\delta} (\log m + \log n))$  bits, spends  $O(\gamma \log \frac{1}{\delta})$  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-Oh 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.

**Remark.** Note that computing the  $\gamma$ -wise independent random numbers  $b_i$  is the bottleneck for the time bounds. We can trade space for faster update times by using other  $\gamma$ -wise independent random number generation schemes. For instance by using an extra  $O(n^{\epsilon})$  words per layer, the technique of [36] can generate a  $b_i$  in  $O(1)$  time provided that  $\gamma \leq n^{\epsilon/2}$ , for  $\epsilon > 0$ . The update and verification times become  $O(\log \frac{1}{\delta})$  and  $O(n \log \frac{1}{\delta})$ , and the space bound  $O(n^{\epsilon} \log \frac{1}{\delta} \log n)$  bits.

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

The exact solution works 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 8** *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 desiderata is on any  $\mathbf{w} =_{\gamma^-} \mathbf{v}$  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 [30].

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 sum 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 less 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  $w_1, w_2, \dots$ , 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  $w_1, \dots, w_i$  have collected all the buckets. Then the problem becomes an instantiation of the *coupon collector’s problem* [30] (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 2 ([30], Theorem 3.8)** *For any constant  $c'$ ,*

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

where  $o(1)$  is in terms of  $k$ .

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

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

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

Now, consider the following two cases. If  $\mathbf{w} =_{\gamma^-} \mathbf{v}$ , then the probability that these less 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, there exists some constant  $\epsilon$  such that the probability of raising a false alarm is (for large  $\gamma$ )

$$e^{-e^c} \leq 1/2 - \epsilon,$$

for any  $c > \ln \ln 2$ . 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  $c > \max\{\ln \ln 2, -\ln \ln 2\} = -\ln \ln 2$ , 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 simple majority. 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

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

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

Finally, we use the technique discussed in Section 5.1 to generate  $\gamma^+$ -wise independent random numbers, by storing  $O(\gamma^+) = O(\gamma)$  truly random numbers per layer. We have thus obtained the desired results:

**Theorem 9** *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.*

As mentioned before, the technique of [36] can be used to obtain space-time trade-offs with respect to generating the  $\gamma^+$ -wise independent random numbers.

### 5.3 Information Disclosure on Multiple Attacks

Similarly to the analysis in Section 4, since  $\text{PIRS}^\gamma$  has false negatives only, it is a randomized algorithm with one-sided errors like PIRS. It is easy to prove that Theorem 5 holds for  $\text{PIRS}^\gamma$  as well.

$\text{PIRS}^{\pm\gamma}$  is a randomized algorithm with two-sided errors, which is in favor of attackers and may be exploited by a smart server, since a portion of the seeds can be excluded both in the case that Alice is sending an incorrect answer and when she is sending a correct answer but  $\text{PIRS}^{\pm\gamma}$  reports a false alarm. Using arguments similar to Theorem 5 the following can be stated:

**Theorem 10** *Assuming that Alice has returned a total of  $k$  results  $\mathbf{w}_1, \dots, \mathbf{w}_k$  to  $\text{PIRS}^{\pm\gamma}$ , including both honest answers and attacks, the probability that  $\text{PIRS}^{\pm\gamma}$  correctly identifies all consistent and inconsistent  $\mathbf{w}_i$ 's is at least  $1 - k\delta$ .*

Theorem 10 is slightly weaker than Theorem 5, since a new possible attack strategy for Alice is to simply return correct results and wait until a sufficient portion of the seeds have been ruled out before launching an attack. However, since the success probability drops linearly to the number of rounds, one can always intentionally set  $\delta$  extremely small. For instance, if we choose  $\delta = 2^{-30}$ , it will take Alice a million rounds in order to have a 0.1% chance of launching a successful attack. As always, information disclosure by the clients is not necessary for most applications.

## 6 Tolerance for Small Errors

In this section we prove the hardness of solving  $\text{CQV}^\eta$  (Definition 3) using sub-linear 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 11** *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 infinity frequency moment, defined as follows. Let  $A = (a_1, a_2, \dots)$  be a sequence of elements from the set  $\{1, \dots, n\}$ . The *infinity frequency moment*, denoted by  $F_\infty$ , is the number of occurrences of the most frequent element. Alon et al. [3] showed that any randomized algorithm that

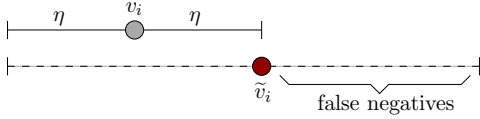
makes one pass over  $A$  and computes  $F_\infty$  with a relative error of at most  $1/3$  and a success probability greater than  $1 - \delta$  for any  $\delta < 1/2$ , has to use  $\Omega(n)$  memory bits. In particular, they proved that even if each element appears at most twice, it requires  $\Omega(n)$  bits in order to decide if  $F_\infty$  is 1 or 2 with probability at least  $1 - \delta$ .

Let  $\mathcal{X}$  be a synopsis solving the problem stated in Theorem 11. We will show how to use  $\mathcal{X}$  to compute the infinity 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  $1 - \delta$ .  $\square$

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  $\mathbf{w} = (n/(1+\eta), \dots, n/(1+\eta))$  in the proof above.

Given the hardness of solving  $\text{CQV}^\eta$ , we are interested in seeking alternative methods that might be able to give us approximate answers using space less than the exact solution. Here we briefly discuss one such method.

**The CM sketch.** The CM sketch [18] uses  $O(\frac{1}{\epsilon} \log \frac{1}{\delta'})$  words of space and provides an approximate answer  $\tilde{v}_i$  for any  $i \in [n]$ , that satisfies  $v_i - 3\epsilon\|\mathbf{v}\|_1 \leq \tilde{v}_i \leq v_i + 3\epsilon\|\mathbf{v}\|_1$  with probability  $1 - \delta'$ , for any  $\epsilon \in (0, 1)$ . However, this does not make it applicable for solving  $\text{CQV}^\eta$  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  $\tilde{v}_i$  such that  $|\tilde{v}_i - v_i| \leq \eta$  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  $|w_i - \tilde{v}_i| \geq 2\eta$ . 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 (see Figure 3). Especially, some false negatives may be observed for some range of  $\mathbf{w}$ . 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 ( $W$  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.



**Figure 3. False negatives for the CM sketch approach.**

## 7 Extensions

In this section we discuss several extensions of PIRS. 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 $^{\pm\gamma}$ .

**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 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). Let  $Q_1, \dots, Q_k$  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  $k$ . Interestingly, 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 1** *PIRS-1 for  $k$  queries occupies  $O(\log \frac{m}{\delta} + \log n)$  bits of space, spends  $O(k)$  time to process an update, and  $O(|\mathbf{w}| \log \frac{m}{|\mathbf{w}|})$  time to perform a verification.*

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

**Handling sliding windows.** Another nice 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 [20]. Again 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  $\mathcal{X}(\mathbf{v}_1), \dots, \mathcal{X}(\mathbf{v}_k)$  be the PIRS for these

periods. When the server returns a result  $\mathbf{w}$ , the client computes the overall  $\mathcal{X}(\mathbf{v}) = \prod_{i=1}^k \mathcal{X}(\mathbf{v}_i)$ , and then verifies the result.

**Corollary 2** *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.*

**Synchronization.** In our discussions we omitted superscript  $\tau$  for simplicity. Hence, an implicit assumption was made that the result  $\mathbf{w}^{\tau_s}$  returned by the server was synchronized with  $\mathcal{X}(\mathbf{v}^{\tau_c})$  maintained by the client, i.e.,  $\tau_s = \tau_c$ . Correct verification can be performed only if the server and the client are synchronized. Obviously, such perfect synchronization is hard to obtain in practice, especially in a DSMS. Also, if  $n$  is large, transmitting the result itself takes non-negligible time. The solution to this problem is as follows. Suppose that the client sends out a request to the server asking for the query result at time  $\tau$ , which is either a time instance at present or in the future. When the client has received  $s^\tau$  from the stream  $\mathcal{S}$  and has computed the synopsis for  $\mathbf{v}^\tau$ , it makes a copy of  $\mathcal{X}(\mathbf{v}^\tau)$ , and continues updating PIRS. When the server returns the answer  $\mathbf{w}^\tau$ , the client can do the verification using the snapshot. The synchronization problem once again illustrates the importance of using small space, as keeping a copy (or potentially many copies if there are significant delays in the server's response) could potentially become very expensive. Similar ideas can be used on the server side for dealing with queries referring to the past.

**Exploiting locality.** In many practical situations data streams tend to exhibit a large degree of locality. Simply put, updates to  $\mathbf{v}$  tend to cluster to the same components. In this setting, it is possible to explore space/time trade-offs. We can 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 selected 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 update the synopsis in bulk, we incur a smaller, amortized update processing cost per tuple.

## 8 Empirical Evaluation

In this section we evaluate the performance of the proposed synopses over two real data streams [6, 1]. The experimental study demonstrates that our synopses: 1. use very small

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

**Table 1. Average update time per tuple.**

space; 2. support fast updates; 3. have very high accuracy; 4. support multiple queries; and 5. are easy to implement.

## 8.1 Setup

Our synopses are implemented using GNU C++ and the GNU GMP extension which provides arbitrary precision arithmetic, useful for operating on numbers longer than 32 bits. The experiments were run on an Intel Pentium 2.8GHz CPU with 512KB L2 cache and 512MB of main memory. Our results show that using our techniques, even a low-end client machine can efficiently verify online queries with millions of groups on real data streams.

The *World Cup (WC)* data stream [6] 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. The *IP traces (IPs)* data stream [1] 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/port, destination IP/port, and packet size header fields. The data set consists of a segment of one day traffic and has 100 million packets. Without loss of generality, unless otherwise stated, we perform the following default query: 1. Count or Sum (on response size) query group-by client id/object id for the WC data set; 2. Count or Sum (on packet size) query group-by source IP/destination IP for the IPs data set. 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.

## 8.2 PIRS

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 clearly the better choice, and is thus used in our experiments. We precomputed  $p$  as the smallest prime above  $2^{64}$  and used the same  $p$  throughout this section. Thus, each word (storing  $p$ ,  $\alpha$ , and  $\mathcal{X}(\mathbf{v})$ ) occupies 9 bytes.

**Space usage.** 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 of memory, respectively.

**Update cost.** PIRS has excellent update cost which is crucial to the streaming algorithm. The average per-tuple update cost is shown in Table 1 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.

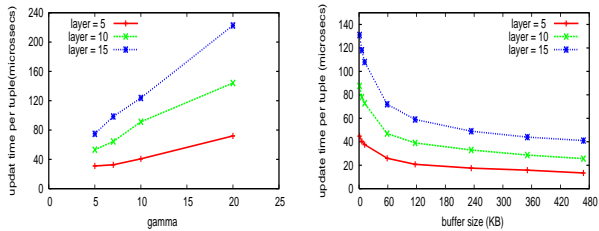
**Detection accuracy.** As guaranteed by the theoretical analysis, the probability of failure of PIRS-1 is  $\delta \leq m/p$ , which is at most  $0.5 \times 10^{-9}$ . Note that our estimate of  $m$  is very conservative; the actual  $\delta$  is much smaller. We generated 100,000 random attacks and,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

## 8.3 PIRS $^\gamma$ and PIRS $^{\pm\gamma}$

For the rest of the experiments, we focus on the Count query on the WC data set. Similar patterns have been observed on the IPs data set.

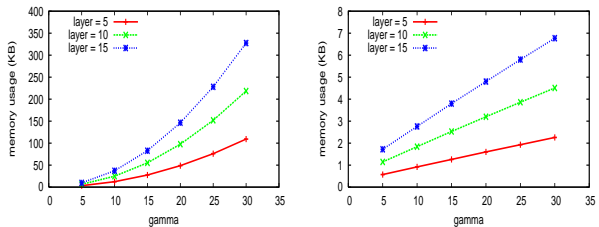
**Update cost.** In this set of experiments we study the performance of PIRS $^\gamma$  and PIRS $^{\pm\gamma}$ . Clearly, PIRS $^\gamma$  has linear update cost w.r.t the number of layers and  $\gamma$  (the number of inconsistent groups to detect), as confirmed in Figure 4(a). It is not hard to see that PIRS $^\gamma$  and PIRS $^{\pm\gamma}$  have almost the same update cost if they are configured with the same number of layers. Essentially, each one has to generate the  $\gamma$ -wise (or  $\gamma^+$ -wise) independent random numbers on-the-fly and update one PIRS synopsis at each layer. Hence, we only show the cost for PIRS $^\gamma$ . However, the space cost of the two synopses is different. PIRS $^\gamma$ , as an exact solution for CQV $^\gamma$ , is expected to use much larger space than its counterpart PIRS $^{\pm\gamma}$ , which gives approximate solutions. This is demonstrated in Figure 5. By construction, at each layer PIRS $^\gamma$  has  $O(\gamma^2)$  and PIRS $^{\pm\gamma}$   $O(\frac{\gamma}{\ln \gamma})$  buckets, which is easily observed in Figure 5(a) and 5(b) respectively.

**Space/Time Trade-offs.** If the client can afford to allocate some extra space, but still cannot store the entire vector  $\mathbf{v}$ , as discussed in Section 7, it is possible to exploit the locality in the input data streams to reduce the amortized update cost. A simple LRU buffer has been added to PIRS $^\gamma$  and PIRS $^{\pm\gamma}$  and its effect on update cost is reported in Figure 4(b) with  $\gamma = 10$ . Again, both synopses exhibit very sim-



(a) Update cost per tuple. (b) Space time tradeoff.

Figure 4.  $\text{PIRS}^\gamma, \text{PIRS}^{\pm\gamma}$  : running time.



(a)  $\text{PIRS}^\gamma$ . (b)  $\text{PIRS}^{\pm\gamma}$ .

Figure 5.  $\text{PIRS}^\gamma, \text{PIRS}^{\pm\gamma}$  : memory usage.

ilar behavior. As the figure indicates, a very small buffer (up to 500 KB) that fits into the cache is able to reduce the update cost by an order of magnitude. The improvement on the update cost of this buffering technique depends on the degree of locality in the data stream. Note that if this simple buffering technique is still insufficient for achieving the desired update speed (e.g., when there is very little locality in the data stream or  $\gamma$  is large) we could use the technique of [36] to reduce the cost to  $O(\log \frac{1}{\delta})$  (independent of  $\gamma$ ) by using extra space, but generating random numbers much faster.

**Detection accuracy.** We observed that both of our synopses can achieve excellent detection accuracy as the theoretical analysis suggests. All results reported here are the ratios obtained from 100,000 rounds. Since the detection mechanism of the synopses does not depend on the data characteristics, both data sets give similar results. Figure 6(a) shows the ratios of raising alarms versus the number of actual inconsistent groups, with  $\gamma = 10$  and 10 layers. As expected,  $\text{PIRS}^\gamma$  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,  $\text{PIRS}^{\pm\gamma}$  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 off  $\gamma$ , both false positives and negatives reduce to zero. We have also studied the impact of the number of layers on the

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

Table 2. Update time and memory usage of PIRS for multiple queries.

detection accuracy. Our theoretical analysis gives provable bounds. For example with  $\text{PIRS}^\gamma$  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 6(b) reports the result for  $\text{PIRS}^\gamma$ . With less layers (4–6) it still achieves excellent detection accuracy. Only when the number of actual inconsistent groups is close to  $\gamma$ , a small drop in the detection ratio is observed. Figure 6(c) reports the same experiment for  $\text{PIRS}^{\pm\gamma}$  with layers from 10 to 20. Smaller number of layers enlarges the transition region and larger number of layers sharpens it. Outside this region, 100% detection ratio is always guaranteed. Finally, experiments have been performed over different values of  $\gamma$ 's and similar behavior has been observed.

## 8.4 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 7, 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 2 we report the update time and memory usage for PIRS; similar results were observed for  $\text{PIRS}^\gamma$  and  $\text{PIRS}^{\pm\gamma}$ .

## 9 Related Work

PIRS is a way of summarizing the underlying data streams. In that respect our work is related with the line of work on sketching techniques [3, 29, 9, 18, 21, 23]. As discussed in Section 3, since these sketches are mainly designed for estimating certain statistics of the stream (e.g. the frequency moments), they cannot solve the verification problem.

Another approach for solving the CQV problem is to use program execution verification [13, 40] on the DSMS of the server. We briefly discuss two representatives from this field [17, 42]. The main idea behind these approaches is for the client to precompute hash values in nodes of the *control*

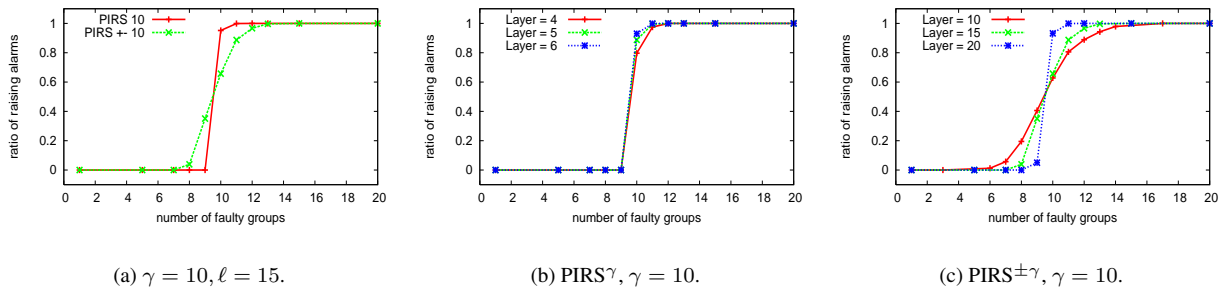


Figure 6. Detection with tolerance for limited number of errors.

*flow graph* of the query evaluation algorithm on the server’s DSMS. Subsequently, the client can randomly select a subset of nodes from the expected execution path and ask the server for the corresponding hashes. If the server does not follow the correct execution it will be unable to provide the correct hashes. With guaranteed probability (with respect to the sampling ratio) the client will be able to verify whether the query has been honestly executed by the server or not. Clearly, these techniques do not solve the CQV problem since, first, the server can honestly execute the algorithm throughout and only modify the results before transmitting them to the client, and second, there is no way for the client to compute the correct hashes of the program execution, unless if it is recomputing the same algorithm in real-time, or storing the whole stream for later processing.

Various authentication techniques are also relevant to the present work. 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. Here the authenticated signature refers to any possible authentication techniques, such as RSA or Message Authentication Code (MAC). However, a fundamental challenge in applying such techniques to our problem is how to perform incremental updates using  $\sigma(\mathbf{v})$  alone, without storing  $\mathbf{v}$ , i.e., in the present setting the message  $\mathbf{v}$  is constantly updated. Cryptography researchers have devoted considerable effort in designing *incremental cryptography* [11]. Among them incremental signature and incremental MAC [11, 12] are especially interesting for this work. 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.

There is also considerable work on authenticating query

execution in an outsourced database setting [37, 33, 27, 28]. 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. In [33, 27] cryptographic primitives such as digital signatures and the Merkle hash tree are applied to design efficient index structures build by the data publisher to enable authentication of query results. In [37], it is assumed that the client possess a copy of the data and a random sampling based approach (by posing test queries with known results) is used. All these techniques apply only to offline settings and not for online, one-pass streaming scenarios. In [28] authentication of sliding window queries over data streams has been studied, however, the outsourced model is different from what we have presented here. The client does not possess the data stream and the data publisher is responsible for injecting “proofs” (in the forms of some cryptographic primitives) into its data stream so that the server could construct a verification object associated with a query requested by the client. Hence, it is fundamentally different from the problem we have studied in this paper (where the client and the data publisher are the same entity).

Verifying the identity of polynomials is a fingerprinting technique [30]. 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 [22], multivariate polynomial identities [35], and verifying equality of strings [30]. We refer readers for an excellent discussion on these problems to [30]. 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.

We would like to point out that other security issues for secure computation/querying over streaming and sensor data start to receive attention recently. For example, orthogonal to our problem, [15, 24] 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. Furthermore, the verifier does not have access to the original data stream. 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.

## 10 Conclusion

The present work introduced an important new problem, that of verifying query results in an outsourced data stream setting. We proposed various space/time efficient probabilistic algorithms for selection queries and aggregation queries that can be reduced to count and sum. First, we proposed algorithms for detecting any error in the results with very high confidence. Then, we extended this algorithm to identify query results with more than a pre-determined number of errors. Finally, we proved that identifying query results with large absolute or relative errors is hard. In the future, we plan to investigate algorithms for join queries and other types of aggregate functions, for example max/min. Finally, we would like to extend these techniques for identifying subsets of the result vectors than contain only correct answers.

**Acknowledgments.** We would like to thank Graham Cormode for many insightful discussions concerning this problem.

## References

- [1] AT&T network traffic streams. AT&T Labs.
- [2] D. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryzkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.
- [3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.
- [4] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, 2003.
- [5] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, pages 286–296, 2004.
- [6] M. Arlitt and T. Jin. <http://www.acm.org/sigcomm/ITA/>. ITA, 1998 World Cup Web Site Access Logs.
- [7] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.
- [8] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE*, pages 350–361, 2004.
- [9] B. Babcock, M. Datar, R. Motwani, and L. O’Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS*, pages 234–243, 2003.
- [10] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM*, pages 1–10, 2002.
- [11] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO*, pages 216–233, 1994.
- [12] M. Bellare, R. Guerin, and P. Rogaway. Xor macs: New methods for message authentication using finite pseudorandom functions. In *CRYPTO*, pages 15–28, 1995.
- [13] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.
- [14] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams—a new class of data management applications. In *VLDB*, pages 215–226, 2003.
- [15] H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation in sensor networks. In *CCS*, pages 278–287, 2006.
- [16] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, D. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: continuous dataflow processing for an uncertain world. In *CIDR*, 2003.
- [17] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. H. Jakubowski. Oblivious hashing: A stealthy software integrity verification primitive. In *Proc. of the International Workshop on Information Hiding (IH)*, pages 400–414, 2002.
- [18] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [19] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for internet databases. In *SIGMOD*, pages 647–651, 2003.
- [20] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, pages 635–644, 2002.
- [21] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.
- [22] R. Freivalds. Fast probabilistic algorithms. In *MFCS*, pages 57–69, 1979.
- [23] S. Ganguly, M. Garofalakis, and R. Rastogi. Tracking set-expression cardinalities over continuous update streams. *The VLDB Journal*, 13(4):354–369, 2004.
- [24] M. Garofalakis, J. M. Hellerstein, and P. Maniatis. Proof sketches: Verifiable in-network aggregation. In *ICDE*, 2007.
- [25] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. M. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong. Nile: A query processing engine for data streams. In *ICDE*, page 851, 2004.

- [26] D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1997.
- [27] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD*, pages 121–132, 2006.
- [28] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios. Proof-infused streams: Enabling authentication of sliding window queries on streams. In *VLDB*, 2007.
- [29] G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *VLDB*, pages 346–357, 2002.
- [30] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [31] S. Muthukrishnan. *Data streams: algorithms and applications*, 2003.
- [32] T. Nagell. *Introduction to Number Theory*. Chelsea Publishing Company, second edition, 1981.
- [33] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, pages 407–418, 2005.
- [34] F. Rusu and A. Dobra. Pseudo-random number generation for sketch-based estimations. *TODS*, 32(2), 2007.
- [35] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.
- [36] A. Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications. In *FOCS*, pages 20–25, 1989.
- [37] R. Sion. Query execution assurance for outsourced databases. In *VLDB*, pages 601–612, 2005.
- [38] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.
- [39] N. Tatbul and S. Zdonik. Window-aware load shedding for aggregation queries over data streams. In *VLDB*, pages 799–810, 2006.
- [40] H. Wasserman and M. Blum. Software reliability via runtime result-checking. *Journal of the ACM*, 44(6):826–849, 1997.
- [41] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3), 1981.
- [42] S. Yang, A. R. Butt, Y. C. Hu, and S. P. Midkiff. Trust but verify: monitoring remotely executing programs for progress and correctness. In *PPOPP*, pages 196–205, 2005.
- [43] R. Zhang, N. Koudas, B. C. Ooi, and D. Srivastava. Multiple aggregations over data streams. In *SIGMOD*, pages 299–310, 2005.