Theory Comput. Systems OF1–OF22 (2004)
DOI: 10.1007/s00224-004-1156-4

Theory of Computing Systems

Distributed Streams Algorithms for Sliding Windows*

Phillip B. Gibbons[1] and Srikanta Tirthapura[2]

[1]Intel Research Pittsburgh, 417 South Craig Street,
Pittsburgh, PA 15213, USA
phillip.b.gibbons@intel.com

[2] Department of Electrical and Computer Engineering, Iowa State University,
Ames, IA 50010, USA
snt@iastate.edu

Abstract. Massive data sets often arise as physically distributed, parallel data streams, and it is important to estimate various aggregates and statistics on the union of these streams. This paper presents algorithms for estimating aggregate functions over a "sliding window" of the N most recent data items in one or more streams. Our results include:

1. For a *single stream*, we present the first ε-approximation scheme for the number of 1's in a sliding window that is optimal in both worst case time and space. We also present the first ε-approximation scheme for the sum of integers in $[0..R]$ in a sliding window that is optimal in both worst case time and space (assuming R is at most polynomial in N). Both algorithms are deterministic and use only logarithmic memory words.
2. In contrast, we show that *any* deterministic algorithm that estimates, to within a small constant relative error, the number of 1's (or the sum of integers) in a sliding window on the *union of distributed streams* requires $\Omega(N)$ space.
3. We present the first (randomized) (ε, δ)-approximation scheme for the number of 1's in a sliding window on the *union of distributed streams* that uses only logarithmic memory words. We also present the first (ε, δ)-approximation scheme for the number of distinct values in a sliding window on distributed streams that uses only logarithmic memory words.

Our results are obtained using a novel family of synopsis data structures called *waves*.

* A preliminary version of this paper appeared in the *Proceedings of the* 14*th ACM Symposium on Parallel Algorithms and Architectures* [19].

stream	0	1	$\cdots$	0	1	0	0	0	0	1	1	0	1	1
position	1	2	$\cdots$	61	62	63	64	65	66	67	68	69	70	71
1-*rank*		1	$\cdots$		31					32	33		34	35

stream	1	1	1	1	1	1	0	1	1	0	0	0	1	1
position	72	73	74	75	76	77	78	79	80	81	82	83	84	85
1-*rank*	36	37	38	39	40	41		42	43				44	45

stream	1	0	0	1	0	1	0	0	1	0	0	0	0	1
position	86	87	88	89	90	91	92	93	94	95	96	97	98	99
1-*rank*	46			47		48			49					50

Fig. 1. An example data stream, through $m = 99$ bits. The position in the stream (*position*) and the rank among the 1-bits (1-*rank*) are computed as the stream is processed.

1. Introduction

There has been a flurry of recent work on designing effective algorithms for estimating aggregates and statistics over data streams [1]–[11], [13], [15], [17], [20]–[24], [28], [31], motivated by their importance in network monitoring, data warehousing, telecommunications, etc. This work has focused almost entirely on the *sequential* context of a data stream observed by a single party. Figure 1 depicts an example data stream, where each data item is a bit, either 0 or 1. Shown are the first 99 data items in the stream, each item's position in the stream, and for the 1-bits, its rank among the 1-bits (1-*rank*). The main challenge in data stream algorithms is to carry out the computation in a single pass over the data using only limited workspace memory.

Although previous work has focused on the sequential data stream context, for many of the above applications there are multiple *concurrent* data sources, each generating its own data stream. In network monitoring and telecommunications, for example, each node/person in the network is a potential source for new streaming data. In a large retail data warehouse each retail store produces its own stream of items sold. To model these scenarios we previously proposed a *distributed streams* model [18], in which there are a number of data streams, each stream is observed by a single party, and the aggregate is computed over the union of these streams.

Moreover, in many real-world scenarios only the most recent data is important. For example, in telecommunications, call records are generated continuously by customers, but most processing is done only on recent call records, after which the records are archived and not used again [9]. To model such scenarios, recent work [9], [14], [20] has studied a *sliding windows* setting for data streams, in which aggregates and statistics are computed over a "sliding window" of the N most recent items in the stream.

This paper studies the sliding windows setting in both the single stream and distributed stream models, improving upon previous results in both models. We next describe the models, the previous results, and our new results in greater detail.

1.1. *Data Stream Computations*

An algorithm for a data stream has to perform its computations in only one pass over the input, and has limited workspace. The goal in a (sequential or distributed) streams algorithm is to estimate a function on the input while minimizing (1) the total workspace (memory) used by all the parties, (2) the time taken by a party to process each data item, and (3) the time to produce an estimate (i.e., the query time). Many functions on (sequential and distributed) data streams require linear space to compute exactly, and hence attention is focused on finding either an (ε, δ)-approximation scheme or an ε-approximation scheme, defined next.

Definition 1. An (ε, δ)-*approximation scheme* for a quantity X is a randomized procedure that, given any positive $\varepsilon < 1$ and $\delta < 1$, computes an estimate $\hat{X}$ that is within a relative error of ε of X with probability at least $1 - \delta$, *i.e.*, $\mathbf{Pr}\{|\hat{X} - X| \le \varepsilon X\} \ge 1 - \delta$. An ε-*approximation scheme* is a deterministic procedure that, given any positive $\varepsilon < 1$, computes an estimate for X whose worst case relative error is at most ε.

1.2. *Algorithms for a Sliding Window on a Single Stream*

In the *sliding windows* model the computations have to be performed only on the N most recent data items in the stream, and the goal is to use $\ll N$ workspace. With limited workspace, one cannot keep track of all the items in the window. Consider the problem of estimating the number of 1's in a sliding window (called *Basic Counting* in [9]). In the stream in Figure 1, for example, the number of 1's in the current window of the $N = 39$ most recent items (i.e., items 61–99) is 20. Datar et al. [9] present an ε-approximation scheme for Basic Counting that uses $O((1/\varepsilon)\log^2(\varepsilon N))$ bits of workspace memory, processes each data item in $O(1)$ amortized and $O(\log N)$ worst case time, and can produce an estimate over the current window in $O(1)$ time. They also prove a matching lower bound for the space. They demonstrate the importance of this problem by using their algorithm as a building block for a number of other functions, such as the sum of bounded integers and the L_p norms (in a restricted model).

We improve on the results in [9] by presenting an ε-approximation scheme for Basic Counting that matches the space and query time bounds, while improving the per-item processing time to $O(1)$ worst case. Moreover, we also present an ε-approximation scheme for the sum of bounded integers in a sliding window that again matches the space and query times, while improving the per-item processing time from $O(1)$ amortized and $O(\log N)$ worst case time to $O(1)$ worst case.

Our algorithms use a family of small space data structures, which we call *waves*.[1] An example wave for Basic Counting is given in Figure 2, for the data stream in Figure 1. The x-axis is the 1-rank, and extends to the right as new 1-bits arrive. The wave contains the positions of the recent 1-bits in the stream, arranged at different levels. Each level of the wave stores a fixed number of positions in the stream, with level i storing the most recent 1-bits whose 1-ranks are a multiple of 2^i. (Waves are described in detail in Section 3.) As we shall see, as additional stream bits arrive, the wave retains this basic shape while "moving" to the right so that the crest of the wave is always over the largest 1-rank thus far.

1.3. *Algorithms for a Sliding Window on Distributed Streams*

Previous work on distributed streams did not consider sliding windows. There are several ways one can define a sliding window on distributed streams: we propose three ways, and show that while two of the definitions can be solved by straightforward applications

[1] The name arises because its basic shape is suggestive of an ocean wave about to break.

of single stream algorithms, the third (based on positionwise union) is more challenging. We present an $\Omega(N)$ lower bound on the space required by any deterministic algorithm that guarantees a small constant relative error for estimating the number of 1's (or the sum of bounded integers) on the positionwise union of distributed streams. The lower bound holds even when there are only two parties and sliding windows are not considered (in which case N is the current length of the stream). This motivates the study of randomized approximation schemes.

We present an (ε, δ)-approximation scheme for the number of 1's in a sliding window on the positionwise union of distributed streams. We use this as a building block in constructing an (ε, δ)-approximation scheme for the number of distinct values in sliding windows on both single and distributed streams. Each scheme uses only logarithmic memory words per party. The key idea is to use *randomized* waves. In contrast with a deterministic wave (discussed above), where a 1-bit's position is selected into level i if its 1-rank is a multiple of 2^i, in a randomized wave each 1-bit's position is selected into level i *with probability* 2^{-i}. Each party uses the same pseudorandom hash function (applied to the position number) as its source of randomness, in order to ensure a "positionwise coordination" among the choices made for each stream.

1.4. *Summary of Contributions*

The contributions of this paper are as follows:

1. We introduce a family of synopsis data structures called *waves*, and demonstrate their utility for data stream processing in the sliding windows setting.
2. For a sliding window of size N on a single stream, we present the first ε-approximation schemes for the number of 1's and for the sum of integers in $[0..R]$ that are optimal in worst case space,[2] processing time, and query time.
3. For a sliding window of size N on the union of distributed streams, we present an $\Omega(N)$ lower bound on the space used by any deterministic algorithm that estimates the number of 1's within a small constant relative error. In contrast, we present the first randomized (ε, δ)-approximation schemes for the number of 1's and for the number of distinct values that use only logarithmic memory words.

The remainder of the paper is organized as follows. Section 2 presents background and further comparisons with previous related work. Sections 3 and 4 present results using the deterministic (randomized, resp.) wave synopsis. Finally, Section 5 shows how the techniques can be used for various other problems such as distinct values counting and nth most recent 1.

2. Background and Related Work

There have been many papers on data streams algorithms (e.g., [1]–[11], [13], [15], [17], [20]–[24], [28], and [31]). In this section we restrict our attention to work related to distributed streams, sliding windows, and distinct values counting.

The distributed streams model used in this paper was introduced in [18]. In the model

[2] For the integer sum problem, the space matches the lower bound when R is at most polynomial in N.

each party observes only its own stream, has limited workspace, and communicates with the other parties only when an estimate is requested. The estimate, however is computed on the *union* of all the streams seen by all the parties. Specifically, when asked for an estimate, each party sends a message to a Referee who computes the estimate. This model reflects the set-up used by commercial network monitoring products, where the data analysis front-end serves the role of the Referee. Among the results in [18] were (i) an (ε, δ)-approximation scheme for the number of 1's in the union of distributed streams (i.e., in the bitwise OR of the streams), using $O((1/\varepsilon^2)\log(1/\delta)\log n)$ memory bits per party, where n is the length of the stream, and (ii) an (ε, δ)-approximation scheme for the number of distinct values in a collection of distributed streams, using $O((1/\varepsilon^2)\log(1/\delta)\log R)$ memory bits, where the values are in $[0..R]$. Both algorithms use a technique the authors call *coordinated sampling*: each stream is sampled at the same random positions, for a given sampling rate. Each party stores the positions of (only) the 1-bits in its sample. When the stored 1-bits exceed the target space bound, the sampling probability is reduced, so that the sample fits in smaller space. Sliding windows were not considered.

The distributed streams model can be contrasted with well-studied communication complexity models [27], where the parties have *unlimited* time and space to process their respective inputs. As observed in [18], simultaneous 1-round communication complexity results can often be related to the distributed streams model. The lower bounds from 1-round communication complexity certainly carry over directly. The distributed streams algorithms in this paper and in [18] are designed for the *stored coins* setting, where all parties can share a string of unbiased and fully independent random bits, but these bits must be stored prior to observing the streams, and the space to store these bits must be accounted for in the workspace bound. Previous work on streaming models (e.g., [1], [10], [11], [13], [23], and [24]) have studied settings with stored coins. Stored coins differ from *private coins* studied in communication complexity [26], [29], [30]. In the stored coins model the same random string can be stored at all parties, whereas in the private coins model the parties are forbidden from sharing any information prior to the start of the computation.

The approximation scheme due to Datar et al. [9] for Basic Counting mentioned in Section 1.2 uses an *exponential histogram* (EH). An EH maintains more information about recently seen items, less about old items, and none at all about items outside the "window" of the last N items. Specifically, the k_0 most recent 1's are assigned to individual buckets, the k_1 next most recent 1's are assigned to buckets of size 2, the k_2 next most recent 1's are assigned to buckets of size 4, and so on, until all the 1's within the last N items are assigned to some bucket. For each bucket, an EH stores only its size (a power of 2) and the position of its most recent 1. Each k_i (up to the last bucket) is either $1/2\varepsilon$ or $1/2\varepsilon + 1$. Upon receiving a new item, the last bucket is discarded if its position no longer falls within the window. Then, if the new item is a 1, it is assigned to a new bucket of size 1. If this makes $k_0 = 1/2\varepsilon + 2$, then the two least recent buckets of size 1 are merged to form a bucket of size 2. If k_1 is now too large, the two least recent buckets of size 2 are merged, and so on, resulting in a cascading of up to $\log N$ bucket merges in the worst case. As we show, our approach using waves avoids this cascading.

More recently, Babcock et al. [3], [4] presented sliding windows algorithms for maintaining a uniform random sample of a specified size, the variance, and a k-median clustering on a single stream. Cohen and Strauss [7] showed how sliding windows algorithms can be used to estimate more general time-decaying aggregates on a single stream.

Distinct values counting has been studied in a number of papers (e.g., [1], [5], [6], [8], [9], [12], [18], and [31]). The seminal algorithm by Flajolet and Martin [12] and its variant due to Alon et al. [1] estimate the number of distinct values in a stream (and also the number of 1's in a bit stream) up to a relative error of $\varepsilon > 1$. The algorithm works in the distributed streams model too, and can be adapted to sliding windows [9]. There are two results we know of that extend this algorithm to work for arbitrary relative error, by Trevisan [31] and by Bar-Yossef et al. [6].[3] Trevisan's algorithm can be extended to distributed streams quite easily, but the cost of extending it to sliding windows is not clear. There are $O(\log(1/\delta))$ instances of the algorithm, using different hash functions, and each must maintain the $O(1/\varepsilon^2)$ smallest distinct hashed values in the sliding window of N values. Assuming the hashed values are random, maintaining just the minimum value over a sliding window takes $O(\log N)$ expected time [9]. We do not know how to extend the algorithm in [6] to sliding windows, and, in addition, its space and time bounds for single streams are worse than ours (however, their algorithm can be made *list efficient* [6]). In work that has appeared since the publication of the preliminary version of this paper [19], Bar-Yossef et al. [5] improve the space complexity of distinct values counting on a single stream, and Cormode et al. [8] show how to compute the number of distinct values in a single stream in the presence of additions *and deletions* of items in the stream. Neither paper considers sliding windows. In this paper we show how the (ε, δ)-approximation scheme in [18] for distinct values counting on distributed streams can be extended to the sliding windows setting, by combining coordinated sampling with waves.

3. Deterministic Waves

We begin by introducing the *wave* data structure, and then describe our ε-approximation scheme for Basic Counting on a single stream. Next, we describe our ε-approximation scheme for estimating the sum of integers in $[0..R]$ in a sliding window on a single stream. Finally, we consider ε-approximation schemes on distributed streams.

3.1. *The Basic Wave*

Recall the Basic Counting problem: given a stream of bits, maintain the number of 1's in a sliding window of the N most recent bits of the stream. We solve a slightly more general problem; our data structure can be used to estimate the number of 1's in *any* sliding window up to a prespecified maximum window size N. The *basic wave* that we describe here is somewhat wasteful in terms of its space bound, processing time, and query time; these are improved in Section 3.2.

Consider a data stream of bits, and a desired bound on the relative accuracy, $0 < \varepsilon < 1$. To simplify the notation, we assume throughout that $1/\varepsilon$ is an integer. We maintain two

[3] Datar et al. [9] also reported an extension to arbitrary relative error for a sliding window over a single stream, using the Trevisan approach [25].

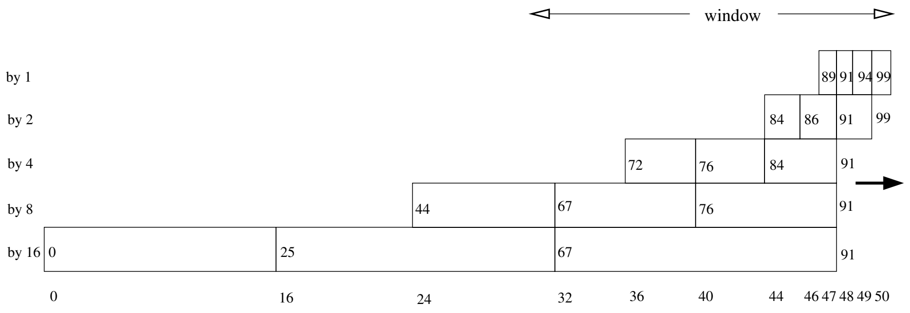

Fig. 2. A deterministic wave for the data stream in Figure 1, and an example window query ($n = 39$).

counters: pos, which is the current length of the stream, and rank, which is the current number of 1's in the stream.

- The wave contains the positions of the recent 1's in the stream, arranged at different "levels." There are $\ell = \lceil \log_2(2\varepsilon N) \rceil$ levels, numbered 0 to $\ell - 1$.
- For $i = 0, 1, \ldots, \ell - 1$, level i contains the positions of the $1/\varepsilon + 1$ most recent 1-bits whose 1-rank is a multiple of 2^i. If there are fewer than $1/\varepsilon + 1$ such 1-bits, level i stores all of them as well as a dummy position 0.

Figure 2 depicts the basic wave for the data stream in Figure 1, for $\varepsilon = \frac{1}{3}$ and $N = 48$. The wave has five levels, with level i labeled as "by 2^i" because it contains the positions of the $1/\varepsilon + 1 = 4$ most recent 1-bits whose 1-ranks are 0 modulo 2^i. The 1-ranks are given on the x-axis. Level 4 has fewer than four 1-bits, so it stores a dummy position 0 with dummy 1-rank 0.

Given this wave, we estimate the number of 1's in a window of size $n \leq N$ using the following steps:

1. If $n \geq$ pos, return $\hat{x} :=$ rank as the exact answer. Otherwise, let $s =$ pos $- n + 1$. (We are estimating the number of 1's in stream positions [s, pos].) Let p_1 be the maximum position stored in the wave that is less than s, and let p_2 be the minimum position stored in the wave that is greater than or equal to s. (If no such p_2 exists, return $\hat{x} := 0$ as the exact answer.)
2. Let r_1 and r_2 be the 1-ranks of p_1 and p_2, respectively. If $s = p_2$, return $\hat{x} :=$ rank $+ 1 - r_2$ as the exact answer. Otherwise, return $\hat{x} :=$ rank $+ 1 - (r_1 + r_2)/2$.

For example, given the window query depicted in Figure 2, we have $n = 39$, pos $= 99$, rank $= 50$, $s = 61$, $p_1 = 44$, $p_2 = 67$, $r_1 = 24$, and $r_2 = 32$, and hence $\hat{x} = 23$. As noted earlier, the actual number of 1's in this window is 20, and indeed $\hat{x} \in [(1 - \varepsilon) \cdot 20, (1 + \varepsilon) \cdot 20] = [\frac{40}{3}, \frac{80}{3}]$.

Lemma 1. *For any window of size $n \leq N$, the above estimation procedure returns an estimate $\hat{x}$ that is within a relative error of ε of the actual number of 1's in the window.*

Proof. If $n \geq$ pos, then the window includes the entire stream, so rank is the exact answer. If p_2 does not exist, then since level 0 does not skip over any 1-bits, it follows that no 1-bits occur in the window. If $s = p_2$, then again the wave returns the exact answer. So consider the general case where none of these three conditions hold.

First, we show that p_1 exists. Note that each level i contains either the dummy position 0 or the positions of exactly $(1/\varepsilon + 1)$ 1's whose 1-ranks are 2^i apart. Thus the smallest 1-rank, r, at level i is either 0 or

$$r \leq \mathsf{rank} - \frac{1}{\varepsilon} \cdot 2^i. \tag{1}$$

If level $\ell - 1$ contains the position 0, then clearly p_1 exists. Otherwise, the difference between rank and the smallest 1-rank in level $\ell - 1$ is at least $2^{\ell-1}/\varepsilon \geq N \geq n$. Since the difference in 1-ranks is at least as large as the difference in position, it follows that p_1 exists. Let j be the smallest numbered level containing position p_1.

Next, we bound the absolute error. We can deduce from the wave that the number of 1's in the window is in $[\mathsf{rank} - r_2 + 1, \mathsf{rank} - r_1]$. For example, it is between $[50 - 32 + 1, 50 - 24]$ in Figure 2. By returning the midpoint of the range, we guarantee that the absolute error is at most $(r_2 - r_1)/2$. By construction, there is at most a 2^j gap between r_1 and its next larger 1-rank r_2. Thus the absolute error in our estimate is at most 2^{j-1}. (Because $s < p_2$, the gap is at least 2, so $j > 0$.)

Finally, to bound the relative error, we show that there are at least $2^{j-1}/\varepsilon$ 1's in the window. Let r_3 be the smallest 1-rank at level $j - 1$. Position p_1 is not in level $j - 1$, so $r_3 > r_1 \geq 0$. Thus by (1), $r_3 \leq \mathsf{rank} - 2^{j-1}/\varepsilon$. Moreover, because r_2 is the smallest 1-rank in the wave larger than r_1, we have $r_2 \leq r_3$. Thus the number of 1's in the window is at least $\mathsf{rank} - r_2 + 1 \geq \mathsf{rank} - r_3 + 1 > 2^{j-1}/\varepsilon$. Therefore, the relative error is less than $1/\varepsilon$. □

3.2. *Improvements*

We now show how to improve the basic wave to get an optimal deterministic wave for a sliding window of size N. Let N' be the smallest power of 2 greater than or equal to $2N$. First, we use modulo N' counters for pos and rank, and store the positions in the wave as modulo N' numbers, so that each takes only $\log N'$ bits, regardless of the length of the stream. Next, we discard or *expire* any positions that are more than N from pos, as these will never be used, and would create ambiguity in the modulo N' arithmetic. We keep track of both the largest 1-rank discarded (r_1, initialized to 0) and the smallest 1-rank still in the wave (r_2), so that the number of 1's in a sliding window of size N can be answered in $O(1)$ time. Processing a 0-bit takes constant time, while processing a 1-bit takes $O(\log(\varepsilon N))$ worst case time and $O(1)$ amortized time, as a 1-bit whose 1-rank is a multiple of 2^i is stored in $i + 1$ levels. Each of these improvements is used for the EH synopsis discussed in Section 2, to obtain the same bounds [9]. However, the steps used to implement these improvements differ because of the differences between deterministic wave synopses and EH synopses.

More significantly, with our waves approach, we can decrease the per-item processing time to $O(1)$ worst case, as follows. Instead of storing a single position in multiple levels, we store each position only at its maximum level, as shown in Figure 3.[4] For

[4] In the figure we have not explicitly discarded positions outside the size $N = 48$ window, in order to

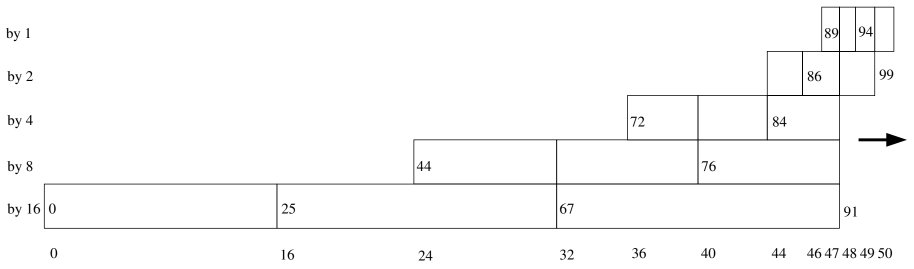

Fig. 3. An optimal deterministic wave for the data stream in Figure 1.

levels $i = 0, \ldots, \ell - 2$, we store $\lceil \frac{1}{2}(1/\varepsilon + 1) \rceil$ positions, and for level $\ell - 1$, we store $\lceil 1/\varepsilon + 1 \rceil$ positions. At all levels, we may store fewer positions, because we discard expired positions and also because initially fewer positions exist.

The positions at each level are stored in a fixed length queue, called a *level queue*, so that each time a new position is added for the level, the position at the tail of the queue is removed (assuming the queue is full). For example, using a circular buffer for each queue, the new head position simply overwrites the next buffer slot. We maintain a doubly linked list of the positions in the wave in increasing order. Positions evicted from the tail of a level queue are spliced out of this list. As each new stream item arrives, we check the head of this sorted list to see if the head needs to be expired.

Finally, as observed in [9], the set of positions is a sorted sequence of numbers between 0 and N', so by storing the difference (modulo N') between consecutive positions instead of the absolute positions, we can reduce the space from $O((1/\varepsilon)\log(\varepsilon N)\log N)$ bits to $O((1/\varepsilon)\log^2(\varepsilon N))$ bits.

Figure 4 depicts the high-level steps of the deterministic wave algorithm (for simplicity, it omits some of the optimizations discussed above).

Summarizing the results of this subsection, we have:

Theorem 1. *Our algorithm maintains a deterministic wave, which can give an estimate for the Basic Counting problem for a sliding window of size N, with relative error at most ε, using $O((1/\varepsilon)\log^2(\varepsilon N))$ bits. Each item is processed in $O(1)$ worst case time. At each time instant, the wave can provide a count estimate in $O(1)$ time.*

Proof. The wave level in step 3(a) is the position of the least-significant 1-bit in rank (numbering from 0). Assuming this is a constant time operation, the time bounds follow from the above discussion.[5] As for the space bound, because the level queues are updated in place, the same block of memory is used throughout, and hence the linked list pointers are offsets into this block and not full-sized pointers. Thus the space bound follows from the above discussion.

show the full levels. All positions less than $\text{pos} - N = 51$ have expired, and $r_1 = 24$ is the largest expired 1-rank.

[5] Below, we show how to determine the wave level in constant time even on a weaker machine model that does not explicitly support this operation in constant time.

```
Upon receiving a bit $b$:
  1. Increment pos.   // All additions and comparisons are done modulo $N'$.
  2. If the head $(p, r)$ of the linked list $L$ has expired (i.e., $p \leq \mathsf{pos} - N$), then discard it from
     $L$ and from (the tail of) its queue, and store $r$ as the largest 1-rank discarded.
  3. If $b = 1$, then do:
     (a) Increment rank, and determine the corresponding level $j$, i.e., the largest $j$ such that
         rank is a multiple of $2^j$.
     (b) If the level $j$ queue is full, then discard the tail of the queue and splice it out of $L$.
     (c) Add (pos, rank) to the head of the level $j$ queue and the tail of $L$.

Answering a query over a sliding window of size $N$:
  1. If $N \geq \mathsf{pos}$, return $\hat{x} := \mathsf{rank}$ as the exact answer. Otherwise, let $r_1$ be the largest 1-rank
     discarded (or 0 if no 1-rank has been discarded). Let $(p_2, r_2)$ be the pair at the head of the
     linked list $L$. (If $L$ is empty, return $\hat{x} := 0$ as the exact answer.)
  2. If $p_2 = \mathsf{pos} - N + 1$, return $\hat{x} := \mathsf{rank} + 1 - r_2$ as the exact answer. Otherwise, return
     $\hat{x} := \mathsf{rank} + 1 - (r_1 + r_2)/2$.
```

Fig. 4. A deterministic wave algorithm for Basic Counting on a single stream.

The proof of ε relative error follows from the proof of Lemma 1. This can be seen by considering executions of the algorithm and the Basic Wave algorithm on the same data stream, and then considering a query q. Let B be the set of positions in the Basic Wave that lie within q's window, together with the position p_1 used to answer q. The error bound follows because the set of positions in the improved wave (including the position corresponding to r_1) is the same or a superset of B. □

Optimality of Bounds. The time bounds are optimal because they are constant worst case bounds. The space bound is optimal because it matches the lower bound by Datar et al. [9] for both randomized and deterministic algorithms. We state their deterministic lower bound theorem below; they obtain similar bounds for randomized algorithms.

Theorem 2 (from [9]). *Any deterministic algorithm that provides an estimate for the Basic Counting problem at every time instant with relative error less than $1/k$ for some integer $k \leq 4\sqrt{N}$ requires at least $(k/16)\log^2(N/k)$ bits of memory.*

Computing the Wave Level on a Weaker Machine Model. Step 3(a) of Figure 4 requires computing the least-significant 1-bit in a given number (rank). On a machine model that does not explicitly support this operation in constant time, a naive approach would be to examine each bit of rank one at a time until the desired position is found. However, this takes $\Theta(\log N)$ worst case time, because rank has $\log(N')$ bits. Instead, we store the $\log(N') - 1$ level numbers associated with the sequence $1, \ldots, \log(N') - 1$ in an array (e.g., $\{0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0\}$ if $\log(N') = 16$). This takes less than $\log(N') \cdot \log\log(N')$ bits. We also store a counter d of $\log(N') - \log\log(N')$ bits, initially 1. As 1-bits are received, the desired wave level is the next element in this array. The first 1-bit after reaching the end of the array has the property that the last $\log\log(N')$ bits of rank are 0, and the desired level is $\log\log(N')$ plus the position of the least-significant

1-bit in d (numbering from 0). We then increment d and return to cycling through the array, starting at the beginning of the array with the arrival of the next 1-bit. This correctly computes the wave level at each step. Moreover, note that while we are cycling through the array, we have $\log(N')$ steps until we need to know the least-significant 1-bit in d. Thus by interleaving (i) the cycling and (ii) the search through the bits of d, we can determine the wave level in $O(1)$ worst case time.

Basic Counting for any Window of Size $n \leq N$. The algorithm in Figure 4 achieves constant worst case query time for a sliding window of size N. For a sliding window of size $n \leq N$, this single wave can be used to give an estimate for the Basic Counting problem that is within an ε relative error, by following the two steps outlined for the Basic Wave (Section 3.1). The query time for window sizes less than N is $O((1/\varepsilon)\log(\varepsilon N))$ in the worst case, because we must search through the linked list L in order to determine p_1 and p_2. This matches the query time bound for the EH algorithm [9]. The precise statement of the bounds obtained by our algorithm will be given after we consider a further generalization, described next.

Sliding Windows with Duplicated Positions. A simple generalization of the Basic Counting problem permits multiple stream items with the same "position." That is, each stream item is a (position, bit-value) pair such that the positions are consecutive integers with possible repetitions, arriving in nondecreasing order. An example of such a stream is

$$(1, 0), (2, 1), (2, 0), (2, 1), (2, 1), (3, 1), (4, 0), (4, 0), \ldots.$$

This scenario can arise when "positions" are increasing time units, and we target a sliding window over the last N time units.

The algorithms of this section can easily be adapted to handle this scenario, as long as we have an upper bound, U, on the number of stream items that can occur in any sliding window. Specifically, we make the following changes. We have $\lceil \log_2(2\varepsilon U) \rceil$ levels, and let N' be the smallest power of 2 greater than or equal to $2U$. We let r_1 (r_2) be the largest (smallest) 1-rank among those with position p_1 (p_2, respectively). We add and maintain a doubly linked list between the *first* items for each position in the linked list L. This enables *all* the items for an expiring position to be discarded in $O(1)$ time in step 2 of Figure 4. Note that the largest 1-rank being discarded is the 1-rank of the expiring item that was adjacent to what becomes the new head of L; thus it can be determined in $O(1)$ time prior to discarding. Moreover, this additional linked list is easily maintained in $O(1)$ time when items are spliced out of L in step 3(b). The bounds obtained by this algorithm are given in the following corollary.

Corollary 1. *Consider a stream where duplicate positions are permitted, and let U be an upper bound on the number of stream items that can occur in any sliding window of N positions. The algorithm maintains a deterministic wave, which can give an estimate for the Basic Counting problem for any window of $n \leq N$ positions, with relative error at most ε, using $O((1/\varepsilon)\log^2(\varepsilon U))$ bits. Each item is processed in $O(1)$ worst case time. Each estimate takes $O(1)$ worst case time for the special case when $n = N$, and $O((1/\varepsilon)\log(\varepsilon N))$ worst case time for general $n < N$.*

Proof. The time and space bounds are immediate, given the above discussion. The proof of ε relative error follows along the lines of the proofs of Lemma 1 and Theorem 1. The one subtlety for the duplicated positions scenario is to show that the desired p_1 exists in the wave, as this is the only aspect of the proofs that crucially depends on the correspondence between positions and 1-ranks. (Other cases where position is important, such as when the exact answer can be returned, are immediate given the definitions of p_2 and r_2.) To see that p_1 exists, observe that having $\lceil \log_2(2\varepsilon U) \rceil$ levels ensures that level $\ell - 1$ either contains the dummy position 0 or spans at least U 1-bits. Hence, the window is contained within level $\ell - 1$ because by definition the sliding window can have at most U stream items. □

Note that the bounds for the scenario without duplicates are obtained by setting $U = N$.

3.3. *Sum of Bounded Integers*

The deterministic wave scheme can be extended to handle the problem of maintaining the sum of the last N items in a data stream, where each item is an integer in $[0..R]$. Datar et al. [9] showed how to extend their EH approach to obtain an ε-approximation scheme for this problem, using $O((1/\varepsilon)(\log N + \log R))$ buckets of $\log N + \log(\log N + \log R)$ bits each, $O(1)$ query time, and $O(\log R / \log N)$ amortized and $O(\log N + \log R)$ worst case per-item processing time. (They also presented a matching asymptotic lower bound on the number of bits.[6]) We show how to achieve *constant worst case* per-item processing time, while using the same number of memory words and the same query time.[7]

Our algorithm is depicted in Figure 5. The sum over a sliding window can range from 0 to $N \cdot R$. Let N' be the smallest power of 2 greater than or equal to $2NR$. We maintain two modulo N' counters: pos, the current length, and total, the running sum. There are $\ell = \lceil \log(2\varepsilon NR) \rceil$ levels. A level is *full* if it has $\lceil 1/\varepsilon + 1 \rceil$ positions. The algorithm follows the same general steps as the algorithm in Figure 4. Instead of storing pairs (p, r), we store triples (p, v, z) where v is the value for the data item (not needed before because the value for a stored item was always 1) and z is the partial sum through this item (the equivalent of the 1-rank for sums). When answering a query, we know that the window sum is in $[\mathsf{total} - z_2 + v_2, \mathsf{total} - z_1]$, where (p, v_2, z_2) is the triple at the head of the linked list L and z_1 is the largest partial sum discarded, and we return the midpoint of this interval.

The key insight in this algorithm is that it suffices to store an item (only) at a level j such that 2^j is the largest power of 2 that divides a number in $(\mathsf{total}, \mathsf{total} + v]$. Naively, one would mimic the Basic Counting wave by viewing a value v as v items of value 1. However, this would lead to $O(R)$ worst case processing time per item. Datar et al. [9] reduced the time to $O(\log N + \log R)$ by directly computing the EH resulting from v insertions of value 1. However, a single item is stored in up to $O(\log N + \log R)$ EH buckets. In contrast, we store the item just once, which enables our $O(1)$ time bound.

[6] The lower bound assumes that if $R > N$, then $\log R = O(N^{\delta})$ for some $\delta < 1$.

[7] Measured in *bits* instead of words, our bound of $O((1/\varepsilon)(\log N + \log R)^2)$ bits is slightly worse than theirs when R is superpolynomial in N.

```
Upon receiving an item with value $v \in [0..R]$:
  1. Increment pos.    // All additions and comparisons are done modulo $N'$
  2. If the head $(p, v', z)$ of the linked list $L$ has expired (i.e., $p \leq \mathsf{pos} - N$), then discard it
     from $L$ and from (the tail of) its queue, and store $z$ as the largest partial sum discarded.
  3. If $v > 0$, then do:
     (a) Determine the wave level, i.e., the largest $j$ such that some number in $(\mathsf{total}, \mathsf{total} + v]$
         is a multiple of $2^j$. Add $v$ to total.
     (b) If the level $j$ queue is full, then discard the tail of the queue and splice it out of $L$.
     (c) Add $(\mathsf{pos}, v, \mathsf{total})$ to the head of the level $j$ queue and the tail of $L$.

Answering a query over a sliding window of size $N$:
  1. If $N \geq \mathsf{pos}$, return $\hat{x} := \mathsf{total}$ as the exact answer. Otherwise, let $z_1$ be the largest partial
     sum discarded from $L$ (or 0 if no partial sum has been discarded). Let $(p, v_2, z_2)$ be the
     triple at the head of the linked list $L$. (If $L$ is empty, return $\hat{x} := 0$ as the exact answer.)
  2. If $p = \mathsf{pos} - N + 1$, return $\hat{x} := \mathsf{total} - z_2 + v_2$ as the exact answer. Otherwise, return
     $\hat{x} := \mathsf{total} - (z_1 + z_2 - v_2)/2$.
```

Fig. 5. A deterministic wave algorithm for the sum in a sliding window on a single stream.

The challenge is to compute the wave level in step 3(a) quickly; we show how to do this in $O(1)$ time. First observe that the desired wave level is the largest position j (numbering from 0) such that some number y in the interval $(\mathsf{total}, \mathsf{total} + v]$ has 0's in all bit positions less than j (and hence is a multiple of 2^j). Second, observe that $y - 1$ and y differ in bit position j, and if this bit changes from 1 to 0 at any point in $[\mathsf{total}, \mathsf{total} + v]$, then j is not the largest. Thus, j is the position of the most-significant bit that is 0 in total and 1 in $\mathsf{total} + v$. Accordingly, let f be the bitwise complement of total, and let $g = \mathsf{total} + v$. Let $h = f \wedge g$, the bitwise AND of f and g. Then the desired wave level is the position of the most-significant 1-bit in h, i.e., $\lfloor \log h \rfloor$.[8]

Summarizing the results of this subsection, we have:

Theorem 3. *The algorithm in Figure 5 is an ε-approximation scheme for the sum of the last N items in a data stream, where each item is an integer in $[0..R]$. It uses $O((1/\varepsilon)(\log N + \log R))$ memory words, where each memory word is $O(\log N + \log R)$ bits (i.e., large enough to hold an item or a window size). Each item is processed in $O(1)$ worst case time. At each time instant, it can provide an estimate in $O(1)$ time.*

Proof. To prove the approximation guarantee, we first define a basic wave corresponding to the input stream for the sums of integers algorithm, and show how the sum wave constructed by the algorithm simulates this basic wave. The approximation guarantee follows from the approximation guarantee provided by the basic wave.

The basic wave is defined as follows. For each triple (p, v, z) that is input to the algorithm (where p is the position, v is the value of the integer, and z is the current

[8] On a weaker machine model that does not support this operation on h in constant time, we can use binary search to find the desired position in $O(\log(\log N + \log R))$ time, as follows. Let w be the word size, and let M be a bit mask comprising of $w/2$ 1's followed by $w/2$ 0's. We begin by checking if $h \wedge M$ equals 0. If so, we left shift M by $w/4$ positions and recurse. Otherwise, we right shift M by $w/4$ positions and recurse.

running total inclusive of v), we have a pair (p, i) input to the basic wave for each $i \in [z - v + 1, z]$. Thus we have the stream model discussed at the end of Section 3.2 where duplicated positions are permitted. Note that the basic wave stores an item (p, r) in every level l such that r is a multiple of 2^l.

Let $N < \mathsf{pos}$ and $s = \mathsf{pos} - N + 1$; we are interested in the interval $[s, \mathsf{pos}]$. Let X be the estimate returned by this basic wave for the number of 1-bits in the window $[s, \mathsf{pos}]$. We know from Corollary 1 (taking $U = N \cdot R$) that the relative error between X and the actual value is at most ε. By construction, the number of 1-bits input to the basic wave in the interval $[s, \mathsf{pos}]$ exactly equals the sum of the integers in the interval $[s, \mathsf{pos}]$ that the sum wave is trying to estimate. Thus the relative error between X and the sum of integers in the window $[s, \mathsf{pos}]$ is at most ε.

Suppose p_1 is the maximum position in the basic wave less than s, p_2 is the minimum position in the basic wave greater than or equal to s, r_1 is the largest 1-rank with position p_1, and r_2 is the smallest 1-rank with position p_2. The absolute error of the basic wave can be as much as $(r_2 - r_1)/2$, and this is sufficient for the approximation guarantee. We now show that the absolute error of the sum wave in estimating the sum of integers in $[s, \mathsf{pos}]$ is not greater than $(r_2 - r_1)/2$, and this will complete the proof.

Suppose the integers that gave rise to (p_1, r_1) and (p_2, r_2) were (p_1, v_1, z_1) and (p_2, v_2, z_2), respectively. We now show that (p_2, v_2, z_2) expires from the sum wave at the same time or later than when (p_2, r_2) expires from the basic wave. Suppose (p_2, v_2, z_2) entered the sum wave at level l_s, and (p_2, r_2) entered the basic wave at level l_b. Firstly, $l_s \geq l_b$, since l_s is the largest power of 2 that divides a number in $[z_2 - v_2 + 1, z_2]$, while l_b is the largest power of 2 that divides r_2, where $r_2 \in [z_2 - v_2 + 1, z_2]$. We next show that for every tuple entering the sum wave at level l_s, there is at least one tuple (perhaps more) entering the basic wave at level l_b. To see this, suppose tuple (p, v, z) entered the sum wave at level l_s after (p_2, v_2, z_2) did. Then tuple (p, r) enters the basic wave at level l_s for some $r \in [z - v + 1, z]$. Since $l_b \leq l_s$, (p, r) also enters the basic wave at level l_b. The above argument shows that (p_2, v_2, z_2) expires from the sums wave at the same time or later than when (p_2, r_2) expires from the basic wave.

By a similar argument, we can also show that (p_1, v_1, z_1) expires from the sum wave at the same time or later than when (p_1, r_1) expires from the sum wave. The absolute error made by the sum wave is thus not greater than $(z_2 - v_2 - z_1)/2$ which is less than or equal to $(r_2 - r_1)/2$, since $r_2 \geq z_2 - v_2$ and $r_1 \leq z_1$.

The space and time bounds are immediate, given the above discussion of how to perform step 3(a) in constant time. $\square$

3.4. *Distributed Streams*

Recall the distributed streams model. There are $t \geq 2$ parties, and each party observes only its own stream. When an estimate is requested, each party sends a message to a Referee who computes the estimate over the *union* of all the streams. Below, we consider three natural definitions for a sliding window over the union of distributed streams, as illustrated for the Basic Counting problem:

Scenario 1: We seek the total number of 1's in the last N items of each of the t streams ($t \cdot N$ items in total).

Scenario 2: A single logical stream has been split arbitrarily among the parties. Each party receives items that include an overall sequence number and the value of the bit, and we seek the total number of 1's in the last N items of the logical stream.

Scenario 3: We seek the total number of 1's in the last N items over the positionwise union (logical OR) of the t streams.

For the first and the second scenarios, the deterministic wave can be used to answer sliding windows queries over a collection of distributed streams. For the first scenario, we apply the single stream algorithm to each stream. To answer a query, each party sends its count to the Referee, who simply sums the answers. Since each individual count is within ε relative error of the actual, so is the total.

In the second scenario we assume that when the estimate is requested from the parties, each party is sent the current overall sequence number, pos, in the logical stream. Then each party separately estimates the number of items in its stream whose overall sequence numbers lie in the interval $[\mathsf{pos} - N + 1, \mathsf{pos}]$ using the deterministic wave. Note that this interval is not necessarily the last N items in the stream observed by the party. Instead, it is guaranteed to lie within the window of the last N items observed by the party. By Corollary 1, each party can separately estimate the number of relevant items in its stream to within a relative error of ε. Each individual count is within ε relative error of the actual, and hence so is the total.

However, the third scenario is more problematic. Denote as the *Union Counting* problem the problem of counting the number of 1's in the positionwise union of t distributed data streams. (If each stream represents the characteristic vector for a set, then this is the size of the union of these sets.) We show the following lower bound on *any* deterministic algorithm for this problem, even for the case not involving sliding windows.

Theorem 4. *Any deterministic algorithm that guarantees a constant relative error $\varepsilon \le \frac{1}{64}$ for the Union Counting problem requires $\Omega(n)$ space for n-bit streams, even for $t = 2$ parties (and no sliding window).*

Proof. The proof is by contradiction. Suppose that an algorithm existed for approximating Union Counting to within a relative error of $\varepsilon = \frac{1}{64}$ using space less than αn, where $\alpha = \frac{1}{16}$. (We have not attempted to maximize the constants ε or α.)

Let A and B be the two parties, and let C be the Referee. Let X be the data stream seen by A and let Y be the stream seen by B. Data streams X and Y are of length n (n even), and a query request occurs only after both streams have been observed. Suppose that both X and Y have exactly $n/2$ 1's and 0's. For this restricted scenario, the exact answer for the Union Counting problem is

$$\frac{n}{2} + \frac{1}{2} H(X, Y), \tag{2}$$

where $H(X, Y)$ is the Hamming distance between X and Y.

For each possible message m from A to C, let S_m denote the set of all inputs to A which cause A to transmit m. Since A's workspace is only αn bits, the number of distinct messages that A could send C is $2^{\alpha n}$. The number of possible inputs to A is $\binom{n}{n/2}$. Using the pigeonhole principle, we conclude that there exists a message m that A sends to C such that

$$|S_m| \geq \frac{\binom{n}{n/2}}{2^{\alpha n}}. \tag{3}$$

Because the relative error is at most ε and the exact answer is at most n, the absolute error of any estimate produced by the algorithm is at most $n\varepsilon$. We claim that no two inputs in S_m can be at a Hamming distance greater than $4n\varepsilon$. The proof is by contradiction. Suppose that there are two inputs X_1 and X_2 in S_m such that $H(X_1, X_2) > 4n\varepsilon$. Consider two runs of the algorithm: in the first, $X = X_1$ and $Y = X_2$, and in the second, $X = X_2$ and $Y = X_2$. In both runs the Referee C gets the same pair of messages, and hence outputs the same estimate z. Because the absolute error in both cases is at most $n\varepsilon$, we have by (2) that $z \geq n/2+\frac{1}{2}H(X_1, X_2)-n\varepsilon > n/2+n\varepsilon$ and $z \leq n/2+\frac{1}{2}H(X_2, X_2)+n\varepsilon = n/2+n\varepsilon$, a contradiction.

For a given n-bit string s with exactly $n/2$ 1's, the number of n-bit strings with exactly $n/2$ 1's at a Hamming distance of k from s (k an even number) is $\binom{n/2}{k/2}^2$ — all combinations of $k/2$ out of $n/2$ 0's in s flipped to 1's and $k/2$ out of $n/2$ 1's in s flipped to 0's. (There are no such inputs at odd distances.) Thus the number of such strings at Hamming distance at most k is $\sum_{j=0}^{k/2}\binom{n/2}{j}^2$, which, for $k \leq n/2$, is at most $(1 + k/2)\binom{n/2}{k/2}^2$. Setting $k = 4n\varepsilon$ in the above claim, for all messages m that A sends to C, we have

$$|S_m| \leq (1 + 2n\varepsilon)\binom{n/2}{2n\varepsilon}^2. \tag{4}$$

By choosing $\alpha = \frac{1}{16}$ in (3) we get

$$|S_m| \geq \frac{\binom{n}{n/2}}{2^{\alpha n}} \geq \frac{2^{n/2}}{2^{\alpha n}} = 2^{7n/16}.$$

By choosing $\varepsilon = \frac{1}{64}$ and n suitably large, it follows from (4) that

$$|S_m| \leq (1 + 2n\varepsilon)\left(\frac{e}{4\varepsilon}\right)^{4n\varepsilon} < 2^{24n/64+\log(1+n/32)} < 2^{7n/16}$$

We obtain the contradiction, which completes the proof. □

For the sum of bounded integers problem on distributed streams, scenario 1 is a straightforward application of the single stream algorithm. For scenario 2, we again use the fact that our deterministic stream algorithm for window size N can provide estimates for any window size n such that $n \leq N$, and this scenario can also be solved by each stream separately executing the single stream algorithm at each party. For scenario 3, if "union" means to take the positionwise sum, the problem reduces to scenario 1. If "union" means to take the positionwise maximum, then the lower bound in Theorem 4 applies, as the number of 1's in the union is a special case of the sum of the positionwise maximum.

The linear space lower bound for deterministic algorithms in Theorem 4 is the motivation for considering *randomized* waves, which are introduced in the next section.

4. Randomized Waves

This section describes the randomized wave data structure for the Union Counting problem. Like the deterministic wave, the randomized wave consists of the 1-bits (more accurately, the positions of the 1-bits) in the data stream stored at different levels. Each level contains the positions of some of the most recently selected 1-bits, where a position is selected into level i with probability 2^{-i}. Thus the difference between the deterministic and randomized waves is that for each level i, the deterministic wave selects one out of every 2^i 1-bits at *regular* intervals, whereas a randomized wave selects an *expected* one out of every 2^i 1-bits at *random* intervals. Also, the randomized wave retains more positions per level than does the deterministic wave.

4.1. *The Randomized Wave*

We now describe the data structure, show how it yields an (ε, δ)-approximation scheme for Union Counting over any sliding window up to a prespecified maximum window size N, and then present an analysis of the time and space complexity.

Let N' be the minimum power of 2 that is at least $2N$; let $d = \log N'$. Let $\varepsilon < 1$ be the desired bound for relative error, and let $\delta < 1$ be the upper bound on the probability that the algorithm fails to achieve this bound. The randomized wave for party j consists of $d + 1$ queues, $Q_j(0), \ldots, Q_j(d)$, one for each level $l = 0 \cdots d$. Each queue can hold c/ε^2 items, for a constant c determined by the analysis.

Hash Function. We use a pseudorandom hash function h to map positions to levels, according to an exponential distribution. The input to h is a position modulo N', i.e., $\{0 \cdots N'-1\}$. For $l = 0, 1, \ldots, d-1$, $\mathbf{Pr}\{h(p) = l\} = 1/2^{l+1}$ and $\mathbf{Pr}\{h(p) = d\} = 1/2^d$. The hash function $h()$ is computed as follows.

We consider the numbers $\{0 \cdots N' - 1\}$ as members of the field $G = GF(2^d)$. In a preprocessing step we choose q and r uniformly and independently at random from G and store them with each party. In order to compute $h(p)$, a party computes $x = q \cdot p + r$ (all operations being performed in G). We represent x as a d-bit vector and then $h(p)$ is the largest i such that the i most-significant bits of x are 0 (i.e., $h(p) = d - \lfloor \log x \rfloor - 1$). Clearly, $h(p) \in [0..d]$. The two properties of h that we use are:

1. x is distributed uniformly over G. Hence, for $l < d$, $\mathbf{Pr}\{h(p) = l\} = 1/2^{l+1}$.
2. The mapping is pairwise independent, i.e., for distinct p_1 and p_2, $\mathbf{Pr}\{(h(p_1) = k_1) \wedge (h(p_2) = k_2)\} = \mathbf{Pr}\{h(p_1) = k_1\} \cdot \mathbf{Pr}\{h(p_2) = k_2\}$.

This is the same hash function used by Alon et al. [1], except that the domain and range sizes now depend only on the maximum window size N and not on the entire stream length.

Party P_j, upon receiving a stream bit b:

```
1. Increment pos.   (Note: All additions and comparisons are done modulo $N'$.)
2. Discard any position $p$ in the tail of a queue that has expired (i.e., $p \le \mathsf{pos} - N$).
3. If $b = 1$, then for $l := 0, \ldots, h(\mathsf{pos})$ do:   (Note: All parties use the same function $h$.)
   (a) If the level $l$ queue $Q_j(l)$ is full, then discard the tail of $Q_j(l)$.
   (b) Add pos to the head of $Q_j(l)$.
```

Answering a query for a sliding window of size $n \le N$, after each party has observed pos bits:

```
1. Let $s := \max(0, \mathsf{pos} - n + 1)$. ($[s, \mathsf{pos}]$ is the desired window.) Each party $j$ determines
   the smallest numbered level, $l_j$, such that the tail of $Q_j(l_j)$ is a position $p \le s$, and sends
   $l_j$ and $Q_j(l_j)$ to the Referee.
2. The Referee computes $l^* := \max_{j=1,\ldots,t} l_j$. For $j = 1, \ldots, t$, let $U_j$ be the set of positions
   $p$ in $Q_j(l_j)$ such that $h(p) \ge l^*$ and $p \ge s$.
3. The Referee returns $\hat{x} := 2^{l^*} \cdot |\bigcup_{j=1}^{t} U_j|$.
```

Fig. 6. A randomized wave algorithm for Union Counting in a sliding window (t streams).

Updating the Randomized Wave. The steps for maintaining the randomized wave are summarized in the top half of Figure 6. A 1-bit arriving at position p is selected into levels $0, \ldots, h(p)$. The sample for each level, stored in a queue, contains the c/ε^2 most recent positions selected into that level. ($c = 36$ is a constant determined by the analysis.) Consider a queue $Q_j(l)$, whose tail (earliest element) is at position i. Then $Q_j(l)$ contains all the 1-bits in the interval $[i, \mathsf{pos}]$ whose positions hash to a value greater than or equal to l. We call this interval $[i, \mathsf{pos}]$ the *range* of $Q_j(l)$. As we move from level l to $l+1$, the range may increase, but it will never decrease. The reason is as follows. Every bit that is selected into level $l+1$ is also selected into level l. Since the queue at a level contains the c/ε^2 most recent positions selected into that level, it is possible that $Q_j(l+1)$ contains earlier positions than $Q_j(l)$ but never vice versa. We assume that each level is initialized with a "dummy" position 0.

Handling a Query. The bottom half of Figure 6 summarizes the steps for answering a query. We receive a query for the number of 1's in the interval $W = [\max(0, \mathsf{pos} - n + 1), \mathsf{pos}]$. In step 1 each party j selects the lowest numbered level l_j such that the range of $Q_j(l_j)$ contains W, and sends l_j and $Q_j(l_j)$ to the Referee. Let l^* be the maximum of these levels over all the parties. Each queue $Q_j(l_j)$ contains the positions, U_j, of *all* the 1-bits in W in its stream that hash to a value at least l^* (along with any other positions outside of W or that hash to values in $[l_j..l^*)$). In step 2 the Referee computes l^* and extracts the U_j. Finally, in step 3, the Referee takes the union of the positions in the t queues, and returns the number of positions in this union, scaled up by a factor of 2^{l^*}.

4.2. *Proof of Correctness*

We use the following lemma which has been proved in [18].

Lemma 2 (from [18]). *Consider x items $\{1, 2, 3, \ldots, x\}$ which are sampled into levels starting from level 0 as follows. Item i is placed in levels from 0 through $h(i)$ (where*

h has been defined above). Denote the number of items at level j by x_j. Let ℓ be the smallest numbered level such that $x_\ell \leq c/\varepsilon^2$, where $c = 36$, and $\varepsilon > 0$. The estimate for x using level j is $\hat{x}_j = x_j 2^j$. Then, for any level $j \leq \ell$, we have

$$\mathbf{Pr}\{|\hat{x}_j - x| < \varepsilon x\} > \tfrac{2}{3}.$$

Lemma 3. *The Randomized Wave Algorithm returns an estimate $\hat{x}$ for Union Counting over a sliding window of size $n \leq N$ that is within a relative error of ε with probability greater than $2/3$.*

Proof. Consider party j. For each level l, define $S_j(l) = \{i | (l \leq h(i)) \wedge (b_i = 1 \text{ in stream } j)\}$. The queue at level l, $Q_j(l)$, contains the positions of the c/ε^2 most recent 1-bits in $S_j(l)$. Consider the size of $S_j(l) \cap W$. This is large for small l and decreases as l increases. If $|S_j(l) \cap W| > c/\varepsilon^2$, then the range of $Q_j(l)$ does not contain W, and vice versa.

Thus, the level selected at party j, l_j, is the smallest value of l such that $|S_j(l) \cap W| \leq c/\varepsilon^2$. In other words, l_j is the smallest value of l such that the number of relevant 1-bits (those in window W) that were selected into l is less than c/ε^2 for stream j, and l^* is the lowest numbered level that satisfies the property for every stream. Our estimate is $\hat{x} = 2^{l^*} |\{S_1(l^*) \cup S_2(l^*) \cup \cdots \cup S_t(l^*)\} \cap W|$.

Now, consider the stream formed by the (bitwise) union of all the streams observed by the various parties. We are interested in estimating x, the number of 1-bits in this union stream that lie in the window W. The 1-bits in the union stream are being sampled into different levels by the same process as described in Lemma 2. Let ℓ^* be the smallest numbered level such that the number of bits in the union that lie in W and have been sampled into ℓ^* is less than c/ε^2. Our algorithm finally estimates x using the sample at level l^*, and clearly $l^* \leq \ell^*$. By applying Lemma 2 the proof is complete. □

Note that the above approximation guarantee holds independent of the number of parties. By taking the median of $O(\log(1/\delta))$ independent instances of the algorithm, we get our desired (ε, δ)-approximation scheme:

Theorem 5. *The above estimation procedure is an (ε, δ)-approximation scheme for Union Counting over any sliding window of size not greater than N, using $O((\log(1/\delta)\log^2 N)/\varepsilon^2)$ bits per party. The time to process an item is dominated by the time for an expected $O(\log(1/\delta))$ finite field operation. The query time is $O(t \log(1/\delta)(\log\log(N) + 1/\varepsilon^2))$.*

Proof. For each of the $O(\log(1/\delta))$ instances, we have $O(\log N)$ queues of $O(1/\varepsilon^2)$ positions, and each position is $O(\log N)$ bits. Also, for each instance, we have the hash function parameters, q and r, which are $O(\log N)$ bits each. This shows the space bound.

The per-item processing is $O(1)$ expected time per instance because the expected number of levels to which the new position is added is bounded by 2 (step 3), and likewise the expected number of levels that position $p = \mathsf{pos} - n$ was ever in, where n is the size of the sliding window, is bounded by 2. Thus scanning the tails of the queues at levels $0, \dots, h(p)$ looking for p (step 2) takes constant expected time. Since there are $O(\log(1/\delta))$ instances, the time bound follows.

As for the query time, each party j can determine l_j for one of its instances in $O(\log\log(N'))$ time using binary search. Each party sends $O(\log(1/\delta)/\varepsilon^2)$ memory words, which the Referee receives and processes. □

5. Extensions

Number of Distinct Values. With minor modifications, the randomized wave algorithm can be used to estimate the number of distinct values in a sliding window over the union of distributed streams. Each element of the sample (wave) is now an ordered pair (p, v) where v is a value that was seen in the stream and p is the position of the most recent occurrence of the value. This is updated every time the value appears again in the stream. A wave at level l stores the c/ε^2 elements with the most recent positions that were sampled into that level. Note that the hash function now hashes the value of the element, rather than its position (as was the case in Union Counting).

Each party maintains pos, the length of its observed stream. It also maintains a (doubly linked) list of all the elements in its wave, ordered by position. This list lets the party discard expired elements.

When an element v arrives, we insert it into levels $0 \cdots h(v)$, storing pos alongside. Since the expected value of $h(v)$ is less than 2, it is inserted into an expected constant number (in this case, two) of levels. If v is already present in the wave, we update its position. To determine the presence of a value in the wave, we use an additional hash table (hashed by an item's value) that contains a pointer to the occurrence of the element in the doubly linked list. Updating an element's position requires moving the element from its current position to the tail of the list, and this can be done in constant time. The element's position has to be updated in each of the levels to which it belongs. All this can be done in constant expected time, because each element belongs to an expected constant number of levels.

To produce an estimate, each party passes its wave to the Referee. The Referee constructs a wave of the union by computing a levelwise union of all the waves that it receives. This resulting wave is used for the estimation. As before, we perform $O(\log(1/\delta))$ independent instances of the algorithm, and take the median. The space bound and proof of correctness are identical to the randomized wave for Union Counting. Summarizing the result, we have:

Theorem 6. *The above estimation procedure is an (ε, δ)-approximation scheme for the number of distinct values in a sliding window of size N over distributed streams. It uses $O((\log(1/\delta)\log N \log R)/\varepsilon^2)$ bits per party, where the values are in $[0..R]$, and the per-item processing time is dominated by the time for an expected $O(\log(1/\delta))$ finite field operation. The query time is $O(t\log(1/\delta)(\log\log(N) + 1/\varepsilon^2))$.*

Handling Predicates. Note that our algorithm for the number of distinct values stores a random sample of the distinct values. This sample can be used to answer queries about predicates on the distinct values (e.g., how many *even* distinct values are there?). In order to provide an (ε, δ)-approximation scheme for *any* predicate of selectivity at least α (i.e., at least an α fraction of the distinct values satisfy the predicate), *where the predicate is not known until query time*, we store a sample of size $O(1/\alpha\varepsilon^2)$ at each level, increasing our space bound by a factor of $1/\alpha$. Such problems without sliding windows were studied in [16].

Nth Most Recent 1. We can use the wave synopsis to obtain an (ε, δ)-approximation scheme for the position of the Nth most recent 1 in the stream, as follows. Instead of storing only the 1-bits in the wave, we store both 0's and 1's. Thus, items in level l are 2^l positions apart, not 2^l 1's apart. In addition, we keep track of the 1-rank of the 1-bit closest to each item in the wave. The rest of the algorithm is similar to our Basic Counting scheme. Note that we need $O((1/\varepsilon)\log^2(\varepsilon m))$ bits, where m is an upper bound on the size of the window needed in order to contain the N most recent 1's.

Other Problems. Our improved time bounds for distinct counting for a single stream lead to improved time bounds for all problems which reduce to distinct counting, as described in [9]. These include L_p norms, averages, histogramming, etc. For example, an ε-approximation scheme for the sliding average is readily obtained by running our sum and count algorithms (each targeting a relative error of $\varepsilon/(2+\varepsilon)$).

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc*. 21*st ACM Sigmod–Sigact–Sigart Symp. on Principles of Database Systems* (*PODS*), pages 1–16, June 2002.

[3] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proc*. 13*th ACM–SIAM Symp. on Discrete Algorithms* (*SODA*), pages 633–634, Jan. 2002.

[4] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proc*. 22*nd ACM Sigmod–Sigact–Sigarch Symp. on Principles of Database Systems* (*PODS*), pages 234–243, June 2003.

[5] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proc*. 6*th International Workshop on Randomization and Approximation Techniques* (*RANDOM*), pages 1–10. Lecture Notes in Computer Science, vol. 2483. Springer-Verlag, Berlin, Sept. 2002.

[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc*. 13*th ACM–SIAM Symp. on Discrete Algorithms* (*SODA*), pages 623–632, Jan. 2002.

[7] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc*. 22*nd ACM Sigmod–Sigact–Sigarch Symp. on Principles of Database Systems* (*PODS*), pages 223–233, June 2003.

[8] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *Proc*. 28*th International Conf. on Very Large Data Bases* (*VLDB*), pages 335–345, Aug. 2002.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[10] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. Technical report, AT&T Shannon Laboratories, Florham Park, NJ, July 1999.

[11] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1-difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1):131–151, 2002.

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

[13] J. Fong and M. Strauss. An approximate L^p-difference algorithm for massive data streams. In *Proc*. 17*th Symp*. *on Theoretical Aspects of Computer Science*, pages 193–204. Lecture Notes in Computer Science, vol. 1770. Springer-Verlag, Berlin, Feb. 2000.

[14] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: mining and monitoring evolving data. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):50–63, Jan.-Feb. 2001.

[15] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proc*. *ACM SIGMOD International Conf. on Management of Data* (*SIGMOD*), pages 13–24, May 2001.

[16] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proc*. 27*th International Conf. on Very Large Data Bases* (*VLDB*), pages 541–550, Sept. 2001.

[17] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In J. M. Abello and J. S. Vitter, editors, *External Memory Algorithms*, pages 39–70. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 50. American Mathematical Society, Providence, RI, 1999. Also, a two-page summary of this paper appeared in *Proc*. *SODA* '99.

[18] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proc*. *ACM Symp*. *on Parallel Algorithms and Architectures* (*SPAA*), pages 281–291, June 2001.

[19] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proc*. *ACM Symp*. *on Parallel Algorithms and Architectures* (*SPAA*), pages 63–72, Aug. 2002.

[20] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Surfing wavelets on streams: one-pass summaries for approximate aggregate queries. In *Proc*. 27*th International Conf. on Very Large Data Bases* (*VLDB*), pages 79–88, Sept. 2001.

[21] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proc*. 33*rd ACM Symposium on Theory of Computing* (*STOC*), pages 471–475, July 2001.

[22] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc*. 41*st IEEE Symp*. *on Foundations of Computer Science* (*FOCS*), pages 359–366, Nov. 2000.

[23] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical report, Digital Systems Research Center, Palo Alto, CA, May 1998.

[24] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc*. 41*st IEEE Symp*. *on Foundations of Computer Science* (*FOCS*), Nov. 2000.

[25] P. Indyk. Personal communication, 2002.

[26] I. Kremer, N. Nisan, and D. Ron. On randomized one-round commmunication complexity. *Computational Complexity*, 8(1):21–49, 1999.

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

[28] S. Muthukrishnan. Data streams: algorithms and applications. Technical report, Rutgers University, Piscataway, NJ, 2003.

[29] I. Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39:67–71, 1991.

[30] I. Newman and M. Szegedy. Public vs. private coin flips in one round communication games. In *Proc*. 28*th ACM Symp*. *on the Theory of Computing* (*STOC*), pages 561–570, May 1996.

[31] L. Trevisan. A note on counting distinct elements in the streaming model. Unpublished manuscript, April 2001.

Received November 15, 2002, *and in revised form October* 5, 2003, *and in final form December* 1, 2003.
Online publication March 16, 2004.