Distributed Streams Algorithms for Sliding Windows

Phillip B. Gibbons, Srikanta Tirthapura

IRP-TR-02-08
August 2002

In Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures, Winnipeg, Manitoba, Canada, August 2002.

Intel Research Pittsburgh

Distributed Streams Algorithms for Sliding Windows

Phillip B. Gibbons
Intel Research Pittsburgh
417 South Craig Street
Pittsburgh, PA 15213
phillip.b.gibbons@intel.com

Srikanta Tirthapura
Computer Science Department
Brown University
Providence, RI 02912
snt@cs.brown.edu

ABSTRACT

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 over 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 over 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 over distributed streams that uses only logarithmic memory words.

Our results are obtained using a novel family of synopsis data structures which we call *waves*.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Applications; F.1.1 [**Theory of Computation**]: Models of Computation

General Terms

Algorithms, Theory

Keywords

Data streams, Sliding windows, Distributed, Waves

1. INTRODUCTION

There has been a flurry of recent work on designing effective algorithms for estimating aggregates and statistics over data streams [1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 25], due to their importance in applications such as network monitoring, data warehousing, telecommunications, and sensor networks. 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 either a 0 or a 1.

On the other hand, for many of these applications, there are multiple data sources, each generating its own 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 such scenarios, we previously proposed a *distributed streams* model [13], 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 (e.g., marketing, traffic engineering), 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. To model these scenarios, Datar et al. [4] recently introduced the *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 data stream.

This paper studies the sliding windows setting in both the single stream and distributed stream models, improving upon previous results under both settings. In order to describe our results, we first describe the models and the previous work in more detail.

1.1 Sequential and Distributed Streams

The goal in a (sequential or distributed) algorithm for data streams is to approximate a function F while minimizing (1) the total workspace (memory) used by all the parties, (2) the time taken by each party to process a 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 so attention is focused on finding either an (ϵ, δ)-approximation scheme or an ϵ-approximation scheme, defined as follows.

Definition 1. *An* (ϵ, δ)-approximation scheme *for a quantity* X *is a randomized procedure that, given any positive* $\epsilon < 1$ *and* $\delta < 1$*, computes an estimate* $\hat{X}$ *of* X *that is*

position	1	2	$\cdots$	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78
stream	0	1	$\cdots$	0	1	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	0
1-rank		1	$\cdots$		31					32	33		34	35	36	37	38	39	40	41	

position	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
stream	1	1	0	0	0	1	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1
1-rank	42	43				44	45	46			47		48			49					50

Figure 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.

within a relative error of ϵ with probability at least $1-\delta$, i.e., $\mathbf{Pr}\left\{|\hat{X} - X| \leq \epsilon X\right\} \geq 1 - \delta$. *An* ϵ-approximation scheme *is a deterministic procedure that, given any positive $\epsilon < 1$, computes an estimate whose worst case relative error is at most ϵ.*

Algorithms for a Sliding Window over a Single Stream. Datar et al. [4] presented a number of interesting results on estimating functions over a sliding window for a single stream. A fundamental problem they consider is that of determining the number of 1's in a sliding window, which they call the Basic Counting problem. In the stream in Figure 1, for example, the number of 1's in the current window of the 39 most recent items is 20. They presented an ϵ-approximation scheme for Basic Counting that uses only $O(\frac{1}{\epsilon}\log^2(\epsilon N))$ memory bits of workspace, 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 on the space. They demonstrated 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 upon their results by presenting an ϵ-approximation scheme for Basic Counting that matches their space and query time bounds, while improving the per-item processing time to $O(1)$ worst case time. We also present an ϵ-approximation scheme for the sum of bounded integers in a sliding window that improves the worst case per-item processing time from $O(\log N)$ to $O(1)$.

Our improved algorithms use a family of small space data structures, which we call *waves*. An example wave for Basic Counting is given in Figure 2, for the data stream in Figure 1. (The basic shape is suggestive of an ocean wave about to break.) The x-axis is the 1-rank, and extends to the right as new 1-bits arrive. 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.

Algorithms for Distributed Streams. In the distributed streams model [13], each party observes only its own stream, has limited workspace, and communicates with the other parties only when an estimate is requested. Specifically, to produce 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 [13] 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 only $O(\frac{1}{\epsilon^2}\log(1/\delta)\log n)$ 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 only $O(\frac{1}{\epsilon^2}\log(1/\delta)\log R)$ memory bits, where the values are in $[0..R]$. Both algorithms use *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.

In this paper, we combine the idea of a *wave* with *coordinated sampling*. We store a wave consisting of many random samples of the stream. Samples that contain only the recent items are sampled at a high probability, while those containing older items are sampled at a lower probability. We obtain an (ϵ,δ)-approximation scheme for the number of 1's in the position-wise union of distributed streams over a sliding window. We also obtain an (ϵ,δ)-approximation scheme for the number of distinct values in sliding windows over both single and distributed streams. Each scheme uses only logarithmic memory words per party.

The algorithms we propose are for the distributed streams model in the *stored coins* setting [13], where all parties 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 works on streaming models (e.g., [1, 5, 6, 8, 18, 19]) have studied settings with stored coins. Stored coins differ from *private coins* (e.g., as studied in communication complexity [21, 23, 24]) because the same random string can be stored at all parties.

1.2 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. We present the first ϵ-approximation scheme for Basic Counting over a single stream that is optimal in worst case space, processing time, and query time. Specifically, for a given accuracy ϵ, it matches the space bound and $O(1)$ query time of Datar et al. [4], while improving the per-item processing time from $O(1)$ amortized ($O(\log N)$ worst case) to $O(1)$ worst case.

3. We present the first ϵ-approximation scheme for the sum of integers in $[0..R]$ in a sliding window over a single stream that is optimal in worst case space (assuming R is at most polynomial in N), processing time,

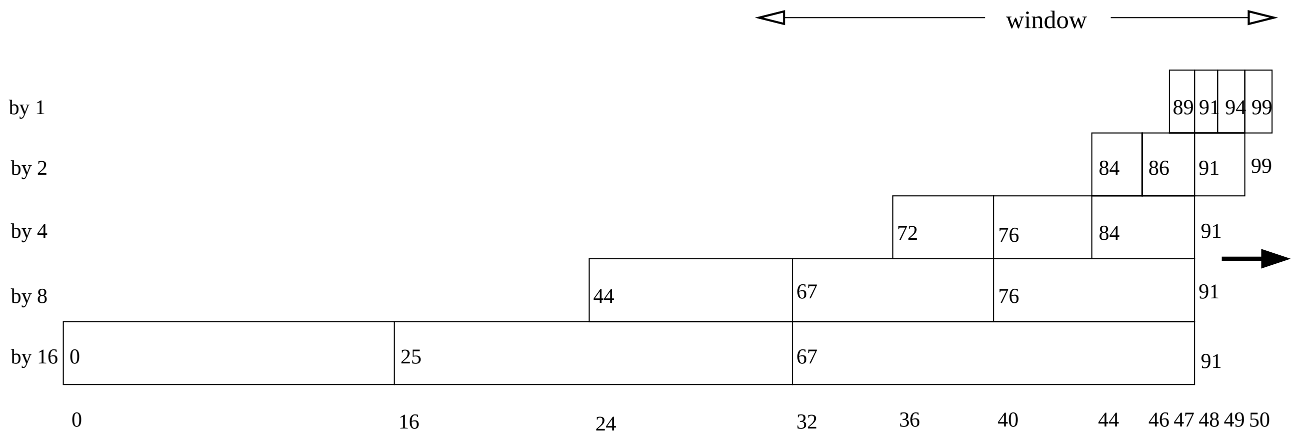

Figure 2: A deterministic wave and an example window query ($n = 39$). The x-axis shows the 1-ranks, and on the y-axis, level i is labeled as "by 2^i".

and query time. Specifically, it improves the per-item processing time of [4] from $O(1)$ amortized ($O(\log N)$ worst case) to $O(1)$ worst case.

4. We show that in contrast to the single stream case, no deterministic algorithm can estimate the number of 1's in a sliding window over the union of distributed streams within a small constant relative error unless it uses $\Omega(N)$ space.

5. We present the first randomized (ϵ, δ)-approximation scheme for the number of 1's in a sliding window over the union of distributed streams that uses only logarithmic memory words. We use this as a building block for the first (ϵ, δ)-approximation scheme for the number of distinct values in a sliding window over distributed streams that uses only logarithmic memory words.

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

2. RELATED WORK

In the paper introducing the sliding windows setting [4], the authors gave an algorithm for the Basic Counting problem that uses *exponential histograms*. An exponential histogram (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, the EH stores only its size (a power of 2) and the position of the most recent 1 in the bucket. Each k_i (up to the last bucket) is either $\frac{1}{2\epsilon}$ or $\frac{1}{2\epsilon}+1$. 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 $k_0 = \frac{1}{2\epsilon} + 2$, 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 shall see, our approach using waves avoids this cascading.

Our previous paper [13] formalized the distributed streams model and presented several (ϵ, δ)-approximation schemes for aggregates over distributed streams. We also compared the power of the distributed streams model with the previously studied *merged streams* model (e.g., [5, 19]), where all the data streams arrived at the same party, but interleaved in an arbitrary order.

The algorithm by Flajolet and Martin [7] and its variant due to Alon, Matias and Szegedy [1] estimate the number of distinct values in a stream (and also the number of 1's in a bit stream) up to a constant relative error of $\epsilon > 1$. The algorithm works in the distributed streams model too, and can be adapted to sliding windows [4]. There are two results we know of that extend this algorithm to work for arbitrary relative error, by Trevisan [25] and by Bar-Yossef et al. [3].[1] 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 $O(\frac{1}{\epsilon^2})$ 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 [4]. We do not know how to extend the algorithm in [3] 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* [3]).

We now quickly survey some other recent related work. Frameworks for studying data synopses were presented in [12], along with a survey of results. There have been algorithms for computing many different functions over a data stream observed by a single party, such as maintaining histograms [16], maintaining significant transforms of the data that are used to answer aggregate queries [14], and computing correlated aggregates [9]. Babcock et al. [2] considered the problem

[1]Datar et al. [4] also reported an extension to arbitrary relative error for a sliding window over a single stream, using the Trevisan approach [20].

of maintaining a uniform random sample of a specified size over a sliding window of the most recent elements.

In communication complexity models [22], the parties have *unlimited* time and space to process their respective inputs. 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.

None of these previous papers use wave-like synopses.

3. DETERMINISTIC WAVES

In this section, we will first present our new ϵ-approximation scheme for the number of 1's in a sliding window over a single stream. Then we will present our new ϵ-approximation scheme for the sum of bounded integers in a sliding window over a single stream. Finally, we will consider distributed streams, for three natural definitions of a sliding window over such streams. We will show that our small-space deterministic schemes can improve the performance for two of the scenarios, but for the third, no deterministic ϵ-approximation scheme can obtain sub-linear space.

3.1 The Basic Wave

We begin by describing the basic wave, and show how it yields an ϵ-approximation scheme for the Basic Counting problem for *any* sliding window up to a prespecified maximum window size N. The basic wave will be somewhat wasteful in terms of its space bound, processing time, and query time.

Consider a data stream of bits, and a desired positive $\epsilon < 1$. To simplify the notation, we will assume throughout that $\frac{1}{\epsilon}$ is an integer. We maintain two counters: pos, which is the current length of the stream, and rank, which is the current number of 1's in the stream (equivalently, the 1-rank of the most recent 1).

The wave contains the positions of the recent 1's in the stream, arranged at different "levels". The wave has $\ell = \lceil \log(2\epsilon N) \rceil$ levels. For $i = 0, 1, \ldots, \ell - 1$, level i contains the positions of the $1/\epsilon + 1$ most recent 1-bits whose 1-rank is a multiple of 2^i.[2] Figure 2 depicts the basic wave for the data stream in Figure 1, for $\epsilon = \frac{1}{3}$ and $N = 48$. In the figure, there are five levels, with level i labeled as "by 2^i" because it contains the positions of the $1/\epsilon + 1 = 4$ most recent 1-bits whose 1-ranks are 0 modulo 2^i. The 1-ranks are given on the x-axis.

Given this wave, we estimate the number of 1's in a window of size $n \le N$ as follows. Let $s = \max(0, \mathsf{pos} - n + 1)$; we are to estimate the number of 1's in stream positions $[s, \mathsf{pos}]$. The steps are:

1. Let p_1 be the *maximum position stored in the wave that is less than* s, and p_2 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.) Let r_1 and r_2 be the 1-ranks of p_1 and p_2 respectively.

2. Return $\hat{x} := \mathsf{rank} - r + 1$, where $r := r_2$ if $r_2 - r_1 = 1$ and otherwise $r := \frac{r_1 + r_2}{2}$.

For example, given the window query depicted in Figure 2, we have $n = 39$, $\mathsf{pos} = 99$, $\mathsf{rank} = 50$, $s = 61$, $p_1 = 44$, $p_2 = 67$, $r_1 = 24$, $r_2 = 32$, $r = 28$, and hence $\hat{x} = 23$. As noted earlier, the actual number of 1's in this window is 20, and indeed $\hat{x} \in [(1 - \frac{1}{\epsilon}) \cdot 20, (1 + \frac{1}{\epsilon}) \cdot 20] = [\frac{40}{3}, \frac{80}{3}]$.

LEMMA 1. *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. Each level i contains $(1/\epsilon + 1)$ 1's (stored with their positions in the stream) whose 1-ranks are 2^i apart. Thus, regardless of the current rank, the earliest 1-rank at level i is at most $\left(\mathsf{rank} - \frac{1}{\epsilon} \cdot 2^i\right)$. Thus, the difference between rank and the earliest 1-rank in level $\ell - 1$ is at least $2^{\ell-1}/\epsilon \ge N \ge n$. Since the difference in 1-ranks is at least as large as the difference in positions, it follows that p_1 exists. Let j be the smallest numbered level containing position p_1.

We know 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. Thus if $r_2 - r_1 = 1$, we return the exact answer. So assume $r = \frac{r_1 + r_2}{2}$ and $j > 0$. By returning the midpoint of the range, we guarantee that the absolute error is at most $\frac{r_2 - r_1}{2}$. By construction, there is at most a 2^j gap between r_1 and its next larger position r_2. Thus the absolute error in our estimate is at most 2^{j-1}. To bound the relative error, we will show that all the positions in level $j - 1$ are contained in the window, and this includes at least $\frac{2^{j-1}}{\epsilon}$ 1's. Let r_3 be the earliest 1-rank at level $j-1$. Position p_1 was not in level $j-1$, so $r_1 < r_3$. Since r_2 is the smallest 1-rank in the wave larger than r_1, we have $r_2 \le r_3$. Moreover, as argued above, $r_3 \le \mathsf{rank} - \frac{2^{j-1}}{\epsilon}$. Therefore, the actual number of 1's in the window is at least $\mathsf{rank} - r_2 + 1 \ge \mathsf{rank} - r_3 + 1 > \frac{2^{j-1}}{\epsilon}$. Thus the relative error is less than $\frac{1}{\epsilon}$.

Note that the proof readily extends beyond the steady state case: Any level with fewer than $\frac{1}{\epsilon} + 1$ positions will contain a position less than s, and hence can not serve the role of level $j - 1$ above. □

3.2 Improvements

We now show how to improve the basic wave in order to obtain 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) 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 $O(\log(\epsilon N))$ worst case time and $O(1)$ amortized time, as a new 1-bit is stored at each level i such that its 1-rank is a multiple of 2^i. Each of these improvements is used for the EH synopsis introduced by Datar et al. [4], to obtain the same bounds.

However, the deterministic wave synopsis is quite different from the EH synopsis, so the steps used are different too. Significantly, 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 will store each position only at its maximum level, as shown in Figure 3.[3] For levels

[2]To simplify the description, we describe throughout the steady state of a wave. Initially, there will be fewer than $1/\epsilon + 1$ such 1-bits and the wave stores all of them.

[3]In the figure, we have not explicitly discarded positions

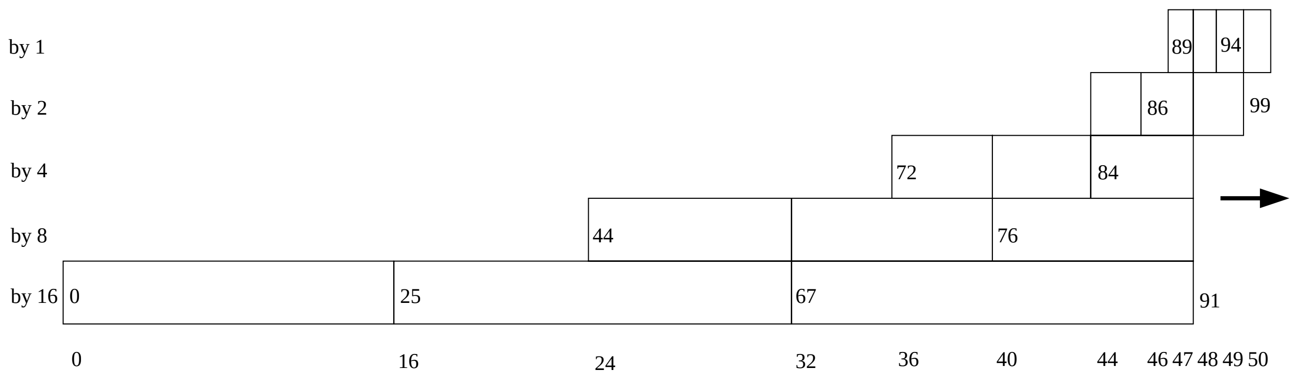

Figure 3: An optimal deterministic wave. The x-axis shows the 1-ranks, and on the y-axis, level i is labeled as "by 2^i".

$i = 0, \ldots, \ell - 2$, we store $\lceil \frac{1}{2}(\frac{1}{\epsilon} + 1) \rceil$ positions, and for level $\ell - 1$, we store $\lceil \frac{1}{\epsilon} + 1 \rceil$ positions. (At all levels, we may store fewer positions, because we discard expired positions.) In the wave, 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 (of the 1-bits) 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 [4], 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 $O(\frac{1}{\epsilon}\log(\epsilon N)\log N)$ bits to $O(\frac{1}{\epsilon}\log^2(\epsilon N))$ bits.

Figure 4 summarizes the steps of the deterministic wave algorithm. Putting it altogether, we have:

THEOREM 1. *The algorithm in Figure 4 is an ϵ-approximation scheme for the number of 1's in a sliding window of size N over a data stream, using $O(\frac{1}{\epsilon}\log^2(\epsilon N))$ bits. Each stream item is processed in $O(1)$ worst case time. At each time instant, it can provide an estimate in $O(1)$ time.*

PROOF. (sketch) The proof of ϵ relative error follows along the lines of the proof of Lemma 1, because the set of positions in the improved wave is the same or a superset of the set of positions in the basic wave. 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.[4] As for the space, because the level queues are updated in place, the same block of memory can be used throughout, and hence the linked list pointers are offsets into this block and not full-sized pointers. The space bound follows. $\square$

The space bound is optimal because it matches the lower bound by Datar et al. [4] for both randomized and deterministic algorithms.

Upon receiving a stream bit b:

1. Increment pos. (Note: All additions and comparisons are done modulo N', the smallest power of 2 greater than or equal to $2N$.)
2. If the head (p, r) of the linked list L has expired (i.e., $p \leq \text{pos} - N$), 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 wave 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 for a sliding window of size N:

1. Let r_1 be the largest 1-rank discarded. (If no such r_1, return $\hat{x} :=$ rank as the exact answer.) Let r_2 be the 1-rank at the head of the linked list L. (If L is empty, return $\hat{x} := 0$ as the exact answer.)
2. Return $\hat{x} :=$ rank $- r + 1$, where $r := r_2$ if $r_2 - r_1 = 1$ and otherwise $r := \frac{r_1 + r_2}{2}$.

Figure 4: A deterministic wave algorithm for Basic Counting over a single stream.

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. 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. But this takes $\Theta(\log N)$ worst case time, because rank has N' bits. Instead, we store the $\log N'$ wave levels 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 only $O(\log N \log\log N)$ bits. We also store a counter d of $\log N' - \log\log N'$

outside the size $N = 48$ window, in order to show the full levels. All positions less than pos $- N = 51$ have expired, and $r_1 = 24$ is the largest expired 1-rank.

[4]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.

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 wave 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. 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 levels in $O(1)$ worst case time.

Basic Counting for Any Window of Size $n \le N$. The algorithm in Figure 4 achieves constant worst case query time for a sliding window of size N. For a sliding window of *any* size $n \le N$, 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). However, the query time for window sizes less than N is $O(\frac{1}{\epsilon}\log(\epsilon N))$ 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 [4].

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. [4] showed how to extend their EH approach to obtain an ϵ-approximation scheme for this problem, using $O(\frac{1}{\epsilon}(\log N + \log R))$ buckets of $\log N + \log(\log N + \log R)$ bits each, $O(1)$ query time, and $O(\frac{\log R}{\log N})$ 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, under certain weak assumptions on the relative sizes of N, R, and ϵ.) 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. (The number of bits is $O(\frac{1}{\epsilon}(\log N + \log R)^2)$, which is slightly worse than the EH bound if R is super-polynomial in N.)

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 $\ell = \lceil \log(2\epsilon NR) \rceil$ levels. 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 $[\mathsf{total} - z_2 + v_2, \mathsf{total} - z_1]$, 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. But this would lead to $O(R)$ worst case processing time per item. Datar et al. [4] reduced the time to $O(\log N + \log R)$ by directly computing the EH resulting from v insertions of items 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.

Upon receiving a stream value $v \in [0..R]$:

1. Increment pos. (Note: All additions and comparisons are done modulo N'.)
2. If the head (p, v', z) of the linked list L has expired (i.e., $p \le \mathsf{pos} - N$), 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 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 for a sliding window of size N:

1. Let z_1 be the largest partial sum discarded from L. (If no such z_1, return $\hat{x} := \mathsf{total}$ as the exact answer.) Let (p, v_2, z_2) be the head of the linked list L. (If L is empty, return $\hat{x} := 0$ as the exact answer.)
2. Return $\hat{x} := \mathsf{total} - \frac{z_1 + z_2 - v_2}{2}$

Figure 5: A deterministic wave algorithm for the sum over a sliding window.

The challenge is to quickly compute the wave level in step 3(a); 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 positions less than j (and hence y 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., $\lfloor \log h \rfloor$.[5]

Putting it altogether, we have:

THEOREM 2. *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 $O(\frac{1}{\epsilon}(\log N + \log R))$ 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.*

[5]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 B be a bit mask comprising of $\frac{w}{2}$ 1's followed by $\frac{w}{2}$ 0's. We begin by checking if $h \wedge B$ equals zero. If so, we left shift B by $\frac{w}{4}$ positions and recurse. Otherwise, we right shift B by $\frac{w}{4}$ positions and recurse.

Proof. (sketch) For the purposes of analyzing the approximation error, we reduce the wave to an equivalent basic wave for the Basic Counting problem, as follows. For each triple (p, v, z) in the sums wave, we have a pair (p, z') in the basic wave for each $z' \in [z - v + 1, z]$, stored in all levels l such that z' is a multiple of 2^l. Also add the pair (p_1, z_1) where z_1 is the largest partial sum discarded by the sums wave algorithm, to all levels l' such that z' is a multiple of $2^{l'}$. Next, for each level, discard all but the most recent $\frac{1}{\epsilon} + 1$ at the level. Let $\mathsf{rank} = \mathsf{total}$. Let $r_1 = z_1$, and let j be the minimum level containing p_1. Adapting the argument in the proof of Lemma 1, it can be shown that (1) regardless of the current rank, the earliest 1-rank at level i is at most $\mathsf{rank} - \frac{1}{\epsilon} \cdot 2^i$, (2) there is at most a 2^j gap between r_1 and its next larger position r_2, and (3) all the positions in level $j - 1$ are contained in the window.

We know that the window sum is in $[\mathsf{total} - z_2 + v_2, \mathsf{total} - z_1]$, and since we take the midpoint, the absolute error of $\hat{x}$ is at most $\frac{z_2 - v_2 - z_1}{2}$. The gap between $z_2 - v_2$ and z_1 is at most the gap between r_1 and r_2 in the basic wave. Thus by (2) above, the absolute error is at most 2^{j-1}. Moreover, by (1) and (3) above, the actual window sum is at least $\mathsf{rank} - r_2 + 1 > \frac{2^{j-1}}{\epsilon}$. Thus the relative error is less than $\frac{1}{\epsilon}$.

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

3.4 Distributed Streams

We consider three natural definitions for a sliding window over a collection of $t \geq 2$ distributed streams, as illustrated for the Basic Counting problem:

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

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

3. We seek the total number of 1's in the last N items in the position-wise union (logical OR) of the t streams.

The deterministic wave can be used to answer sliding windows queries over a collection of distributed streams, for both the first two scenarios. 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. Because each individual count is within ϵ relative error, so is the total. The second scenario can similarly be reduced to the single stream problem. The only issue is that each party knows only the latest sequence number in its stream, not the overall latest, so some waves may contain expired positions. Thus to answer a query, each party sends its wave to the Referee, who computes the maximum sequence number over all the parties and then uses each wave to obtain an estimate over the resulting window, and sums the result. Because each individual estimate is within an ϵ relative error (recall the discussion at the end of Section 3.2), so is the total. By improving the single stream performance over the previous work, we have improved the distributed streams performance for these two scenarios.

However, the third scenario is more problematic. Denote as the Union Counting problem the problem of counting the number of 1's in the position-wise 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 present next a linear space lower bound for deterministic algorithms for this problem, before considering randomized algorithms in Section 4.

A Lower Bound for Deterministic Algorithms. We show the following lower bound on *any* deterministic algorithm for the Union Counting problem that guarantees a small constant relative error.

Theorem 3. *Any deterministic algorithm that guarantees a constant relative error $\epsilon \leq \frac{1}{64}$ 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 within a relative error of $\epsilon = \frac{1}{64}$ 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, where A sees the data stream X and B sees the data stream Y. 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 $\frac{n}{2}$ ones and zeroes. Note that for this restricted scenario, the exact answer for the Union Counting problem is

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

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

For each possible message m from A to the Referee C, let S_m denote the set of all inputs to A for which A sends m to C. Since A's workspace is only αn bits, the number of distinct messages that A could send to C is $2^{\alpha n}$. The number of possible inputs for 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

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

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 $n\epsilon$. We claim that no two inputs in S_m can be at a Hamming distance greater than $4n\epsilon$. The proof is by contradiction. Suppose there are two inputs X_1 and X_2 in S_m such that $H(X_1, X_2) > 4n\epsilon$. 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 so it outputs the same estimate z. Because the absolute error in both cases is at most $n\epsilon$, we have by equation (1) that $z \geq \frac{n}{2} + \frac{1}{2}H(X_1, X_2) - n\epsilon > \frac{n}{2} + n\epsilon$ and $z \leq \frac{n}{2} + \frac{1}{2}H(X_2, X_2) + n\epsilon = \frac{n}{2} + n\epsilon$, a contradiction.

For a given n-bit input t with exactly $\frac{n}{2}$ 1's, the number of n-bit inputs with exactly $\frac{n}{2}$ 1's at a Hamming distance of k from t (k an even number) is $\binom{n/2}{k/2}^2$ — all combinations of $\frac{k}{2}$ out of $\frac{n}{2}$ 0's in t flipped to 1's and $\frac{k}{2}$ out of $\frac{n}{2}$ 1's in t flipped to 0's. (There are no such inputs at odd distances.) Thus the number of such inputs at Hamming distance at most k

is $\sum_{j=0}^{k/2}\binom{n/2}{j}^2$, which, for $k \leq \frac{n}{4}$, is at most $(1+\frac{k}{2})\binom{n/2}{k/2}^2$. By the above claim, for all messages m that A sends to C, we have:

$$|S_m| \leq (1+2n\epsilon)\binom{n/2}{2n\epsilon}^2 \tag{3}$$

By choosing $\alpha = \frac{1}{16}$ in equation (2), we have that

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

By choosing $\epsilon = 1/64$ and n suitably large, it follows from equation (3) that

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

We obtain the contradiction, which completes the proof. □

Sum of Bounded Integers. For the sum of bounded integers problem, scenarios 1 and 2 are straightforward applications of the single stream algorithm. For scenario 3, if "union" means to take the position-wise sum, the problem reduces to the first scenario. If "union" means to take the position-wise maximum, then the lower bound applies, as the number of 1's in the union is a special case of the sum of the position-wise maximum.

The linear space lower bound for deterministic algorithms in Theorem 3 is the motivation for considering the *randomized* waves introduced in the next section.

4. RANDOMIZED WAVES

Similar to the deterministic wave, the randomized wave contains the positions of the recent 1's in the data stream, stored at different levels. Each level i contains the most recently selected positions of the 1-bits, where a position is selected into level i with probability 2^{-i}. Thus the main difference between the deterministic and randomized waves is that for each level i, the deterministic wave selects 1 out of every 2^i 1-bits at *regular* intervals, whereas a randomized wave selects an *expected* 1 out of every 2^i 1-bits at *random* intervals. Also, the randomize wave retains more positions per level.

4.1 The Basic Randomized Wave

We begin by describing the basic randomized wave, and show how it yields an (ϵ, δ)-approximation scheme for Union Counting in *any* sliding window up to a prespecified maximum window size N. We then sketch the proof of the approximation guarantees, which uses the main error analysis lemma from [13]. Finally, we show the time and space bounds.

Let N' be the minimum power of 2 that is at least $2N$; let $d = \log N'$. Let $\epsilon < 1$ be the desired error probability. Each party P_j maintains a basic randomized wave for its stream, consisting of $d+1$ queues, $Q_j(0), \ldots, Q_j(d)$, one for each level $l = 0 \ldots d$.

We use a pseudo-random hash function h to map positions to levels, according to an exponential distribution. For $0 \leq l \leq (d-1)$, $\mathbf{Pr}\{h(p) = l\} = 1/2^{l+1}$. $\mathbf{Pr}\{h(p) = d\} = 1/2^d$. $h(\cdot)$ is computed as follows: we consider the numbers $\{0 \ldots N'-1\}$ 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 $x =$

```
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., $p \leq \text{pos} - N$).
  3. If $b = 1$ then for $l := 0, \ldots, h(\text{pos})$ 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 $n \leq N$, after each party has observed pos bits:
  1. Each party $j$ sends its wave, $\{Q_j(0), \ldots, Q_j(\log N')\}$, to the Referee. Let $s := \max(0, \text{pos} - n + 1)$. Then $W = [s, \text{pos}]$ is the desired window.
  2. For $j := 1, \ldots, t$, let $l_j$ be the minimum level such that the tail of $Q_j(l_j)$ is a position $p \leq s$.
  3. Let $l^* := \max_{j=1,\ldots,t} l_j$. Let $U$ be the union of all positions in $Q_1(l^*), \ldots, Q_t(l^*)$.
  4. Return $\hat{x} := 2^{l^*} \cdot |U \cap W|$.
```

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

$q \cdot p + r$ (all operations being performed in G). We represent x as a d-bit vector and then $h(p)$ is the largest y such that the y most significant bits of x are zero (i.e., $y = d - \lfloor \log x \rfloor - 1$). Clearly, $h(p) \in [0..d]$. The two properties of h that we use are: (1) x is distributed uniformly over G. Hence the probability that $h(i)$ equals l (where $l < d$) is exactly $1/2^{l+1}$. (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 we used in [13], except that the domain and range sizes now depend only on the maximum window size N and not the entire stream length.

The steps for maintaining the randomized wave are summarized in the top half of Figure 6. A 1-bit arriving in 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; we have not attempted to minimize c.) 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 $[i, \text{pos}]$ whose positions hash to a value greater than or equal to l. We call this the *range* of $Q_j(l)$. As we move from level l to $l+1$, the range may increase, but it will never decrease. For any window of size at most N, the queues at lower numbered levels may have ranges that fail to contain the window, but as we move to higher levels, we will (with high probability) find a level whose range contains the window.

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 $W = [\max(0, \text{pos} - n + 1), \text{pos}]$. Each party P_j initially selects the lowest numbered level l_j such that the range of $Q_j(l_j)$ contains W (step 2). Let l^* be the maximum of these levels over all the parties. Thus at each party P_j, the range of $Q_j(l^*)$ contains W. This implies that each queue contains the positions of *all* the 1-bits in W in its stream that hash to a value at least l^*. We take the union of the positions in the t queues, to form the queue for level l^* of the position-wise OR of the streams (step 3). The algorithm returns the number of positions in this queue that fall within the window W, scaled up by a factor of 2^{l^*} (step 4).

LEMMA 2. *The algorithm in Figure 6 returns an estimate for the Union Counting problem for any sliding window of size $n \leq N$ that is within a relative error of ϵ with probability greater than $2/3$.*

PROOF. For each level l, define $S_j(l) = \{i|(l \leq h(i)) \wedge (b_i = 1 \text{ in stream } j)\}$. $Q_j(l)$ maintains the positions of the c/ϵ^2 most recent 1's in $S_j(l)$. Consider the size of the overlap of $S_j(l)$ and W. This is large for small l (because the probability of selecting a 1-bit in W for $S_j(l)$ is $1/2^l$), and decreases as l increases. If the overlap at level l is greater than c/ϵ^2, then W contains a position not among the c/ϵ^2 most recent positions of $S_j(l)$. On the other hand, if the overlap is less than or equal to c/ϵ^2, then the range of $Q_j(l)$ contains W. Thus, l_j (the level selected by P_j) is the minimum level such that $|S_j(l) \cap W| \leq c/\epsilon^2$.

In other words, we are progressively halving the sampling probability until we are at a level where the number of points in the overlap is less than or equal to c/ϵ^2. This very random process has been analyzed in our previous paper [13] (though in a different scenario). Thus, the lemma follows from Lemma 1 in [13]. □

By taking the median of $O(\log(1/\delta))$ independent instances of the algorithm, we get our desired (ϵ, δ)-approximation scheme:

THEOREM 4. *The above estimation procedure is an (ϵ, δ)-approximation scheme for the Union Counting problem for any sliding window of size at most N, using $O(\frac{\log(1/\delta)\log^2 N}{\epsilon^2})$ bits per party. The time to process an item is dominated by the time for an expected $O(\log(1/\delta))$ finite field operations.*

PROOF. For each of the $O(\log(1/\delta))$ instances, we have $O(\log N)$ queues of $O(1/\epsilon^2)$ 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. Note that the approximation guarantees hold regardless of the number of parties.

The per-item processing is $O(1)$ expected time per instance because the expected number of levels to which each new position is added (step 3) is bounded by 2, and likewise the expected number of levels that position $p = \mathsf{pos} - N$ was ever in is bounded by 2. Thus scanning the tails of the queues at levels $0, \ldots, h(p)$ looking for p (step 2) takes constant expected time. □

4.2 Improvements in Query Time

The query time for the above estimation procedure is the time for the Referee to receive and process $O(t\frac{\log(1/\delta)\log N}{\epsilon^2})$ memory words. If all queries were for window size N, each party P_j could easily keep track of the minimum level l_j at which the range of $Q_j(l_j)$ contains the window, with constant processing time overhead. When a query is requested, P_j sends only l_j and $Q_j(l_j)$. After determining l^*, the Referee retains only those positions p in the queues that both fall within the window and have $h(p) \geq l^*$. (To avoid recomputing $h(p)$, $h(p)$ could be stored in all queues containing p.) In this way, the Referee computes $Q_j(l^*) \cap W$ for each j without ever explicitly receiving $Q_j(l^*)$ from party P_j. It takes the union of these retained positions and returns the estimate $\hat{x}$ as before. This reduces the query time to $O(t/\epsilon^2)$ time per instance, while preserving the other bounds.

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 distributed streams. An item selected for a level's sample is now stored as 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 sample at level l stores the $\frac{c}{\epsilon^2}$ pairs with the most recent positions that were sampled into that level. Note that, in contrast to the Union Counting scheme, the hash function now hashes the *value* of the item, rather than its position.

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

When an item v arrives, we insert (pos, v) into levels $0, \ldots, h(v)$. 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 value in the doubly linked list. Updating a value's position requires moving its corresponding pair from its current position to the tail of the list, and this can be done in constant time. The value's position has to be updated in each of the levels to which it belongs. A straightforward argument shows that all this per-item processing can be done in constant expected time, because each value 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 level-wise 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 approximation guarantees follow directly from the arguments in the previous section. Putting it altogether, we have:

THEOREM 5. *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 $O(\frac{\log(1/\delta)\log N \log R}{\epsilon^2})$ bits per party, where 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 operations.*

Handling Predicates. Note that our algorithm for distinct values counting stores a random sample of the distinct values. This sample can be used to answer more complex queries on the set of distinct values (e.g., how many *even* distinct values are there?), where the predicate ("evenness") is not known until query time. In order to provide an (ϵ, δ)-approximation scheme for *any* such ad hoc predicate that has selectivity at least α (i.e., at least an α fraction of the distinct values satisfy the predicate), we store a sample of size $O(\frac{1}{\alpha\epsilon^2})$ at each level, increasing our space bound by a factor of $\frac{1}{\alpha}$. Such problems without sliding windows were studied in [10].

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 $O(\frac{1}{\epsilon}\log^2(\epsilon m))$ bits, where m is an upper bound on the window size needed in order to contain the N most recent 1's.

Other Problems. Our improved time bounds for Basic Counting and for Sum over a single stream lead to improved time bounds for all problems which reduce to these problems, as described in [4]. 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 $\frac{\epsilon}{2+\epsilon}$).

6. REFERENCES

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

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

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

[4] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pages 635–644, Jan. 2002.

[5] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1-difference algorithm for massive data streams. In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, pages 501–511, Oct. 1999.

[6] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms*, pages 165–174, Jan. 2000.

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

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

[9] 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*, pages 13–24, May 2001.

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

[11] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 331–342, June 1998.

[12] 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. AMS, 1999. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 50. A two page summary appeared as a short paper in SODA'99.

[13] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architectures*, pages 281–290, July 2001.

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

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

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

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

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

[19] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, pages 189–197, Nov. 2000.

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

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

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

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

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

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