Approximate Aggregation Techniques for Sensor Databases

Jeffrey Considine, Feifei Li, George Kollios, and John Byers
Computer Science Dept., Boston University
{jconsidi, lifeifei, gkollios, byers}@cs.bu.edu

Abstract

In the emerging area of sensor-based systems, a significant challenge is to develop scalable, fault-tolerant methods to extract useful information from the data the sensors collect. An approach to this data management problem is the use of sensor database systems, exemplified by TinyDB and Cougar, which allow users to perform aggregation queries such as MIN, COUNT and AVG on a sensor network. Due to power and range constraints, centralized approaches are generally impractical, so most systems use in-network aggregation to reduce network traffic. However, these aggregation strategies become bandwidth-intensive when combined with the fault-tolerant, multi-path routing methods often used in these environments. For example, duplicate-sensitive aggregates such as SUM cannot be computed exactly using substantially less bandwidth than explicit enumeration. To avoid this expense, we investigate the use of approximate in-network aggregation using small sketches. Our contributions are as follows: 1) we generalize well known duplicate-insensitive sketches for approximating COUNT to handle SUM, 2) we present and analyze methods for using sketches to produce accurate results with low communication and computation overhead, and 3) we present an extensive experimental validation of our methods.

1. Introduction

As computation-enabled devices shrink in scale and proliferate in quantity, a relatively recent research direction has emerged to contemplate future applications of these devices and services to support them. A canonical example of such a device is a *sensor mote*, a device with measurement, communication, and computation capabilities, powered by a small battery [12]. Individually, these motes have limited capabilities, but when a large number of them are networked together into a *sensor network*, they become much more capable. Indeed, large-scale sensor networks are now being applied experimentally in a wide variety of areas — some sample applications include environmental monitoring, surveillance, and traffic monitoring.

In a typical sensor network, each sensor produces a stream of sensory observations across one or more sensing modalities. But for many applications and sensing modalities, such as reporting temperature readings, it is unnecessary for each sensor to report its entire data stream in full fidelity. Moreover, in a resource-constrained sensor network environment, each message transmission is a significant, energy-expending operation. For this reason, and because individual readings may be noisy or unavailable, it is natural to use data aggregation to summarize information collected by sensors. As a reflection of this, a database approach to managing data collected on sensor networks has been advocated [15, 20], with particular attention paid to efficient query processing for aggregate queries [15, 20, 23].

In the TAG system [15], users connect to the sensor network using a workstation or base station directly connected to a sensor designated as the sink. Aggregate queries over the sensor data are formulated using a simple SQL-like language, then distributed across the network. Aggregate results are sent back to the workstation over a spanning tree, with each sensor combining its own data with results received from its children. If there are no failures, this in-network aggregation technique is both effective and energy-efficient for distributive and algebraic aggregates [11] such as MIN, MAX, COUNT and AVG. However, as we will argue, this technique is much less effective in sensor network scenarios with moderate node and link failure rates. Node failure is inevitable when inexpensive, faulty components are placed in a a variety of uncontrolled or even hostile environments. Similarly, link failures and packet losses are common across wireless channels because of environmental interference, packet collisions, and low signal-to-noise ratios [23].

When a spanning tree approach is used for aggregate queries, as in TAG, a single failure results in an entire subtree of values being lost. If this failure is close to the sink, the change in the resulting aggregate can be significant. Retransmission-based approaches are expensive in this environment, so solutions based upon multi-path routing were proposed in [15]. For aggregates such as MIN and MAX which are monotonic and exemplary, this provides a fault-tolerant solution. But for duplicate-sensitive aggregates such as COUNT or AVG, that give incorrect results when the same value is counted multiple times, existing methods are not satisfactory.

In this paper, we propose a robust and scalable method for computing duplicate-sensitive aggregates across faulty sensor networks. Guaranteeing exact solutions in the face of losses is generally impractical, so we instead consider approximate methods. These methods are robust against both link and node failures. Our contributions can be summarized as follows:

- We extend well-known duplicate insensitive sketches [7] to handle SUM aggregates. Through analysis and experiments, we show that the new sketches provide accurate approximations.
- We present a method to combine duplicate insensitive sketches with multi-path routing techniques to produce accurate approximations with low communication and computation overhead.
- We provide an analysis of the expected performance of previous methods as well as our method.
- Finally, we present an extensive experimental evaluation of our proposed system which we compare with previous approaches.

Concurrent with our work, Nath and Gibbons [17] independently studied the use of duplicate-insensitive sketches for aggregation in sensor networks. Their work focused more on the logical decoupling of routing and aggregation, along with the necessary sketch properties for correctness using multipath routing, namely having an associative, commutative, and idempotent aggregation operation.

The remainder of this paper proceeds as follows. Background material is covered in Section 2. Counting sketches, along with theory and new generalizations, are discussed in Section 3. A robust aggregation framework using these sketches is then presented in Section 4. We validate our methods experimentally in Section 5 and conclude in Section 6.

2. Background

We now briefly survey the related work of our methods. Sensors and their limitations are described in Section 2.1. Previous frameworks for processing aggregates are covered in 2.2, and multipath routing techniques are covered in 2.3. Finally, the sketches which we use to improve upon these frameworks are introduced in Section 2.4.

2.1. Sensor Devices

Today's sensor motes (e.g. [12]) are full fledged computer systems, with a CPU, main memory, operating system and a suite of sensors. They are powered by small batteries and their lifetime is primarily dependent on the extent to which battery power is conserved. The power consumption tends to be dominated by transmitting and receiving messages and most systems try to minimize the number of messages in order to save power. Also, the communication between sensors is wireless and the packet loss rate between nodes can be high. For example, [23] reports on experiments in which more than 10% of the links suffered average loss rate greater than 50%. Another challenge is that links may be asymmetric, both in loss rates and even reachability. These limitations motivate query evaluation methods in sensor networks that are fundamentally different from the traditional distributed query evaluation approaches. First, the query execution plan must be energy efficient and second, the process must be as robust as possible given the communication limitations in these networks.

2.2. In-network Aggregate Query Processing

A simple approach to evaluate an aggregation query is to route all sensed values to the base station and compute the aggregate there. Although this approach is simple, the number of messages and the power consumption can be large. A better approach is to leverage the computational power of the sensor devices and compute aggregates in-network. Aggregates that can be computed in-network include all decomposable functions [15].

Definition 1 *A function f is decomposable, if it can be computed by another function g as follows:* $f(v_1, v_2, ..., v_n) = g(f(v_1, ..., v_k), f(v_{k+1}, ..., v_n))$.

Using decomposable functions, the value of the aggregate function can be computed for disjoint subsets, and these values can be used to compute the aggregate of the whole using the merging function g. Our discussion is based on the Tiny Aggregation (TAG) framework used in TinyDB [15]. However, similar approaches are used to compute aggregates in other systems [20, 21, 23, 13].

In TAG, the in-network query evaluation has two phases, the *distribution* phase and the *collection* phase. During the distribution phase, the query is flooded in the network and organizes the nodes into an *aggregation tree*. The base station broadcasting the query is the *root* of the tree. The query message has a counter that is incremented with each retransmission and counts the hop distance from the root. In this way, each node is assigned to a specific level equal to the node's hop distance from the root. Also, each sensor chooses one of its neighbors with a smaller hop distance from the root to be its parent in the aggregation tree.

During the collection phase, each leaf node produces a single tuple and forwards this tuple to its parent. The non-leaf nodes receive the tuples of their children and combine these values. Then, they submit the new partial results to their own parents. This process runs continuously and after h steps, where h is the height of the aggregation tree, the total result will arrive at the root. In order to conserve energy, sensor nodes sleep as much as possible during each step where the processor and radio are idle. When a timer expires or an external event occurs, the device wakes and starts the processing and communication phases. At this point, it receives the messages from its children and then submits the new value(s) to its parent. After that, if no more processing is needed for that step, it enters again into the sleeping mode [16].

As mentioned earlier, this approach works very well for ideal network conditions, but is less satisfactory under lossy conditions. To address these issues, Madden at al. [15] proposed various methods to improve the performance of their system. One solution is to cache previous values and reuse them if newer ones are unavailable. Of course, these cached values may reflect losses at lower levels of the tree.

Another approach considered in [15] takes advantage of the fact that a node may select multiple parents from neighbors at a higher level. Using this approach, which we refer to as "fractional parents," the aggregate value is decomposed into fractions equal to the number of parents. Each fraction is then sent to a distinct parent instead of sending the whole value to a single parent. For example, given an aggregate sum of 15 and 2 parents, each parent would be sent the value 7.5. It is easy to demonstrate analytically that this approach does not improve the expected value of the estimate over the single parent approach; it only helps to reduce the variance of the estimated value at the root. Therefore, the problem of losing a significant fraction of the aggregate value due to network failures remains.

2.3. Best Effort Routing in Sensor Networks

Recent years have seen significant work on best-effort routing routing in sensor and other wireless networks. Due to high loss rates and power constraints, a common approach is to use multi-path routing, where more than one copy of a packet is sent to the destination over different paths. For example, directed diffusion [13] uses a flood to discover short paths which sensors would use to send back responses. Various positive and negative reinforment mechanisms are used to improve path quality. Braided diffusion [8] builds on directed diffusion to use a set of intertwined paths for increased resilience. A slightly different approach is used by GRAB [22], where paths are not explicitly chosen, but the width of the upstream broadcast is controlled.

Our techniques are meant to complement and leverage any of these routing techniques. We note that combining these methods with duplicate-insensitive in-network aggregation will allow some of the overhead of these techniques to be amortized and shared amongst data items from many different sensors.

2.4. Counting Sketches

Counting sketches were introduced by Flajolet and Martin in [7] for the purpose of quickly estimating the number of distinct items in a database (or stream) in one pass while using only a small amount of space. Since then, there has been much work developing and generalizing counting sketches (e.g. [1, 6, 10, 3, 9, 2]).

It is well known that exact solutions to the distinct counting problem require $\Omega(n)$ space. As shown in [1], $\Theta(\log n)$ space is required to approximate the number of distinct items in a multi-set with n distinct items. The original FM sketches of [7] achieve this bound, though they assume a fixed hash function that appears random, so they are vulnerable to adversarial choices of inputs. We use these sketches since they are very small and accurate in practice, and describe them in detail in Section 3.

A different sketching scheme using linear hash functions was proposed in [1]. These sketches are somewhat larger than FM sketches in practice, although a very recent technique [5] extending these methods uses only $O(\log \log n)$ space. We intend to investigate the effectiveness of these "loglog" sketches for sensor databases in future work.

3. Sketch Theory

One of the core ideas behind our work is that duplicate insensitive sketches will allow us to leverage the robustness typically associated with multi-path routing. We now present some of the theory behind such sketches and extend it to handle more interesting aggregates. First, we present details of the FM sketches of [7] along with necessary parts of the theory behind them. Then, we generalize these sketches to handle summations, and show that they have almost exactly the same accuracy as FM sketches.

3.1. Counting Sketches

We now describe FM sketches for the distinct counting problem.

Definition 2 *Given a multi-set of items $M = \{x_1, x_2, x_3, \dots\}$, the* distinct counting *problem is to compute* $n \equiv |\text{distinct}(M)|$.

Given a multi-set M, the FM sketch of M, denoted $S(M)$, is a bitmap of length k. The entries of $S(M)$, denoted $S(M)[0, \dots, k-1]$, are initialized to zero and are set to one using a random binary hash function h applied to the elements of M. Formally,

$$S(M)[i] \equiv 1 \text{ iff } \exists x \in M \text{ s.t. } \min\{j \mid h(x,j) = 1\} = i.$$

By this definition, each item x is capable of setting a single bit in $S(M)$ to one – the minimum i for which $h(x,i) = 1$. This gives a simple serial implementation which is very fast in practice and requires two invocations of h per item on average.

Theorem 1 *An element x_i can be inserted into an FM sketch in $O(1)$ expected time.*

Algorithm 1 COUNTINSERT(S,x)

```
1: i = 0;
2: while hash(x,i) = 0 do
3:     i = i + 1;
4: end while
5: S[i] = 1;
```

We now describe some interesting properties of the sketches observed in [7].

Property 1 *The FM sketch of the union of two multi-sets is the bit-wise OR of their FM sketches. That is, $S(M_1 \cup M_2)[i] = (S(M_1)[i] \vee S(M_2)[i])$.*

Property 2 *$S(M)$ is entirely determined by the distinct items of M. Duplication and ordering do not affect $S(M)$.*

Property 1 allows each node to compute a sketch of locally held items and send the small sketch for aggregation elsewhere. Since aggregation via union operations is cheap, it may be performed in the network without significant computational burden. Property 2 allows the use of multi-path routing of the sketches for robustness without affecting the accuracy of the estimates. We expand upon these ideas in Section 4.

The next lemma provides key insight into the behavior of FM sketches and will be the basis of efficient implementations of summation sketches later.

Lemma 1 *For $i < \log_2 n - 2\log_2\log_2 n$, $S(M)[i] = 1$ with probability $1 - O(ne^{-\log_2^2 n})$. For $i \geq \frac{3}{2}\log_2 n + \delta$, with $\delta \geq 0$, $S(M)[i] = 0$ with probability $1 - O\left(\frac{2^{-\delta}}{\sqrt{n}}\right)$.*

Proof: This lemma is proven in [7] and follows from basic balls and bins arguments. ■

The lemma implies that given an FM sketch of n distinct items, one expects an initial prefix of all ones and a suffix of all zeros, while only the setting of the bits around $S(M)[\log_2 n]$ exhibit much variation. This gives a bound on the number of bits k required for $S(M)$ in general: $k = \frac{3}{2}\log_2 n$ bits suffice to represent $S(M)$ with high probability. It also suggests that just considering the length of the prefix of all ones in this sketch can produce an estimate of n. Formally, let $R_n \equiv \min\{i \mid S(M)[i] = 0\}$ when $S(M)$ is an FM sketch of n distinct items. That is, R_n is a random variable marking the location of the first zero in $S(M)$. In [7], a method to use R_n as an estimator for n is developed using the following theorems.

Theorem 2 *The expected value of R_n for FM sketches satisfies $E(R_n) = \log_2(\varphi n) + P(\log_2 n) + o(1)$, where the constant φ is approximately 0.775351 and $P(u)$ is a periodic and continuous function of u with period 1 and amplitude bounded by 10^{-5}.*

Theorem 3 *The variance of R_n for FM sketches, denoted σ_n^2, satisfies $\sigma_n^2 = \sigma_\infty^2 + Q(\log_2 n) + o(1)$, where constant σ_∞^2 is approximately 1.12127 and $Q(u)$ is a periodic function with mean value 0 and period 1.*

Thus, R_n can be used for an unbiased estimator of $\log_2 n$ if the small periodic term $P(\log_2 n)$ is ignored. A much greater concern is that the variance is slightly more than one, dwarfing $P(\log_2 n)$, and implying that estimates of n will often be off by a factor of two or more in either direction. To address this, methods for reducing the variance will be discussed in Section 3.3.

3.2. Summation Sketches

As our first theoretical contribution, we generalize approximate counting sketches to handle summations. Given a multi-set of items $M = \{x_1, x_2, x_3, \dots\}$ where $x_i = (k_i, c_i)$ and c_i is a non-negative integer, the *distinct summation* problem is to calculate

$$n \equiv \sum_{\text{distinct}((k_i,c_i) \in M)} c_i.$$

When c_i is restricted to one, this is exactly the distinct counting problem.

We note that for small values of c_i, one might simply count c_i different items based upon k_i and c_i, e.g. $(k_i, c_i, 1), \dots, (k_i, c_i, c_i)$, which we denote *subitems* of (k_i, c_i). Since this is merely c_i invocations of the counting insertion routine, the analysis for probabilistic counting applies. Thus, this approach is equally accurate and takes $O(c_i)$ expected time. While very practical for small c_i values (and trivially parallelizable in hardware), this approach does not scale well for large values of c. Therefore, we consider more scalable alternatives for handling large c_i values.

Algorithm 2 SumInsert(S,x,c)

```
1:  d = pick_threshold(c);
2:  for i = 0, ... , d - 1 do
3:      S[i] = 1;
4:  end for
5:  a = pick_binomial(seed=(x, c), c, 1/2^d);
6:  for i = 1, ... , a do
7:      j = d;
8:      while hash(x,c,i,j) = 0 do
9:          j = j + 1;
10:     end while
11:     S[j] = 1;
12: end for
```

The basic intuition beyond our more scalable approach is as follows. We intend to set the bits in the summation sketch *as if* we had performed c_i successive insertions to an FM sketch, but we will do so much more efficiently. The method proceeds in two steps: we first set a prefix of the summation sketch bits to all ones, and then set the remaining bits by randomly sampling from the distribution of settings that the FM sketch would have used to set those bits. Ultimately, the distribution of the settings of the bits in the summation sketch will bear a provably close resemblance to the distribution of the settings of the bits in the equivalent FM sketch, and we then use the FM estimator to retrieve the value of the count.

We now describe the method in more detail. First, to set the prefix, we observe that it follows from Lemma 1, that the first $\delta_i = \lfloor \log_2 c_i - 2\log_2 \log c_i \rfloor$ bits of a counting sketch are set to one with high probability after c_i insertions. So our first step in inserting (k_i, c_i) into the summation sketch is to set the first δ_i bits to one. In the proof of Theorem 2 in [7], the authors prove that the case where the first δ_i bits are not all set to one only affects the expectation of R_n by $O(n^{-0.49})$. In practice, we could correct for this small bias, but we disregard it in our subsequent aggregation experiments.

The second step sets the remaining $k - \delta_i$ bits by drawing a setting at random from the distribution induced by the FM sketch we are emulating. We do so by simulating the insertions of items that set bits δ_i and higher in the counting sketch. First, we say an insertion x_i *reaches* bit z of a counting sketch if and only if $\min\{j \mid h(x_i, j) = 1\} \geq z$. The distribution of the number of items reaching bit z is well-known for FM sketches. An item x_i reaches bit z if and only if $\forall_{0 \leq j < z}(h(x_i, j) = 0)$, which occurs with probability 2^{-z}. So for a set of c_i insertions, the number of insertions reaching bit δ_i follows a binomial distribution with parameters c_i and $2^{-\delta_i}$. This leads to the following process for setting bits $\delta_i, \delta_i + 1 \ldots k$ (initialized to zero). First, draw a random sample y from $B(c_i, 2^{-\delta_i})$, and consider each of these y insertions as having reached bit δ_i. Then use the FM coin-flipping process to explicitly set the remaining bits beyond δ_i.

The pseudo-code for this approach is shown in Algorithm 2, and the analysis of its running time is presented next.

Theorem 4 *An element $x_i = (k_i, c_i)$ can be inserted into a sum sketch in $O(\log^2 c_i)$ expected time.*

Proof Sketch: Let α_i denote the number of items chosen to reach δ_i. Setting the first δ_i bits takes $O(\delta_i)$ time and simulating the α_i insertions takes expected $O(\alpha_i)$ time. The total expected time to insert x_i is then $O(\delta_i + f(\alpha_i) + \alpha_i)$, where $f(\alpha_i)$ denotes the time to pick α_i. Thus, the time depends on both α_i and the method used to pick α_i. By construction,

$$E(\alpha_i) = c_i * 2^{-\lfloor \log_2 c_i - 2\log_2 \log c_i \rfloor},$$

so

$$\log^2 c_i \leq E(\alpha_i) < 2\log^2 c_i.$$

Selecting an appropriate method for picking α_i requires more care. While there exist many efficient methods for generating numbers from a binomial distribution ([14] has a brief survey), these generally require floating point operations or considerable memory for pre-computed tables (linear in c_i). Since existing sensor motes often have neither, in Section 4.3.1 we describe a space-efficient method that uses no floating point operations, uses pre-computed tables of size $\mathrm{O}(c/\log^2 c)$, where c is an upper bound on the c_i values, and individual insertions take time $\mathrm{O}(\log^2 c_i)$. Combining these results give the stated time bound. ■

We note that for small c_i values, it may be faster to use a hybrid implementation combining the naive and scalable insertion functions. Especially for very low c_i values, the naive insertion function will be faster.

Theorem 5 *The expected value of R_n for sum sketches satisfies $E(R_n) = \log_2(\varphi n) + P(\log_2 n) + o(1)$, where φ and $P(u)$ are the same as in Theorem 2.*

Proof: The proof of this theorem follows the proof of Theorem 2 since the sum insertion function approximates repeated use of the count insertion function. Let

$$c_{\max} = \max\{c_i \mid (k_i, c_i) \in M\}$$

and

$$\delta_{\max} = \lfloor \log_2 c_{\max} - \log_2 \log c_{\max} \rfloor.$$

By the insertion method, the bottom $\delta_{\max}$ bits of $S_\Sigma(M)$ are guaranteed to be set. By construction, the remaining bits are distributed identically to those of an FM sketch with n distinct items have inserted. Thus, the distribution of R_n is the same except for the cases when the FM sketch had one of the first $\delta_{\max}$ bits not set. By Lemma 1, these cases occur with probability $O(ne^{-\log^2 n})$, so the difference in the expectation is at most $(\log_2 n - \log_2 \log n) * O(ne^{-\log^2 n})$, which is bounded (loosely) by $O(1/n)$. Therefore, $E(R_n)$ for summation sketches is within $o(1)$ of that of FM sketches. ■

Theorem 6 *The variance of R_n for sum sketches, also denoted σ_n^2, satisfies $\sigma_n^2 = \sigma_\infty^2 + Q(\log_2 n) + o(1)$, where σ_∞^2 and $Q(u)$ are as defined in Theorem 3.*

Proof: The proof of Theorem 3 is adapted in a similar fashion. ■

3.3. Improving Accuracy

To improve the variance and confidence of the estimator, FM sketches can use multiple bitmaps. That is, each item is inserted into each of m independent bitmaps to produce m R values, $R^{\langle 1\rangle}, \ldots, R^{\langle m\rangle}$. The estimate is then calculated as follows:

$$n \approx (1/\varphi) 2^{\sum_i R^{\langle i\rangle}/m}.$$

This estimate is more accurate, with standard error $O(1/\sqrt{m})$, but comes at the cost of increased insertion times ($O(m)$). To avoid this overhead, an algorithm called *Probabilistic Counting with Stochastic Averaging*, or PCSA, was proposed in [7]. Instead of inserting each item into each of the m bitmaps, each item is hashed and inserted into only one of them. Thus, each of the bitmaps summarizes approximately n/m items. While there is some variation in how many items are assigned to each bitmap, further analysis showed that the standard error of PCSA is roughly $0.78/\sqrt{m}$. Using PCSA, insertion takes $O(1)$ expected time.

PCSA can also be applied to summation sketches, but greater care must be applied when combining PCSA to summation sketches. The potential for imbalance is much larger with summation sketches - a single item can contribute an arbitrarily large fraction of the total sum. Thus, we employ the following strategy. Each c_i value has the form $c_i = q_i m + r_i$ for some integers q_i and r_i, with $0 \le r_i < m$. We then add r_i distinct items once as in standard PCSA, and then add q_i to each bitmap independently. Thus, we preserve the balance necessary for the improved accuracy and its analysis, but at the cost of $O(m \log^2(c_i/m))$ for each insertion. We will employ PCSA in our experiments.

3.4. Tradeoffs and Other Approaches

In situations where computational resources are severely constrained, it may be desirable to reduce the cost of performing insertion operations with summation sketches. We now briefly mention some tradeoffs in the computational time at the cost of increased communication and decreased accuracy. While this is unlikely to be desirable in sensor networks, given the high power costs of communication relative to computation, it may be desirable in other settings where there are large numbers of items per node.

Suppose that the largest value being inserted is bounded by y^x. Insertions with the algorithm described already take $O(x^2 \log^2 y)$ time. We can instead use x different summation sketches, each corresponding to a different digit of the c_i's using radix y. To add a c_i value, each digit of c_i is inserted into the corresponding sketch, taking expected $O(x \log^2 y)$ time, and estimates are made by summing the counting sketch estimates with the appropriate weights. The accuracy of this approach is essentially the same as before, and the increase in space is bounded by a factor of x.

An alternative approach for reducing the space overhead is to replace FM sketches with the very recently developed "loglog" sketches of [5]. The reduction of Section 3.2 can be applied similarly, again with small effects on accuracy. In parallel with our work, the sketches of [1] were adapted to summations in [17], but their methods involve both logarithms and exponentiation, making them unsuitable for sensor networks.

3.5. Other Aggregates

So far, we have only discussed two aggregates, COUNT and SUM. These techniques can also be extended to other aggregate functions beyond summation and counting. For example, AVG can also be computed directly from COUNT and SUM sketches. The second moment can be computed as an average of the squares of the items, and then combined with the average of the items, to compute the variance and standard deviation. Finally, we note that the sketches themselves are easily generalized to handle other data types such as fixed point and signed numbers, and to a certain extent, products (summing logarithms) and floating point.

4. Approximate Estimation of Duplicate Sensitive Aggregates

In this section, we show how to use duplicate insensitive sketches to build a robust, loss-resilient framework for aggregation. First, our algorithm for leveraging the broadcast nature of wireless communication in combination with sketching techniques is described in Section 4.1. A simple analytic evaluation of the proposed methods is given in Section 4.2. Finally, practical details of implementations on sensor motes are given in Section 4.3.

4.1. Algorithm and Discussion

Our methods for aggregation leverage two main observations. First, the wireless communication of sensor networks gives the ability to broadcast a single message to multiple neighbors simultaneously. Second, the duplicate-insensitive sketches discussed in Section 3 allow a sensor to combine all of its received sketches into a single message to be sent. Given proper synchronization, this will allow us to robustly aggregate data with each sensor sending just one broadcast.

For simplicity, the remainder of this section will focus on continuous queries (one-shot queries simply terminate earlier). Given a new continuous query, the computation proceeds in two phases. In the first phase, the query is distributed across the sensor network, often using some form of flooding. During this phase, each node also computes its level (i.e. its hop distance from the root), and notes the level values of its immediate neighbors. The second phase is divided into a series of *epochs* specified by the query. The specified aggregate will be computed once for each epoch.

At the beginning of each epoch, each node constructs a sketch of its local values for the aggregate. The epoch is then sub-divided into a series of rounds, one for each level, starting with the highest (farthest) level. In each round, the nodes at the corresponding level broadcast their sketches, and the nodes at the next level receive these sketches and combine them with their sketches in progress. In the last round, the sink receives the sketches of its neighbors, and combines them to produce the final aggregate.

As an example, we step through a single epoch aggregating over the topology of Figure 1. First, each node creates a fresh sketch summarizing its own observed values. In the first round of communication, nodes at level 3 broadcast their sketches, which are then received by neighboring level 2 nodes and combined with the sketches of the level 2 nodes. In the second round, nodes at level 2 broadcast their sketches, which are then received by neighboring level 1 nodes and combined with the sketches of the level 1 nodes. In the third and last round, nodes at level 1 send their sketches to the sink, which combines them and extracts the final aggregate value. Note that each node in this topology except those on the diagonals has multiple shortest paths which are effectively used, and a value will be included in the final aggregate unless all of its paths suffer from losses.

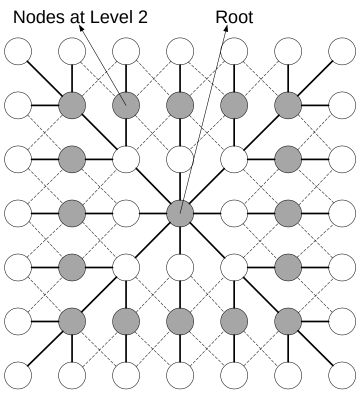


Figure 1. Routing topology for 49 node grid.

The tight synchronization described so far is not actually necessary. Our methods can also be applied using gossip-style communication - the main advantage of synchronization and rounds is that better scheduling is possible and power consumption can be reduced. However, if a node receives no acknowledgments of its broadcast, it may be reasonable in practice to retransmit. More generally, loosening the synchronization increases the robustness of the final aggregate as paths taking more hops are used to route around failures. This increased robustness comes at the cost of power consumption, since nodes broadcast and receive more often (due to values arriving later than expected) and increased time (and variability) to compute the final aggregate. As mentioned earlier, this general principle allows us to make use of any best-effort routing protocol (e.g. [13, 8]), with the main performance metric of interest being the delivery ratio.

4.2. Analysis

We now analyze the methods discussed so far for a restricted class of regular topologies. We compare the resilience of a single spanning tree against using multiple parents, but only one broadcast per node, as described in the previous section. For simplicity, we only consider exact COUNT aggregate under independent link failures; more elaborate analysis for other aggregates and failure models is possible. These calculations tend to be "back of the envelope" in nature; they illustrate the advantages of multipath routing over spanning trees for resilience. For more detailed analysis, we refer the reader to work such as [22].

In the following, we use p as the probability of (independent) link loss, and h as the maximum number of hops from the sink.

4.2.1. Fault Resilience of the Spanning Tree First, we consider a baseline routing topology in which aggregates are computed across a single spanning tree.

For simplicity, we assume that we have a complete d-ary tree of height h. In general, the probability of a value from a node at level i to reach the root is proportional to $(1-p)^i$. The expected value of the COUNT aggregate is $E(count) = \sum_{i=0}^{h}(1-p)^i n_i$, where n_i is the number of nodes at level i. This gives us $E(count) = \sum_{i=0}^{h}((1-p)d)^i = \frac{(d-pd)^{h+1}-1}{d-pd-1}$. For $h=10$, $d=3$ and $p=0.1$ (a 10% link loss rate) the expected fraction of the nodes that will be counted is poor, only 0.369.

4.2.2. Fault Resilience of Multiple Paths In order to analyze the use of multiple paths we now make a stronger assumption about the routing topology. Starting with the leaves at level 0, we assume that each node at level i has exactly d neighbors within its broadcast radius at level $i+1$, for all $0 \leq i \leq h-1$. From these neighbors, each node selects $k \leq d$ of these nodes as its parents, where k is a fault-resilience parameter, and it transmits its aggregate value to all k of these nodes. We use the pessimistic simplification that only one copy of a leaf's value reaches a level; while somewhat tighter bounds can be obtained, it suffices to provide close agreement with our experimental results. Let $\mathcal{E}_i$ denote the event that a copy of the leaf's value reached level i conditioned on the value having reached level $i+1$. With leaves at level h, these events are well-defined for levels $1, 2 \ldots h-1$. Clearly $\Pr[\mathcal{E}_i] \geq (1-p^k)$ (from the above simplification), and thus the overall probability of a message successfully reaching the root is $\Pi_i \Pr[\mathcal{E}_i] \geq (1-p^k)^h$. Using the same argument for the other levels of the tree we can get the following: $E(count) \geq \sum_{i=0}^{h}(1-p^k)^i n_i = \frac{(d-p^k d)^{h+1}-1}{d-p^k d-1}$. For $k=2$, $p=0.1$ and $h=10$ we get $E(count) \approx 0.9n$, where n is the total number of nodes. For $k=3$ the bound is close to $0.99n$, thus we have only a 1% degradation in the set of reporting sensors.

4.3. Practical Details

Since our protocols are being developed for use in sensor networks, it is important to ensure that they do not exceed the capabilities of individual sensors. Section 4.3.1 considers the computational costs of generating random numbers for the summation sketches of 3.2. Section 4.3.2 considers the bandwidth overhead of sending sketches.

4.3.1. Binomial Random Number Generation Existing sensor motes have a small word size (8 or 16 bits), lack floating point hardware and have little available memory for pre-computed tables. For these reasons, standard methods for drawing from the binomial distribution are unsuitable. Here, we outline a randomized algorithm which draws from $B(n,p)$ in $O(np)$ expected running time using $O(1/p)$ space in a pre-computed table and without use of floating point operations. We first note the following relationship between drawing from the binomial distribution and drawing from the geometric distribution, also used in [4].

Fact 1 *Suppose we have a method to repeatedly draw at random from the geometric distribution $G(1-p)$. Let d be the random variable that records the number of draws from $G(1-p)$ until the sum of the draws exceeds n. The value $d-1$ is then equivalent to a random draw from $B(n,p)$.*

The expected number of draws d from the geometric distribution using this method is np, so to bound the expected running time to draw from $B(n,p)$, we simply need to bound the running time to draw from $G(1-p)$. We will make use of the elegant alias method of [19] to do so in $O(1)$ expected time. In [19] Walker demonstrates the following (which has a simple and beautiful implementation):

Theorem 7 (Walker) *For any discrete probability density function $\mathcal{D}$ over a sample space of size k, a table of size $O(k)$ can be constructed in $O(k)$ time that enables random variables to be drawn from $\mathcal{D}$ using two table lookups.*

We can apply this method directly to construct a table of size $n+1$ in which the first n elements of the pdf $\mathcal{D}$ respectively correspond to the probabilities p_i of drawing $1 \leq i \leq n$ from the geometric distribution $G(1-p)$, and the final element corresponds to the tail probability of drawing any value strictly larger than n from $G(1-p)$. Note that for simulating a draw from $B(n,p)$ using the method implicitly defined by Fact 1, we never care about the exact value of a draw from $G(1-p)$ that is larger than n. This direct application enables $O(1)$ draws from $G(1-p)$ in $O(n)$ space, thus yields $O(np)$ expected running time to draw from $B(n,p)$.

To achieve $O(1/p)$ space, we make use of the memoryless property of the geometric distribution (which the binomial distribution does not have). Instead of storing the first n probabilities p_i for the geometric distribution, we store only the first $\lceil 1/p \rceil$ such probabilities, and a final element corresponding to the tail probability of drawing any value strictly larger than $\lceil 1/p \rceil$ from $G(1-p)$. By the memorylessness property, if we select the event corresponding to the tail probability, we can recursively draw again from the table, setting our outcome to $\lceil 1/p \rceil + x$, where x is the result of the recursive draw. The recursion terminates whenever one of the first $\lceil 1/p \rceil$ events in the table is selected, or whenever the accumulated result exceeds n. Since $1/p$ is the expectation of $G(1-p)$, this recursion terminates with constant probability at each round, and thus the expected number of table lookups is O(1). Further reduction in space is possible, but at the cost of incurring a commensurate increase in the expected number of recursive calls.

Using table sizes of $\lceil 1/p \rceil$ and assuming a maximum sensor value of $c_i \le 2^{16}$ (from a 16 bit word size), the lowest value of p used in summation sketches will be $16^2/2^{16} = 1/2^8$. Therefore, we will have tables for $p = 1/2^1, \ldots, 1/2^8$, with $2, \ldots, 2^8$ entries each, respectively. Walker's method utilizes two values for each entry - the first is an index into the table and the second is a real value used for comparison. The index value only requires one byte since the largest table size is 2^8, and a 64 bit fixed-point real value (8 bytes) should more than suffice. This gives a total table size of $\sum_{i=1}^{8}(2^i) * (1+8) = 4590$ bytes. This can be improved further by reducing the number of entries in each table as mentioned before. The smaller tables (e.g. for $p = 1/2$ and $p = 1/4$) can also be removed in favor of directly simulating the "coin flips" of the geometric distribution, but with negligible space savings.

4.3.2. Sketch Sizes and Compression As mentioned earlier, the other main limitation of sensor networks is their limited bandwidth. This limitation is a cause for concern when comparing sketching based techniques against the spanning tree strategies of TAG. While 2 bytes of data per packet will generally suffice for TAG, a single 16 bit sketch takes the same amount of space and our later experiments will actually be using 20 sketches per packet for a single aggregate. However, as one might guess from Lemma 1, these sketches are quite compressible. To leverage this, our experiments will use the compression techniques of [18]. In brief, the sketches are first "flattened", enumerating the first bit of each sketch, then the second bit of each sketch, and so on, and then the result sequence of bits is run-length encoded. This reduces the space requirements to about 30% of uncompressed versions. This is sufficient for two aggregates to be sketched within one TinyDB packet (up to 48 bytes).

5. Experimental Evaluation

In this section, we present an evaluation of our methods using the TAG simulator of [15]. Section 5.1 describes the various strategies employed for aggregation and transmission of data and Section 5.2 presents the experimental results for different scenarios.

5.1. Experimental Setup

We implemented various strategies for aggregation and transmission using the TAG simulator. Under each of these strategies, each node aggregates results received from its children with its own reading, and then sends the aggregate to one or more of its parents. Any node within broadcast range which was at a lower level (closer to the root) was considered a candidate parent. In particular, we used the following methods in our experiments:

TAG1: The main strategy of [15] (each sensor sends its aggregate to a single parent).

TAG2: The "fractional parents" strategy of [15] described in Section 2.2.

LIST: The aggregate consists of an explicit list of all the items in the aggregate with any duplicates removed. These lists are sent to all parents.

SKETCH: The strategy described in Section 4 using duplicate insensitive sketches. The default values for sketches is 20 bitmaps and 16 bits in each bitmap using the PCSA technique.

For our basic experimental network topology, we used a regular 30×30 grid with 900 sensors, where each sensor was placed at each grid point. The communication radius was $\sqrt{2}$ (allowing the nearest eight grid neighbors to be reached) and the default link loss rate was set at 5%. The root node is always at the center of the grid. Figure 1 illustrates an example of 7×7 grid.

In all the graphs, we show the average values of 500 runs. Also, for each average value, we show the 5th and 95th percentiles.

Strategy	Total	Messages	
	Data Bytes	Sent	Received
TAG1	1800	900	900
TAG2	1800	900	2468
SKETCH	10843	900	2468
LIST	170424	900	2468

Table 1. Communication Cost Comparisons

5.2. Experimental Results

First, we evaluate the communication cost of each approach. Table 1 shows the total number of bytes transmitted for a single sum query during one epoch, along with the total number of messages sent and received (assuming no losses). For TAG1 and TAG2, we assume that values are 16 bits each. SKETCH uses 20 bitmaps and the compression techniques of [18] with group size of 2 (the uncompressed size is 36000 bytes). LIST sends $(id, value)$ pairs as its message format using 32 bits in total for ids and values (two bytes each). As expected, the TAG strategies send the least data, while LIST sends the most and SKETCH is between them. We note that the message size of SKETCH can

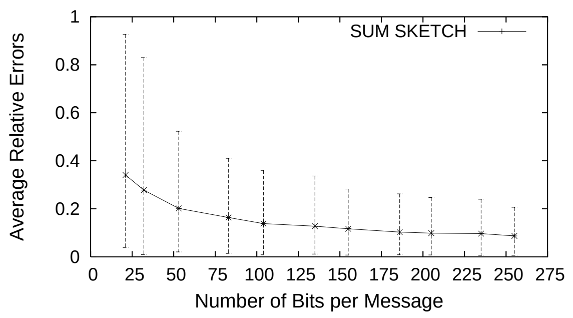
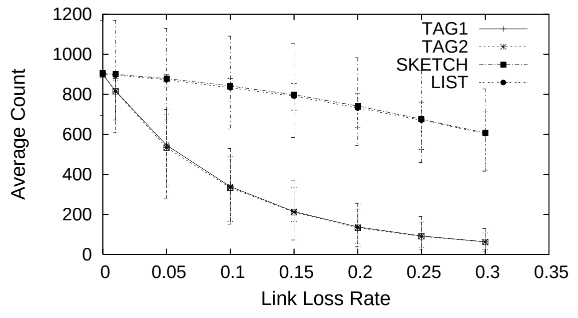
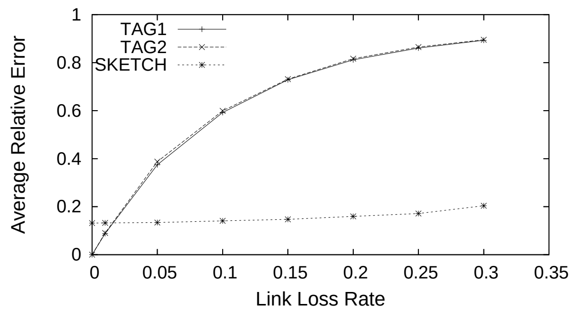


Figure 2. Number of bits in sketches vs relative error.



(a) Average counts



(b) Average relative error

Figure 3. Performance varying link loss rates.

be tuned further at the cost of accuracy by changing the number of bitmaps used, as Figure 2 illustrates. Similar results were obtained for count queries.

Figure 3 shows the effects of link losses on the performance of each strategy for a count query. In Figure 3(a), we see that for all loss rates, the average counts returned by LIST and SKETCH are extremely close, as are the average counts returned by TAG1 and TAG2. As the loss rates increase, the counts returned by LIST and SKETCH decrease slowly, while the counts returned by TAG1 and TAG2 decrease at a much higher rate. For both pairs, the main difference is that SKETCH and TAG1 have higher variation

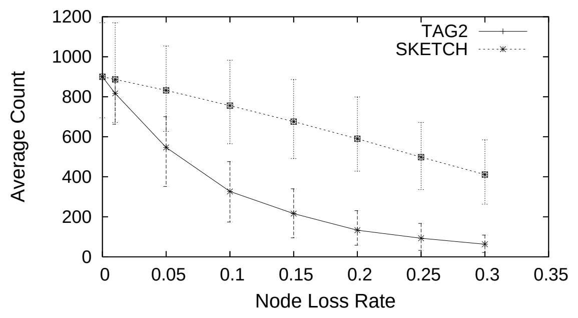


Figure 4. Performance varying node loss rates.

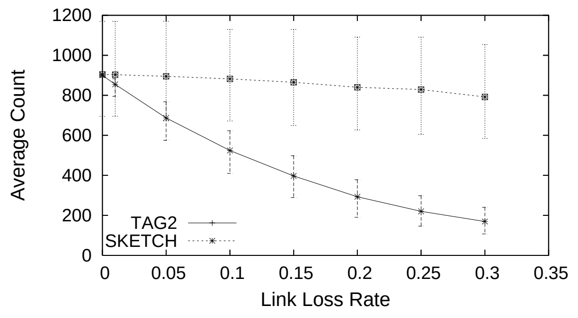


Figure 5. Performance with random placement and communication radius $2\sqrt{2}$

than LIST and TAG2 respectively.

Figure 3(b) shows the relative errors of SKETCH, TAG1, and TAG2 compared to LIST. Here, given sample value x and correct value $\hat{x}$, the relative error is $\left|\frac{x-\hat{x}}{\hat{x}}\right|$. Again, TAG1 and TAG2 are virtually identical, with quickly growing relative errors. In comparison, the relative error of SKETCH only increases a small amount, but we note that SKETCH has higher relative errors for very small loss rates. We omit plots of the relative error for most of the remaining scenarios since they have similar performance trends and are easily extrapolated from the average counts returned. We also omit plots for LIST and TAG1 since LIST is infeasible in practice, and TAG1 is strictly worse than TAG2.

Figure 4 shows the effect of node losses for the same query. The general trends here look similar to link loss plots in Figure 3(a), but the average counts reported drop off faster, while the average relative error grows more slowly. Intuitively, a major difference here for the LIST and SKETCH strategies is that a value can be "lost" if just the node fails, while all of the links to parents must fail to achieve the same loss.

Figure 5 shows the results of placing sensor nodes at random grid locations, with the communication range was increased to $2\sqrt{2}$ for the random grid placements to compensate for sparse regions of connectivity. Fig-

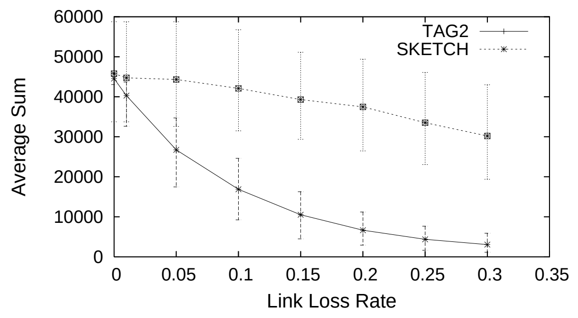
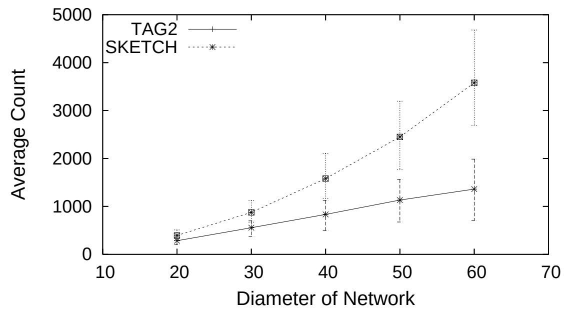
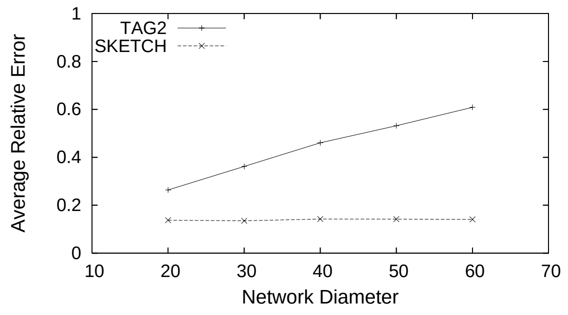


Figure 6. Performance of sum queries using different link loss rates.



(a) Average counts.



(b) Average relative error.

Figure 7. Performance varying network size.

ure 6 shows the results of using sum sketches, where each node chose an integer value uniformly at random from the range $[0, 100]$, so the expected sum is $50 * 900 = 45,000$. The basic trends in both figures were essentially the same as when just loss rates were varied. Results for AVG aggregates, combining summation and count sketches, were similar to those of SUM and were omitted.

Finally, Figure 7 shows the results of varying the network size while preserving the grid shape. Despite the loss rate being held constant, the TAG strategies perform increasingly poorly as the network size increases. Meanwhile, the SKETCH strategy maintains an almost constant average relative error around 13 percent, though it seems slightly higher for the larger network sizes (14 percent).

6. Conclusions and Future Work

We have presented new methods for approximately computing duplicate-sensitive aggregates across distributed datasets. Our immediate motivation comes from sensor networks, where energy consumption is a primary concern, faults occur frequently, and exact answers are not required or expected. An elegant building block which enables our techniques are the duplicate-insensitive sketches of Flajolet and Martin, which give us considerable freedom in our choices of how best to route data and where to compute partial aggregates. In particular, use of this duplicate-insensitive data structure allowed us to make use of dispersity routing methods to provide fault tolerance that would be inappropriate otherwise.

The implications of these results reach beyond sensor networks to other unreliable systems with distributed datasets over which best-effort aggregate queries are posed. Examples include estimating the number of subscribers participating in a multicast session, or counting the number of peers storing a copy of a given file in a peer-to-peer network. In these settings, nodes are less resource-constrained than in sensor networks, but the problems are still difficult due to packet loss and frequent node arrivals and departures.

Acknowledgments

John Byers and Jeffrey Considine are supported in part by NSF grants ANI-9986397, ANI-0093296 and ANI-0205294. George Kollios and Feifei Li are supported in part by NSF grants IIS-0133825 and IIS-0308213.

The authors would like to thank Phil Gibbons and Suman Nath for valuable feedback on versions of this manuscript, Sam Madden for providing the simulation code and answering questions about the code, and Joe Hellerstein for valuable discussions.

References

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

[2] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proc. of RANDOM*, 2002.

[3] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *VLDB*, 2002.

[4] L. Devroye. Generating the maximum of independent identically distributed random variables. *Computers and Mathematics with Applications*, 6:305–315, 1980.

[5] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *ESA*, 2003.

[6] P. Flajolet. On adaptive sampling. *COMPUTG: Computing*, 43, 1990.

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

[8] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. *ACM Mobile Computing and Communications Review*, 5(4), 2001.

[9] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *ACM SIGMOD*, 2003.

[10] P.B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 281–291, 2001.

[11] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[12] M. Horton, D. Culler, K. Pister, J. Hill, R. Szewczyk, and A. Woo. Mica, the commercialization of microsensor motes. 19(4):40–48, April 2002.

[13] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCOM*, 2000.

[14] V. Kachitvichyanukul and B.W. Schmeiser. Binomial random variate generation. *Communications of the ACM*, 31(2):216–222, 1988.

[15] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *OSDI*, 2002.

[16] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *ACM SIGMOD*, 2003.

[17] S. Nath and P.B. Gibbons. Synopsis Diffusion for Robust Aggregation in Sensor Networks. Technical Report ITR-03-08, Intel Research Pittsburgh, August 2003.

[18] C. Palmer, P. Gibbons, and C. Faloutsos. ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs. In *SIGKDD*, 2002.

[19] A.J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):253–256, 1977.

[20] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record 31(3)*, 2002.

[21] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[22] F. Ye, G. Zhong, S. Lu, and L. Zhang. GRAdient Broadcast: A Robust Data Delivery Protocol for Large Scale Sensor Networks. *ACM Wireless Networks (WINET)*, 11(2), 2005.

[23] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *SNPA*, 2003.