Tributaries and Deltas: Efficient and Robust Aggregation in Sensor Network Streams

Amit Manjhi, Suman Nath, and Phillp B. Gibbons

IRP-TR-05-02

Research at Intel

Tributaries and Deltas: Efficient and Robust Aggregation in Sensor Network Streams

Amit Manjhi
Carnegie Mellon University
manjhi@cs.cmu.edu

Suman Nath
Carnegie Mellon University
sknath@cs.cmu.edu

Phillip B. Gibbons
Intel Research Pittsburgh
phillip.b.gibbons@intel.com

ABSTRACT

Existing energy-efficient approaches to in-network aggregation in sensor networks can be classified into two categories, tree-based and multi-path-based, with each having unique strengths and weaknesses. In this paper, we introduce Tributary-Delta, a novel approach that combines the advantages of the tree and multi-path approaches by running them simultaneously in different regions of the network. We present schemes for adjusting the regions in response to changes in network conditions, and show how many useful aggregates can be readily computed within this new framework. We then show how a difficult aggregate for this context—finding frequent items—can be efficiently computed within the framework. To this end, we devise the first algorithm for frequent items (and for quantiles) that provably minimizes the worst case total communication for non-regular trees. In addition, we give a multi-path algorithm for frequent items that is considerably more accurate than previous approaches. These algorithms form the basis for our efficient Tributary-Delta frequent items algorithm. Through extensive simulation with real-world and synthetic data, we show the significant advantages of our techniques. For example, in computing Count under realistic loss rates, our techniques reduce answer error by up to a factor of 3 compared to *any* previous technique.

1. INTRODUCTION

Networked collections of smart sensors are increasingly being used to monitor and query the physical world. These small sensor *motes* are typically battery-powered, possess limited CPUs and memory, and organize themselves into ad hoc multi-hop wireless networks around more capable base stations. A paramount concern in these sensor networks is to conserve the limited battery power, as it is usually impractical to install new batteries in a deployed sensor network. Because the battery drain for sending a message between two neighboring sensors exceeds by several orders of magnitude the drain for local operations within a sensor mote, minimizing sensor communication is a primary means for conserving battery power [1, 17]. Thus for aggregation queries (e.g., the average temperature reading across the sensor network), it is now accepted practice [10, 11, 21] that aggregates are computed *in-network* whenever possible—this avoids the excessive communication required to route all the sensor readings to the base station. With the in-network approach, sensor readings are accumulated into partial results that are combined as messages propagate toward the base station. In many common cases (such as Sum, Count, Average, Min, Max), each sensor node transmits only one short message during the aggregation—a considerable energy savings over the route-all approach.

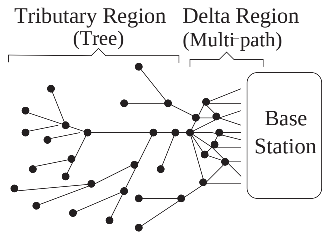


Figure 1: Tributaries and Deltas

Existing energy-efficient approaches to in-network aggregation can be classified into two categories: tree-based and multi-path-based. In the tree-based approach (such as in TAG [10], TinyDB [11], and Cougar [21]), a spanning tree, rooted at the base station, is constructed for use in answering queries. Subsequently, each query answer is generated by performing in-network aggregation along the tree, proceeding level-by-level from its leaves to its root. Many aggregates (including those given above) can be computed exactly and with minimal communication on a tree topology, assuming no communication failures. However, wireless sensor networks have high communication failure rates (up to 30% loss rate is common [22]), and each dropped message results in an entire subtree of readings being dropped from the aggregate answer. As a result, it is not uncommon to lose 85% of the readings in a multi-hop sensor network, causing significant answer inaccuracy [15].

To overcome the severe robustness problems of the tree approach, two recent papers [5, 15] propose using multi-path routing for in-network aggregation. Instead of having each node send its accumulated partial result to its *single* parent in an aggregation tree, the multi-path approach exploits the wireless broadcast medium by having each node broadcast its partial result to *multiple* neighbors. Both papers recommend a topology called *rings*, in which the nodes

An abridged version of this paper appears in ACM SIGMOD Conference 2005: Baltimore, Maryland, USA.

	Energy Components			Error Components			Latency
	Number of messages	Message size		Communication error	Approximation error		
Aggregate:	*any*	*Count*	*Freq.Items*	*any*	*Count*	*Freq.Items*	*any*
Tree [10, 11, 21, 23]	minimal	small	medium	very large	none	small	minimal
Multi-path (rings) [5, 15]	minimal	small	large	very small	small	small	minimal
Tributary-Delta [this paper]	minimal	small	medium	very small	very small	small	minimal

Table 1: Comparison of previous in-network aggregation approaches and the Tributary-Delta approach. The total energy consumption is given by its two components. The total error is given by the sum of the *communication error* produced by message losses within the network and the *approximation error* coming from the aggregation algorithm (independent of message loss). Because the message size and approximation error depend on the aggregate, these metrics are shown for two representative aggregates: Count and Frequent Items.

are divided into levels according to their hop count from the base station, and the multi-path aggregation is performed level-by-level toward the base station. This approach sends the same minimal number of messages as the tree approach (i.e., one transmission per node), making it energy-efficient. It is also very robust to communication failures because each reading is accounted for in many paths to the base station, and *all* would have to fail for the reading to be lost. However, there are two drawbacks to the multi-path approach: (1) for many aggregates, the known energy-efficient techniques provide only an *approximate* answer (with accuracy guarantees), and (2) for some aggregates, the message size is longer than when using the tree approach, thereby consuming more energy.

The first two rows of Table 1 provide a qualitative comparison of the two previous in-network aggregation approaches. For multi-path, we consider the rings topology. As the table shows, the tree approach suffers from very high communication error while the multi-path approach can have larger message sizes and approximation errors.

Tributary-Delta. In this paper we present a new approach to in-network aggregation that combines the advantages of both the tree and multi-path approaches, by dynamically adapting the aggregation scheme to the current message loss rate. Under low loss rates, trees are used for their low or zero approximation error and their short message size. Under higher loss rates or when transmitting partial results accumulated from many sensor readings, multi-path is used for its robustness. We call our approach *Tributary-Delta* because of the visual analogy to a river flowing to a gulf: when far from the gulf, the merging of river *tributaries* forms a tree-like shape, whereas near the gulf, the river branches out into a multi-path *delta* in order to reach the gulf despite the increased obstacles (see Figure 1).

We show that our Tributary-Delta approach significantly outperforms both previous approaches. An example result is shown in Figure 2 (full details in Section 7). As expected, the tree approach is more accurate than the multi-path approach at very low loss rates, because of its lower approximation error (0% versus 12%). However, at loss rates above 0.05%, tree is much worse than multi-path because of its high communication error. On the other hand, Tributary-Delta provides not just the best of both (e.g., from running either tree or multi-path in the entire network), but in fact provides a significant error reduction over the best, across a wide range of loss rates—thus demonstrating the synergies of using both in tandem. The last row of Table 1 summarizes the benefits of Tributary-Delta.

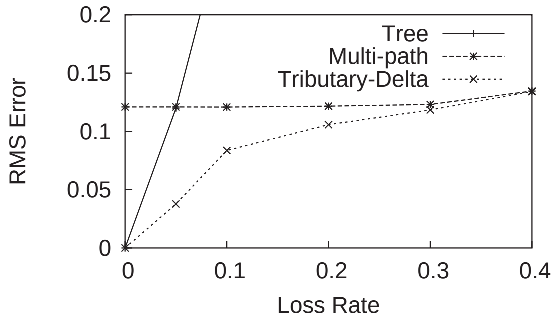


Figure 2: RMS error of a Count query under varying message loss rates. The experimental setup and the full graph are provided in Section 7.

To enable simultaneous use of the tree and multi-path aggregation approaches, we must resolve several challenges. For example, how do the sensor nodes decide whether to use the tree or the multi-path aggregation approach? How do nodes using different approaches communicate with each other? How do nodes convert partial results when transitioning between approaches? We identify and address these and other challenges in this paper, through a careful system design and algorithmic study. We also discuss how a large number of aggregates can be computed within the Tributary-Delta framework.

Our most significant algorithmic result is a new Tributary-Delta algorithm for finding frequent items. For this result, we devise a new tree-based algorithm, a new multi-path-based algorithm, and a new (combined) Tributary-Delta algorithm. Previous *tree-based* frequent items algorithms worked only for balanced, regular[1] trees [12] and/or used too much communication [8, 12]. We present the first frequent items algorithm that provably minimizes the worst case total communication for non-regular trees that have certain properties common to typical sensor network deployments. In addition, our new *multi-path* algorithm uses low total communication while providing high accuracy; the only previous approaches [15] are far less accurate.

Contributions. In summary, the main contributions of this paper are:

- We introduce the Tributary-Delta approach to in-network aggregation in sensor networks, for adapting the aggregation scheme to current message loss rates. We

[1]A *regular* tree is one where each internal node has the same number of children.

show that many aggregates can be readily computed in this framework.

- We present schemes for adjusting the balance between tributaries and deltas in response to changes in network conditions.
- We present a novel Tributary-Delta algorithm for finding frequent items—a difficult aggregate for this context. To this end, we devise the first algorithm for frequent items (and for quantiles) that provably minimizes worst case total communication for non-regular trees. The algorithm's guarantees hold for a class of trees that arise naturally in sensor networks; we also present a new tree construction algorithm well-suited to generating trees in this class. In addition, we give a multi-path algorithm for frequent items that is considerably more accurate than previous approaches.
- We provide an extensive evaluation of Tributary-Delta aggregation on a realistic sensor network simulator, using real-world and synthetic data sets, confirming the significant advantages of our techniques. For example, in computing Count under typical loss rates ($0-40\%$), Tributary-Delta reduces errors by up to a factor of 3 compared to the best existing approach for that rate.

Although the general framework encompasses optimizing many possible metrics based on the criteria in Table 1, in this paper we focus on the following setting. Users provide target thresholds on both the communication error (e.g., at least 90% of the nodes should be accounted for in the answer) and the approximation error (e.g., the answer should be within 10% of the actual answer of the query applied to the "accounted for" nodes). Our goal is to achieve these thresholds while incurring minimal latency, using a minimal number of messages, and minimizing the message size.

Roadmap. Section 2 describes background information and related work. Section 3 overviews our Tributary-Delta approach. Section 4 presents our adaptation design. Section 5 discusses Tributary-Delta algorithms for many aggregates. Section 6 presents our frequent items algorithm. Section 7 presents our experimental results. Finally, Section 8 presents conclusions.

2. PRELIMINARIES AND RELATED WORK

There has been a flurry of recent papers on energy-efficient, in-network computation of aggregate queries in sensor networks [5, 6, 8, 10, 11, 15, 18, 21]. As discussed in Section 1, this previous work can be classified according to the aggregation topology used: tree-based or multi-path-based. In this section, we describe these two approaches in more detail and survey the related work. We begin by describing the general set-up used in this paper.

Aggregation Set-up. We have m sensor nodes each generating a stream of sensor readings. The sensor nodes are connected (either directly or via other sensor nodes) to a base station. Aggregate queries, which may be one-time or continuous, are sent from the base station to all the nodes. Queries may aggregate over a single value at each sensor (e.g., the most recent reading) or over a window of values from each sensor's stream of readings. Each sensor node evaluates the query locally (including any predicates), and produces a local result. There is an aggregation topology (e.g., tree or rings) that is used to route these local results to the base station, combining them along the way into concise partial results. For continuous queries, the process of computing, routing and combining local results repeats itself at regular intervals, possibly in a pipelined fashion within the network [10].

We consider the realistic setting where the communication between sensors may be lossy, and network conditions change over time. In evaluating the quality of an answer, we consider both the *communication error*, which results from message losses in the network, and the *approximation error*, which results from lossy data reduction performed to reduce message lengths [15].

Tree-Based. In the tree-based approach [6, 8, 10, 11, 18, 21], a spanning tree rooted at the base station is constructed for use in answering queries. Each node computes its *level* (i.e., minimum number of hops from the root) in the tree during this construction by adding one to the level of its parent. In-network aggregation proceeds level-by-level toward the root, starting with the nodes at the highest level. To coordinate the sending and receiving of messages, nodes are loosely time synchronized and are allotted specific time intervals (according to their level) when they should be awake to send and receive messages. In this way, level i nodes are listening when level $i+1$ nodes are sending. The period of time allotted for exchanging messages between two levels is called an *epoch* [10]. Epochs must be sufficiently long such that each sensor in a level can transmit its message once without interference from other sensors' transmissions. The *latency* of a query result is dominated by the product of the epoch duration and the number of levels.

To adapt the tree to changing network conditions, each node monitors the link quality to and from its neighbors [23]. This is done less frequently than aggregation, in order to conserve energy. If the relative link qualities warrant it, a node will switch to a new parent with better link quality, in order to make the tree more robust [23]. However, because each lost message drops an entire subtree, even trees with high link quality produce very inaccurate answers once the tree is beyond a certain size. This inaccuracy has been previously studied [15] and can also be seen in Figure 2.

A key advantage of using a tree topology is that aggregating within the network is often straightforward, using minimal resources and incurring no approximation error. For example, for a Sum query, each node transmits the sum of the readings in its subtree, by listening for its children's subtree sums, adding them to its own readings, and sending the result to its parent. In the absence of communication error, the resulting sum would be exact.

Multi-Path-Based. The multi-path-based approach [5, 15] allows for arbitrary aggregation topologies, beyond a tree. Note that multi-path aggregation in general does not require a longer epoch length [15]. In this paper, we focus on the *rings* topology, because it provides a good energy-robustness trade-off [5, 15]. To construct a rings topology, first the base station transmits and any node hearing this transmission is in ring 1. At each subsequent step, nodes in ring i transmit and any node hearing one of these transmissions—but not already in a ring—is in ring $i+1$. The ring number defines the *level* of a node in the rings topology. Aggregation proceeds level-by-level, with level $i+1$ nodes transmitting while level i nodes are listening. In contrast to trees, the rings topology exploits the wireless broadcast medium by having *all* level i nodes that hear a level $i + 1$ partial result incorporate that result into their own. This significantly increases robustness because each reading is accounted for in many paths to the base station, and *all* would have to fail for the reading to be unaccounted for in the query result. As with trees, nodes can monitor link quality and change levels as warranted.

A key advantage of using a rings topology is that the communication error is typically very low, in stark contrast with trees. This can be seen in Figure 2, where the accuracy of multi-path decreases very slowly with increasing loss rates (the approximation error is around 12% in this experiment, independent of the loss rate). Moreover, the rings approach is as energy-efficient as the tree approach (within 1% [15]).

However, because each partial result is accounted for in multiple other partial results, special techniques are required to avoid double-counting. Previous work has shown how to avoid double-counting in computing Count, Sum, and many other aggregates [5, 15]. For this paper, we adopt our terminology of [15], where the multi-path approach is called *synopsis diffusion*.[2] There are three functions used to compute an aggregate: (1) A *synopsis generation (SG)* function that takes a stream of local sensor readings at a node and produces a partial result, called a *synopsis*; this function is applied by each node on its local readings. (2) A *synopsis fusion (SF)* function that takes two synopses and generates a new synopsis that summarizes both; this function is applied when combining partial results in-network. (3) A *synopsis evaluation (SE)* function that translates a synopsis into a query answer; this function is applied at the base station.

We next illustrate these functions using the Count aggregate; this example highlights one of the clever techniques used to avoid double-counting in the multi-path approach.

Example 1. Consider answering a Count query requesting the number of live sensors. To avoid double-counting, we view the `Count` query as a `Count Distinct sensor-id` query, and use a well-known distributed distinct-values counting algorithm [7], as follows [5, 15]. Let n be a (possibly loose) upper bound on the total number of sensors. The synopsis is a bit vector of $\log(n)$ bits. Let $h()$ be a hash function from sensor-ids to $[1..\log(n)]$ such that a random $\frac{1}{2}$ of the domain maps to 1, $\frac{1}{4}$ maps to 2, $\frac{1}{8}$ maps to 3, etc. The SG function creates a bit vector of all 0's and then sets the $h(i)$'th bit to 1, where i is the sensor node's unique sensor-id. The SF function takes two bit vectors and outputs their bit-wise OR. The SE function takes a bit vector and outputs $2^{j-1}/0.77351$, where j is the index of the lowest-order unset bit. The approximation guarantees are provided in [7]—intuitively, if no node sets the jth bit then there are probably less than 2^j nodes. The algorithm avoids double-counting, intuitively, because each sensor i is associated with the $h(i)$'th bit being set and ORing that bit in multiple times is identical to ORing it in just once. The accuracy of this algorithm can be improved by using multiple bit vectors based on different hash functions, at a cost of sending longer messages [5, 15].

Other Related Work. Many papers have presented techniques for computing aggregates over data streams, including distributed data streams (see [2, 14] for surveys). Several recent papers [4, 20] have proposed duplicate-insensitive multi-path in-network aggregation techniques for peer-to-peer networks or mobile environments. None of their techniques are suitable for the sensor network setting in which reducing energy consumption is of paramount importance.

[2]For readers familiar with the synopsis diffusion framework, note that we will deviate slightly from [15] for some of the aggregates discussed in the paper (Frequent Items, Quantiles), by relaxing the requirement that the *same synopsis* be generated regardless of the aggregation topology.

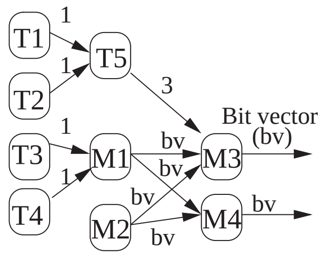


Figure 3: Combining Tree and Multi-path algorithms for computing Count in the Tributary-Delta framework. *Bit vector (bv)* is the multi-path synopsis for the Count aggregate [5, 7].

In summary, none of the previous work has proposed and studied combining the complementary strengths of the two approaches in order to obtain the best of both (and more).

3. TRIBUTARY-DELTA APPROACH

In our Tributary-Delta aggregation scheme, we leverage the synergies between the existing energy-efficient schemes, by combining the efficiency and accuracy of (small) trees under low loss rates with the robustness of multi-path schemes. Specifically, part of the network runs a multi-path scheme while at the same time the rest of the network runs tree schemes. In the extreme, all nodes might either run a multi-path or a tree scheme. We dynamically adjust the use of trees and multi-path, based on current message loss rates. In this section we provide an overview of our Tributary-Delta scheme.

We begin by defining a directed graph G representing the aggregation topology during a Tributary-Delta aggregation. The sensors and the base station form the set of vertices of G, and there is a directed edge for each successful transmission. Each vertex is labeled either $\mathcal{M}$ (for multi-path) or $\mathcal{T}$ (for tree) depending on whether it runs a multi-path aggregation algorithm or a tree aggregation algorithm. An edge is assigned the same label as that of its source vertex. Note that both the set of edges and the labels of individual vertices and edges may change over time. Figure 1 depicts an example graph G, where the edges are directed to the right in the figure. Figure 3 depicts a portion of another example graph, where T1–T5 are $\mathcal{T}$ vertices and M1–M4 are $\mathcal{M}$ vertices.

There are many ways to construct an aggregation topology with both $\mathcal{M}$ and $\mathcal{T}$ vertices. The basic correctness criteria is that no two $\mathcal{M}$ vertices with partial results representing an overlapping set of sensors are connected to a $\mathcal{T}$ vertex. This is necessary, since otherwise the corresponding $\mathcal{T}$ vertex, whose local aggregation algorithm is duplicate-sensitive, may double-count the same sensor data and provide an incorrect answer. Formally, for every maximal subgraph G' consisting of $\mathcal{M}$ vertices, there is exactly one vertex $m \in G'$ directly connected to a $\mathcal{T}$ vertex in $G - G'$ and every vertex $v \in G'$ has a path to m. Ensuring this requires electing a suitable leader (m) within G'. This general construction, although achievable, thwarts our objectives: it restricts the amount of available redundancy an $\mathcal{M}$ vertex can exploit and the leader election process complicates the aggregation process.

We therefore restrict ourselves to a simpler model where a sensor receiving a partial result from an $\mathcal{M}$ vertex uses a multi-path aggregation scheme. This ensures that a partial result from an $\mathcal{M}$ vertex never reaches a $\mathcal{T}$ vertex downstream towards the base station, and therefore a $\mathcal{T}$ node never gets the chance to double-count sensor data.

In terms of the graph, this correctness condition can be formulated as either an **Edge Correctness** property (Property 1) or a **Path Correctness** property (Property 2). The two properties are equivalent—both formulations are useful depending on the context.

PROPERTY 1. **Edge Correctness**: *An $\mathcal{M}$ edge can never be incident on a $\mathcal{T}$ vertex, i.e., an $\mathcal{M}$ edge is always between two $\mathcal{M}$ vertices.*

PROPERTY 2. **Path Correctness**: *In any directed path in G, a $\mathcal{T}$ edge can never appear after an $\mathcal{M}$ edge.*

An implication of *path correctness* is that the $\mathcal{M}$ vertices will form a subgraph (a multi-path "delta") that includes the base station, which is fed by trees of $\mathcal{T}$ vertices ("tributaries"), as depicted in Figure 1. Let the *delta region* of G be the set of $\mathcal{M}$ vertices. Coincidentally, any graph G satisfying *path correctness* is also desirable for high accuracy—partial results near the base station account for larger numbers of sensor readings than partial results near the leaves of G, and hence the additional robustness provided by the delta region significantly improves answer accuracy.

Our Tributary-Delta scheme requires multi-path algorithms that can operate on (approximate or exact) partial results from both tree and multi-path schemes. For example, M3 in Figure 3 receives inputs from both a $\mathcal{T}$ vertex and two $\mathcal{M}$ vertices. We address this algorithmic challenge in Section 5.

Dynamic Adaptation. Our goal is to dynamically adapt where in the sensor network we use trees versus where we use multi-path, based on current message loss rates in various regions of the network. However, an implication of *edge correctness* is that individual vertices cannot switch between the two modes independently. We say an $\mathcal{M}$ vertex is *switchable* (to a $\mathcal{T}$ vertex) if all its incoming edges are $\mathcal{T}$ edges or it has no incoming edges. Similarly, a $\mathcal{T}$ vertex is *switchable* if its parent is an $\mathcal{M}$ vertex or it has no parent. In Figure 3, vertices T3, T4, T5, M1, and M2 are switchable. Based on these two definitions, we make the following observation.

OBSERVATION 1. **All** *children of a switchable $\mathcal{M}$ vertex are switchable $\mathcal{T}$ vertices.*

Note that a delta region uniquely defines the set of switchable $\mathcal{M}$ and $\mathcal{T}$ vertices in G. The next lemma implies that by considering only the *switchable* $\mathcal{T}$ and $\mathcal{M}$ vertices, it is always possible to expand (or shrink) the delta region if it makes sense to do so. Let G' be the connected component of G that includes the base station. Then expanding (shrinking) the delta region only makes sense if there is a $\mathcal{T}$ vertex (an $\mathcal{M}$ vertex, respectively) in G'. A simple induction proof yields the following result:

LEMMA 1. *If the set of $\mathcal{T}$ vertices in G' is not empty, at least one of them is switchable. If the set of $\mathcal{M}$ vertices in G' is not empty, at least one of them is switchable.*

PROOF. If the base station is a $\mathcal{T}$ vertex, it is switchable by definition. Otherwise, either there is a $\mathcal{T}$ vertex in the next level (which would be a switchable vertex) or *all* nodes in next level are $\mathcal{M}$. In this latter case, the argument proceeds inductively.

Similarly, if at least one leaf vertex (vertices with no incoming edges) is an $\mathcal{M}$, that vertex is switchable. Otherwise, all leaves are $\mathcal{T}$ vertices and we proceed inductively by considering vertices whose children are all leaves. $\square$

In the next section, we study strategies for adapting the tributary and delta regions to changing network conditions.

4. ADAPTING TO NETWORK CONDITIONS

In this section we study in detail how our Tributary-Delta scheme dynamically adapts to changing network conditions.

4.1 Adaptation Design

We first discuss a number of practical issues that arise in designing our adaptation strategies.

Adaptation Decision. Recall from Section 3 that the only possible ways to adapt to changing network conditions are (1) to shrink the delta region by switching switchable $\mathcal{M}$ vertices (multi-path nodes) to $\mathcal{T}$ vertices (tree nodes) or (2) to expand the delta region by switching switchable $\mathcal{T}$ vertices (tree nodes) to $\mathcal{M}$ vertices (multi-path nodes). However, because of the different types of errors introduced by the tree and multi-path schemes (recall Table 1), it is unclear how switching one or more nodes impacts the answer accuracy. Therefore, we require users to specify a threshold on the minimum percentage of nodes that should contribute to the aggregate answer. It then becomes natural for the base station to be involved in the decision process: depending on the % of nodes contributing to the current result, the base station decides whether to shrink or expand the delta region for future results. Because there is only minimal communication error in multi-path schemes (recall Figure 2), increasing the delta region always increases the % contributing. Similarly, decreasing the delta region always decreases the % contributing. The system seeks to match the target % contributing, in order to take advantage of the smaller approximation error in tree aggregation. Because this design does not rely on the specifics of any one query, the resulting delta region is effective for a variety of concurrently running queries. Designs specialized to *particular* queries are part of our future work.

Synchronization. A key concern in switching individual nodes from tree aggregation to multi-path aggregation (and vice-versa) is how to ensure that nodes that *should* be communicating after the switch are indeed sending and receiving during the same epoch. When a node switches from $\mathcal{M}$ to $\mathcal{T}$, it needs to change its sending epoch to match its new parent's listening epoch and change its new children's sending epoch to match its listening epoch, etc. Conversely, when a node switches from $\mathcal{T}$ to $\mathcal{M}$, it needs to change its sending epoch to match the listening epoch of its neighboring nodes in the next level and change its children's sending epoch to match its listening epoch, etc. This re-synchronization overhead could arise, for example, if TAG [10], a popular tree aggregation approach, were to be used together with rings for multi-path, and it would be a large deterrent to switching between tree and multi-path schemes. To ensure that no such re-synchronization is necessary, we make a simplifying design choice: a node in level i when switching from $\mathcal{M}$ to $\mathcal{T}$ must choose its tree parent from one of its neighbors in level $i-1$. Similarly, when the node switches from $\mathcal{T}$ to $\mathcal{M}$, it transmits to all its neighbors in level $i-1$, including its parent. In other words, all tree links should be a subset of the links in the ring. This ensures that the switched node can retain its current epoch, since the new parent in level $i-1$ is already synchronized to receive data from the node in level i. Trees constructed with this restriction may have inferior link quality; however, this is mitigated with Tributary-Delta because (1) we use multi-path to overcome poor link quality and (2) our tree construction algorithm (see Section 6.1.3), which guarantees that tree links are a subset of rings links, produces bushy trees that are effective in reducing total communication errors.

Oscillation. Base station's desire to match the accuracy of the final result to the user-defined threshold may lead to a repeated expansion and shrinking sequence of the delta region. This can happen when expansion of the delta region improves the accuracy to significantly above the threshold so that for reducing the overhead of multi-path aggregation, the base station decides to shrink it; and shrinking the delta region reduces the accuracy to below the threshold, and hence it needs to be expanded again. This situation can be common when a large number of nodes simultaneously switch their states. Such an oscillation can unnecessarily increase the adaptation overhead. There are two ways to prevent such oscillations. First, the delta region can be expanded or shrunk at a small granularity, e.g., one node at a time. After a small adjustment of the delta region boundary, the base station can wait to see the result of the adjustment before further adjustment. Second, the base station can use heuristics to damp down the oscillation. For example, if it experiences a repeated sequence of expansion and shrinking, it can simply stop the repetition or can gradually reduce the frequency of adjustments.

4.2 Adaptation Strategies

In this section, we present two alternative strategies to shrink and expand the delta region. In both strategies, we augment the messages being sent between nodes with an (approximate) Count of the number of nodes contributing to the partial result being sent. Assuming that the base station knows the number of sensors in the network, it can compute the % contributing to the current result.

Strategy `TD-Coarse`. In the first strategy, `TD-Coarse`, if the % contributing is below the user-specified threshold, the base station expands the delta region by broadcasting a message asking *all* the current switchable $\mathcal{T}$ nodes to switch to $\mathcal{M}$ nodes. This effectively widens the delta region by one level. Similarly, if the % contributing is well above the threshold, it shrinks the delta region by one level by switching *all* current switchable $\mathcal{M}$ nodes to $\mathcal{T}$ nodes. The coarse-grained control of `TD-Coarse` is well-suited to quickly adapting the size of the delta region to network-wide fluctuations. However, it can not adapt well to different conditions in different parts of the network; for this, we introduce the following more fine-grained strategy.

Strategy `TD`. In the second strategy, `TD`, we use the existence of the parent-child relationship among switchable $\mathcal{M}$ nodes and switchable $\mathcal{T}$ nodes (Observation 1), as follows. Each switchable $\mathcal{M}$ node includes in its outgoing messages an additional field that contains the number of nodes in its subtree that did *not* contribute.[3] As the multi-path aggregation is done, the maximum, *max*, and the minimum, *min*, of such numbers are maintained. If the % contributing is below the user-specified threshold, the base station expands the delta region by switching from $\mathcal{T}$ to $\mathcal{M}$ all children of switchable $\mathcal{M}$ nodes belonging to a subtree that has *max* nodes not contributing. In this way, subtrees with the greatest robustness problems are targeted for increased use of multi-path. Shrinking is done by switching each switchable $\mathcal{M}$ node whose subtree has only *min* nodes not contributing. The fine-grained control of `TD` facilitates adapting to non-uniform network conditions, at a cost of higher convergence time and additional message overhead because the base station needs to send one message every time it switches a small number of nodes. Note that there are many possible heuristics to improve the adaptivity of `TD`, such as using $max/2$ instead of *max* or maintaining the top-k values instead of just the top-1 value (*max*). Exploration of optimal heuristics is part of our future work.

5. COMPUTING SIMPLE AGGREGATES

To compute an aggregate in our Tributary-Delta framework, we need a corresponding tree algorithm, a multi-path algorithm, and a *conversion function* that takes a partial result generated by the tree algorithm and outputs a synopsis that can be used by the multi-path algorithm. For example, in Figure 3, the node M3 receives two multi-path partial results (denoted as bv) and one tree partial result (3). The conversion function needs to transform the tree result to a synopsis so that M3 can use its synopsis fusion function to combine it.

The synopsis generated by the conversion function must be valid over the inputs contributing to the tree result. For example, the conversion function for the Count aggregate of Example 1 (Section 2) should take the output of the tree scheme–a subtree count c–and generate a synopsis that the multi-path scheme equates with the value c. Intuitively, this enables a node running a multi-path algorithm to become oblivious to whether an input synopsis is from a multi-path node or the result of a conversion function applied to a tree result.

Many aggregates (e.g., Count, Sum, Min, Max, Average, Uniform sample, etc.) with known efficient multi-path [15] and tree algorithms have simple conversion functions[4], and hence can be efficiently computed in our Tributary-Delta framework. Moreover, the Uniform sample algorithm can be used to compute various other aggregates (e.g., Quantiles, Statistical moments) using the framework.

However, there are no previous, efficient multi-path algorithms for identifying frequent items in sensor networks.

[3]Note that there is no double-counting here because it follows from the *path correctness* property that the node is the root of a unique subtree.

[4]Since the multi-path algorithms for computing Min, Max, and Uniform sample (using our approach in [15]) have *no* approximation error, the tree algorithms can be same as their multi-path counterparts. For these aggregates then, the "identity function" suffices as the conversion function. For the Count and Sum aggregate, the Sum synopsis generation function [5] can be used as the conversion function. Lastly, a separate conversion function for Average is not required since Sum and Count can be used to compute Average.

This is an important aggregate particularly in the context of biological and chemical sensors, where individual readings can be highly unreliable and it is necessary to get a consensus measure [18]. In the next section, we present an algorithm to do so, as well as an efficient tree algorithm and the corresponding conversion function. Together, these enable computing frequent items in our Tributary-Delta framework.

6. IDENTIFYING FREQUENT ITEMS

In this section we present the first energy-efficient Tributary-Delta algorithm for finding frequent items, describing first the tree scheme (Section 6.1), then the multi-path scheme (Section 6.2), and finally the conversion function (Section 6.3).

Following [13, 12], we consider the following formulation of the frequent items problem. Each of the m sensor nodes generates a collection of *items*. For example, an item can be a value of a sensor reading at a particular point in time. The same "item" may appear multiple times at one or more sensor nodes. Let $c(u)$ be the frequency of occurrence of item u over all m nodes. Given a user-supplied error tolerance ϵ, the goal is to obtain for each item u, an *ϵ-deficient count* $\tilde{c}(u)$ at the base station, where each $\tilde{c}(u)$ satisfies

$$\max\{0, c(u) - \epsilon \cdot N\} \leq \tilde{c}(u) \leq c(u)$$

and N denotes the sum of item occurrences across all items, i.e., $N = \sum_u c(u)$. By computing ϵ-deficient counts, communication is not wasted aggregating counts for rare items (i.e., items with $c(u) \leq \epsilon \cdot N$). Moreover, for small ϵ values, little error is introduced by using ϵ-deficient counting for frequent items (i.e., items with $c(u) \gg \epsilon N$). Given a user specified support threshold s ($s \gg \epsilon$), similar to [13, 12], we report as frequent all items with ϵ-deficient counts greater than $(s-\epsilon)N$, thus ensuring that there are no false negatives, and all false positives have frequency at least $(s - \epsilon)N$.

6.1 Tree Algorithm

In this subsection, we present our tree-based frequent items algorithm, MIN TOTAL-LOAD, the first algorithm for identifying frequent items that uses only $O(\frac{m}{\epsilon})$ words[5] of total communication, which is optimal. The algorithm's guarantees hold for a class of trees that arise naturally in sensor networks. Previous tree-based frequent items algorithms [8, 12, 18] provided only a weak bound of $O(\frac{m}{\epsilon} \log m)$ on total communication, even for the simplified case of balanced, regular trees.

6.1.1 Solution Approach and Challenges

A useful data structure encapsulating the partial result sent by a node X to its parent is a *summary*, defined as follows. A *summary* $\mathcal{S} = \langle N, \epsilon, \{(u, \tilde{c}(u))\}\rangle$ includes a (possibly empty) set of items u and their estimates $\tilde{c}(u)$. Each estimate $\tilde{c}(u)$ satisfies $\max\{0, c(u) - \epsilon \cdot N\} \leq \tilde{c}(u) \leq c(u)$, where $N = \sum_u c(u)$ such that $c(u)$ is the frequency of item u in the (multi-set) union of the multi-sets belonging to nodes in the subtree rooted at X. The salient property of a summary is that items with frequency at most $\epsilon \cdot N$ need not be stored, resulting in a smaller-sized summary and therefore, less communication.

[5]We adopt the standard convention that a *word* holds one item or one counter.

Algorithm 1: Generate a $\epsilon(k)$-summary (executed by all nodes, where k is the height of the node)

Input: summaries $\mathcal{S}_j = \langle n_j, \epsilon_j, \{(u, \tilde{c}_j(u))\}\rangle$ from each child j among the node's children $\mathcal{C}$, and its own summary $\mathcal{S}' = \langle n', 0, \{(u, \tilde{c}'(u))\}\rangle$
Output: single $\epsilon(k)$-summary $\mathcal{S} = \langle n, \epsilon(k), \{(u, \tilde{c}(u))\}\rangle$

```
1. Set $n := \sum_{j \in \mathcal{C}} n_j + n'$
2. For each $u \in \bigcup_{j \in \mathcal{C}} \mathcal{S}_j \cup \mathcal{S}'$, set $\tilde{c}(u) := \sum_{j \in \mathcal{C}} \tilde{c}_j(u) + \tilde{c}'(u)$
3. For each $u \in \mathcal{S}$, set $\tilde{c}(u) := \tilde{c}(u) - (\epsilon(k) \cdot n - \sum_{j \in \mathcal{C}} \epsilon_j \cdot n_j)$
   and if $\tilde{c}(u) \leq 0$ remove $(u, \tilde{c}(u))$ from $\mathcal{S}$
```

Our approach (similar to [8, 12]) is to distribute the ϵ error tolerance among intermediate nodes in the tree. We make the error tolerance a function of the *height* of a node, which is defined recursively as follows: the height of a leaf node is 1; the height of any other node is one more than the maximum height of its children. Let $\epsilon(i)$ denote the error tolerance of a node with height i.

Algorithm 1 presents the steps to generate a summary for a node X of height k, for a generic setting of the $\epsilon(i)$'s. Proceeding level-by-level up the tree, each node uses Algorithm 1 to generate an $\epsilon(k)$-summary, until at last the base station outputs an $\epsilon(h)$-deficient count for each item, where h is the height of the base station. For correctness, we need $\epsilon(1) \leq \epsilon(2) \leq \ldots \leq \epsilon(h)$. As long as $\epsilon(h) \leq \epsilon$, the user-specified guarantee is met. The sequence $\epsilon(1), \ldots, \epsilon(h)$ is called the *precision gradient* [12], because the precision of the data gradually decreases as it traverses the aggregation tree.

Thus far the approach has been similar to [12]. The key new challenge we address is how to set a precision gradient that minimizes the worst case total communication and is not restricted to balanced, regular trees. Note that when minimizing the *maximum* load on a link, as considered in previous approaches [8, 12], it suffices to bound the load on each link. This is a *local* property: for a node of height k, Step 3 of Algorithm 1 implies that estimates for at most $\frac{1}{\epsilon(k)-\epsilon(k-1)}$ items will be present in the summary it sends on its outgoing link [12]. Total communication, however, is a *global* property, and requires minimizing a sum. Indeed, it is not obvious how to set the precision gradient to achieve optimal total communication, even for regular trees.

6.1.2 Solution: The MIN TOTAL-LOAD Algorithm

We provide intuition for our solution by first considering a balanced, regular tree of degree d. In such trees, the number of nodes at height k is a $\frac{1}{d^k}$-th fraction of the number at height 1. Moreover, as discussed above, the maximum number of counters sent by a node at height k is $\frac{1}{\epsilon(k)-\epsilon(k-1)}$. Thus the total communication from height k nodes is proportional to the product of $\frac{1}{\epsilon(k)-\epsilon(k-1)}$ and $\frac{1}{d^k}$. To minimize total communication, it is then necessary to have large differences $\epsilon(k) - \epsilon(k-1)$ when k is small, so that the total number of counters sent by the numerous height k nodes is kept small. On the other hand, large differences for smaller k's leave only small differences for larger k's, because the sum of all the differences is at most ϵ. Thus for larger k's, the $\frac{1}{\epsilon(k)-\epsilon(k-1)}$ term is much larger than for smaller k's. Hence, care must be taken to ensure that the total number of counters sent is kept small even for larger k's, by keeping the term from getting too large. Our solution MIN TOTAL-

	Example Tree T_e				Regular Tree T_2			
h	1	2	3	4	1	2	3	4
$h(i)$	37	10	6	1	8	4	2	1
$H(i)$	$\frac{37}{54}$	$\frac{47}{54}$	$\frac{53}{54}$	$\frac{54}{54}$	$\frac{8}{15}$	$\frac{12}{15}$	$\frac{14}{15}$	$\frac{15}{15}$

Table 2: Example of a 2-dominating tree.

LOAD balances the allocation by setting $\epsilon(1)$ to be a constant fraction of the final ϵ, and then making $\frac{1}{\epsilon(k)-\epsilon(k-1)}$ vary as c^k for some $c < d$ so that the total number of counters (the product of $\frac{1}{\epsilon(k)-\epsilon(k-1)}$ and $\frac{1}{d^k}$) that could possibly be sent by all nodes at height k still decreases with increasing k. To extend our solution to non-regular trees, we exploit the observation that the maximum possible total communication of a tree (with our solution) can only decrease if nodes in an (initially balanced, regular) tree are replaced by nodes of lesser height.

d-dominating tree. MIN TOTAL-LOAD is based on the notion of a d-dominating tree, which we define as follows. For any tree, let $H(i)$ denote the fraction of nodes having height at most i. Mathematically, $H(i) = \frac{1}{m}\sum_{j=1}^{i} h(j)$, where $h(j)$ denotes the number of nodes at height j and m denotes the total number of nodes. For any $d \geq 1$, we say that a tree is *d-dominating* if for any $i \geq 1$, $H(i) \geq \frac{d-1}{d}(1 + \frac{1}{d} + \ldots + \frac{1}{d^{i-1}})$. Note that for any tree, $h(i)$ is monotonically non-decreasing as i increases, so every tree is 1-dominating. From the definition of a d-dominating tree, it follows that a tree that is $(d + \delta)$-dominating is also d-dominating for any $\delta \geq 0$. Also, given any tree and a precision $\delta_1 > 0$, we can always find some d such that the tree is d-dominating, but not $(d + \delta_1)$-dominating. We refer to such a d as the *domination factor* for the tree.

A relatively straightforward induction proof yields the following result:

LEMMA 2. *A tree in which each internal node of height i has at least d children of height $i - 1$ is d-dominating.*

Example 2. Consider a tree T_e with height 4 and $h(i)$ values as shown in Table 2. The table also shows the $h(i)$ and the $H(i)$ value of a completely balanced, regular tree T_2 of height 4 and degree 2. As the table shows, for all i, $H(i)$ of T_e is at least $H(i)$ of T_2. From Lemma 2, we know that T_2 is 2-dominating. Therefore, the example tree is 2-dominating. Assuming the granularity of d is 0.05, it can be shown that T_e has a domination factor of 2 (i.e., T_e is not 2.05 dominating).

We are now ready to present the precision gradient settings for our MIN TOTAL-LOAD algorithm:

LEMMA 3. *For any d-dominating tree of m nodes, where $d > 1$, a precision gradient setting of $\epsilon(i) = \epsilon \cdot (1 - t)(1 + t + \ldots + t^{i-1})$ with $t = \frac{1}{\sqrt{d}}$ limits total communication to $(1 + \frac{2}{\sqrt{d}-1})\frac{m}{\epsilon}$.*

PROOF. In Step 3 of Algorithm 1, the frequency estimate of each item u is decremented by at least $n \cdot (\epsilon(i) - \epsilon(i - 1))$. Since $\sum_{u\in\mathcal{S}} \tilde{c}(u) \leq n$, frequency estimates for at most $\frac{1}{\epsilon(i)-\epsilon(i-1)}$ items are sent by a node at height i to its parent.

Furthermore, because the maximum number of possible estimates a node at height i can send on its outgoing link increases with its height ($\propto d^{i/2}$), the total communication for a d-dominating tree is bounded by a (hypothetical) tree in which exactly $\frac{d-1}{d}\frac{1}{d^{i-1}}$ fraction of the nodes occur at height i. Therefore, the total communication is bounded by:

$$\sum_{i=1}^{\infty} \frac{m(d-1)}{d(d^{i-1})} \frac{1}{(\epsilon(i) - \epsilon(i-1))} = \frac{m(d-1)}{\epsilon\sqrt{d}(\sqrt{d}-1)} \sum_{i=1}^{\infty} \frac{1}{d^{(i-1)/2}}$$

$$\leq \frac{m}{\epsilon}\frac{d-1}{(\sqrt{d}-1)^2} = \frac{m}{\epsilon}\frac{\sqrt{d}+1}{\sqrt{d}-1} = \frac{m}{\epsilon}\left(1 + \frac{2}{\sqrt{d}-1}\right) \square$$

OBSERVATION 2. *As the constant factor $(1+\frac{2}{\sqrt{d}-1})$ shows, total communication decreases as d increases. Therefore, it is desirable to have aggregation trees which are d-dominating for large d values.*

6.1.3 Trees with Large Domination Factors

As Lemma 3 shows, our MIN TOTAL-LOAD algorithm works especially well for trees that are d-dominating for high d values. The chief concern for guaranteeing low communication is to ensure that the domination factor d is sufficiently far from 1, say at least 2. In Section 7, we show that a real-world sensor deployment has a domination factor of 2.25, suggesting that while it may be infeasible to construct a *regular* tree in a sensor network, it may be easy to generate d-dominating trees for $d \geq 2$. Moreover, we show next an explicit tree construction algorithm that seeks to increase the domination factor.

We modify the standard tree construction algorithm (described in Section 2) with two optimizations. First, when a node in level i chooses its parent, and even when it switches parents, it selects a node only from level $i - 1$; the standard algorithm [10] allows choosing a parent from the same level. Second, we use the following *opportunistic parent switching* technique, inspired by Lemma 2. Each node of height $j + 1$ that has two or more children of height j, pins down any two of its height j children, so that they cannot switch parents, and then flags itself. Next, the non-pinned nodes in each level i switch parents randomly to any other reachable non-flagged node in level $i - 1$. As soon as a non-flagged node has at least two flagged children of the same height, it pins both of them and then flags itself. This local search technique quickly makes the tree 2-dominating if there is an opportunity to do so.

6.1.4 Extensions

Combining Objective Functions. Limiting total communication places an upper bound on the energy usage of all the sensors. However, in some cases, limiting the maximum load on a link is also very important [8, 12, 18]. We now show how to obtain a precision gradient setting that simultaneously achieves *both* optimal (within constant factors) maximum and optimal (within constant factors) total communication, by combining solutions that are optimal for each of the two individual metrics. Our combination technique can easily be generalized to multiple linear communication metrics, i.e., the communication metric is a weighted sum, weighted min, or weighted max of the communication load on the tree links. The following two observations are the key insights that lead to our technique.

OBSERVATION 3. *If each $\epsilon(i)$ in a precision gradient setting is divided by some constant $c \geq 1$, then the worst case value of any linear communication metric is multiplied by c.*

OBSERVATION 4. *For some $k \in [1, h]$, if the precision gradient setting is changed so that $\epsilon(i)$ is increased for $i \geq k$ and left unchanged otherwise, then the value of any linear communication metric does not increase.*

LEMMA 4. *Let $\epsilon^a = (\epsilon(1)^a, \ldots, \epsilon(h)^a)$ and $\epsilon^b = (\epsilon(1)^b, \ldots, \epsilon(h)^b)$ denote the precision gradient settings that respectively minimize the* total communication *and the* maximum load on any link. *Then $\epsilon^* = (\epsilon(1)^*, \ldots, \epsilon(h)^*)$ where $\epsilon(i)^* = \frac{1}{2}(\epsilon(i)^a + \epsilon(i)^b)$ gives a simultaneous 2-approximation for* total communication *and* maximum load on any link.

PROOF. Follows from Observation 3 and Observation 4. □

We call this variant of our algorithm HYBRID, because its objective function includes both maximum and total communication.

Computing Quantiles. The quantiles algorithm by Greenwald and Khanna [8] can be extended to use our precision gradients and hence to achieve useful bounds. For example, our techniques above for bounding total communication to $O(\frac{m}{\epsilon})$ words and for simultaneously bounding multiple communication metrics can be easily extended to the problem of *finding quantiles*. As such, they are the first quantiles algorithms that achieve these bounds. We formally state the technique analogous to MIN TOTAL-LOAD for computing quantiles as Corollary 1.

COROLLARY 1. *If the quantiles algorithm provided by Greenwald and Khanna [8] is extended to use precision gradient, for any d-dominating tree of m nodes, where $d > 1$, a precision gradient setting of $\epsilon(i) = \epsilon \cdot (1-t)(1+t+\ldots+t^{i-1})$ with $t = \frac{1}{\sqrt{d}}$ permits the computation of quantiles with total communication at most $(1 + \frac{2}{\sqrt{d}-1})\frac{m}{\epsilon}$.*

6.2 Multi-path Algorithm

We are aware of two previous multi-path algorithms for computing frequent items. The first algorithm, presented in [15], performs multi-path counting of the items in the network and keeps track of the items with high count values. Since the multi-path counting algorithm is approximate, it can find the frequent items only if they appear *significantly* more than the non-frequent items. The second algorithm computes a uniform sample of the items (which can be computed over multi-path as shown in [15]) and then estimates the frequent items over that sample. The accuracy of this algorithm depends on the sample size and the data distribution. As we show in Section 7, these two algorithms suffer from relatively high false positive and false negative rates under a real sensor dataset.We address this problem with a new multi-path algorithm for finding frequent items with high accuracy.

Ideally, we would like to base our algorithm on Algorithm 1, adapting it to be duplicate-insensitive so that it works correctly in the multi-path setting. Steps 1 and 2 of Algorithm 1 are readily adapted: simply replace the addition operator in those steps with a known *duplicate-insensitive* addition operator (which we will denote $\oplus$). Step 3 is more problematic, however, because it uses a subtraction operator and no duplicate-insensitive subtraction algorithms exist that combine high accuracy with small synopses.

Overcoming the Duplicate-Insensitive Subtraction Problem. Most duplicate-insensitive *addition* algorithms

Algorithm 2: Synopsis fusion function

Inputs: synopses $\mathcal{S}_1 = \langle \tilde{n}_1, \{(u, \tilde{c}_1(u))\}\rangle$ and $\mathcal{S}_2 = \langle \tilde{n}_2, \{(u, \tilde{c}_2(u))\}\rangle$, of class i

Output: synopsis $\mathcal{S} = \langle \tilde{n}, \{(u, \tilde{c}(u))\}\rangle$, of either class i or $i+1$

```
1. Set $\tilde{n} := \tilde{n}_1 \oplus \tilde{n}_2$     ($\oplus$ is duplicate-insensitive sum)
2. For each item $u \in (\mathcal{S}_1 \cup \mathcal{S}_2)$, add $(u, \tilde{c}_1(u) \oplus \tilde{c}_2(u))$ to $\mathcal{S}$
3. If $\tilde{n} > 2^{i+1}$, then increment the class of $\mathcal{S}$ from $i$ to $i+1$
   and for each item $u \in (\mathcal{S}_1 \cup \mathcal{S}_2)$:
   if $\frac{\epsilon \cdot \tilde{n}}{\log N} \geq \eta \cdot \tilde{c}(u)$, drop $(u, \tilde{c}(u))$ from $\mathcal{S}$ (We restrict $\eta > 1$)
```

(for positive numbers) guarantee that the error in their estimates is at most a constant factor of the actual value. Specifically, for a user-specified relative error ϵ_c, $0 < \epsilon_c < 1$, and confidence parameter δ_c, $0 < \delta_c < 1$, the estimate is within a relative error ϵ_c of the actual sum with probability at least $1 - \delta_c$. Because each added item increases the actual sum, the allowed *absolute* error increases. If subtraction were allowed, the actual value would decrease and it is not known how to achieve a corresponding reduction in error while preserving a small synopsis.

Our solution is to avoid using subtraction altogether. First, we note that the primary purpose of the subtraction in Step 3 is to enable items with small estimated counts to be dropped from the summary. Instead of subtracting at each node and dropping the item if its estimate is negative, we eliminate the subtraction and drop the item if its estimate is below a rising threshold.

Second, we observe that we do not need highly accurate duplicate-insensitive addition in our thresholding approach. If we give ourselves some slack on the threshold, then we can tolerate less accurate addition and will not drop items we should not have dropped. We may fail to drop some items that should be dropped, but this does not change the asymptotic communication bounds. The importance of making use of less accurate addition arises from the fact that known duplicate-insensitive addition algorithms [3, 4, 5, 15] require synopses whose size is proportional to $\frac{1}{\epsilon_c^2}$. When $\epsilon_c = \frac{1}{2}$, this equals 4; when $\epsilon_c = \frac{1}{100}$, this equals 10,000.

Finally, our algorithm adapts from [8] the concept of *classes*. A synopsis is in class i if i is the logarithm of the number of items it represents. The idea is to have the error tolerance ϵ of a synopsis vary linearly with its class number and only combine synopses having the same class. Assuming there are no duplicates, doing so ensures that there is always an opportunity for further pruning once any two synopses are combined. Therefore, a synopsis never becomes too large. Even with duplicates, as our analysis later shows, the size of a synopsis does not grow beyond a constant factor of the case when there are no duplicates. Also, because the count for the total number of items represented by a synopsis is approximate (because $\oplus$ is used), this approximate count determines the class of a synopsis. As long as the relative error ϵ_c for $\oplus$ is less than 1, there are at most $\log N + 1$ classes of synopses varying from $i = 0, \ldots, \log N$ (N denotes the total number of items).

We now describe the components of our multi-path algorithm, using the terminology presented in Section 2.

Synopsis Generation. Each sensor node processes its collection of items by counting item frequencies and discarding (pruning) all items u whose frequency is at most $\frac{in'\epsilon}{\log N}$, where n' is the total number of items in its collection and

$i = \lfloor \log n' \rfloor$. Then the node generates a class i synopsis by using a duplicate-insensitive addition algorithm to compute a frequency estimate of each remaining item. If the node is a leaf node, it forwards the synopsis to its parent.

Synopsis Fusion. Each intermediate node, in general, receives from each of its children at most a single synopsis of each class. After receiving all synopses from its children, beginning with the smallest class for which the node has a synopsis, the node starts combining two synopses of the same class using Algorithm 2, until it is left with at most one synopsis of each class. It then transmits the resulting collection of synopses to its parent. The parameter η controls the accepted slack in the thresholding procedure, as a function of the error parameter ϵ_c.

Synopsis Evaluation. Finally, at the base station, the frequency estimates corresponding to an item are simply added (again using $\oplus$) across all the different $\log N$ classes.

Accuracy and Communication Bounds. The accuracy and communication bounds of our algorithm are summarized in the following theorem. The bounds are in terms of the number of sensors m, the total number of items N, the user-specified error tolerance ϵ and confidence parameter δ, and the relative error parameter ϵ_c of $\oplus$, where ϵ, δ, and ϵ_c are each between 0 and 1. In addition, the bound assumes that the duplicate-insensitive sum operator $\oplus$ is also accuracy-preserving, as defined next.

DEFINITION 1. *Suppose $X_{(\epsilon_c,\delta_c)}$ denotes an (ϵ_c, δ_c)-estimate of a scalar X, with a relative error of ϵ_c, and a confidence parameter of δ_c. Then, the operator $\oplus$ is called an accuracy preserving duplicate-insensitive sum operator if $X_{(\epsilon_c,\delta_c)} \oplus Y_{(\epsilon_c,\delta_c)} = Z_{(\epsilon_c,\delta_c)}$, where $Z = X + Y$.*

An example of an accuracy preserving duplicate-insensitive sum operator can be found in [3].

THEOREM 1. *When $\oplus$ is an accuracy preserving duplicate-insensitive sum operator, then with probability at least $1-\delta$, for all items u, the algorithm produces estimated frequencies $\tilde{c}(u)$ such that $(1-\epsilon_c)(c(u) - \epsilon N) \leq \tilde{c}(u) \leq (1+\epsilon_c)c(u)$. Moreover, the maximum load on a link is $O(\frac{\log^2 N}{\epsilon} \cdot \frac{1}{\epsilon_c^2} \cdot \log(\frac{m \log N}{\epsilon\delta}))$ memory words.*

The proof is provided in Appendix A.

6.3 Tributary-Delta Algorithm

Recall from the discussion in Section 5 that to combine tree and multi-path algorithms, we need a conversion function that takes a partial result generated by the tree algorithm and outputs a synopsis that can be used by the multi-path algorithm. For frequent items, when using our tree and multi-path algorithms provided in Section 6.1 and Section 6.2 respectively, this conversion function can simply be the synopsis generation (SG) function of our multi-path algorithm, applied to the estimated frequencies of the tree algorithm. Specifically, we take the summary $\langle n, \epsilon(k), \{(u, \tilde{c}(u))\}\rangle$ from Algorithm 1 and view $\tilde{c}(u)$ as the actual frequency to which we apply the thresholding test of SG (Section 6.2), letting the n' in SG be the n from the summary, in order to create the synopsis. In this way, the final error in estimating the frequency of an item is at most the sum of the errors in the tree and the multi-path algorithm. So, given a user-specified error ϵ, we can obtain (approximate) ϵ-deficient counts in the Tributary-Delta framework, by running our tree-algorithm with error tolerance ϵ_a (where $\epsilon(k) \leq \epsilon_a$) and our multi-path algorithm with error tolerance ϵ_b, such that $\epsilon_a + \epsilon_b = \epsilon$.

7. EXPERIMENTAL RESULTS

In this section we evaluate our two proposed Tributary-Delta approaches, `TD-Coarse` and `TD`, in varying network conditions, using existing tree and multi-path approaches as baselines. We begin in Section 7.1 by describing the real-world data and the simulation environment we used. Then, in Section 7.2, we show the different ways in which `TD-Coarse` and `TD` adapt to changes in network conditions. Using a simple aggregate (Sum), we report the error reductions due to our proposed approaches over the baseline approaches in Section 7.3. Then, in Section 7.4, we provide measurements of communication load for our frequent items algorithm MIN TOTAL-LOAD in both tree and multi-path scenarios, and evaluate `TD-Coarse` and `TD` for this more complex aggregate. Finally, in Section 7.5, we summarize the results.

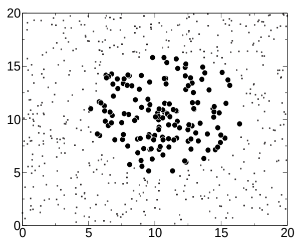
7.1 Methodology

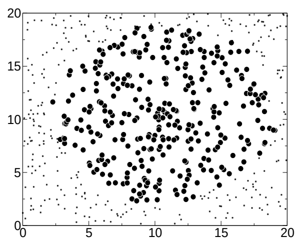
We implement `TAG` [10], `SD` [15] and our proposed Tributary-Delta approaches, `TD-Coarse` and `TD`, within the TAG simulator [10]. `TAG` is a tree-based aggregation approach used by TinyDB [10], whereas `SD` (for Synopsis Diffusion) is a multi-path aggregation approach over Rings [15]. In each simulation run, we use Sum as the aggregate collecting an aggregate value every epoch for 100 epochs, unless noted otherwise. We begin data collection only after the underlying aggregation topologies become stable. In all the experiments, we use a variant of [7] (as in [5]) for achieving duplicate-insensitive addition. We allow the Tributary-Delta approaches to adapt their topologies every 10 epochs. Recall from Section 4 that the adaptivity decisions in our proposed approaches are guided by a threshold on the percentage of nodes contributing to the aggregate. We use 90% as the threshold. We use 48-byte messages, as used by the TinyDB system. This allows us to fit 40 32-bit `Sum` synopses (with the help of run-length encoding [16]) within a single message, and produce an approximate Sum based on the average of these 40 estimates.

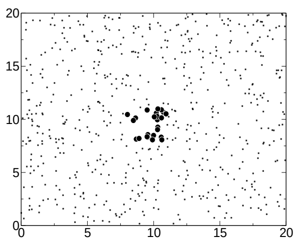
Scenarios. We use LABDATA, a scenario reconstructing a real deployment, and SYNTHETIC, a synthetic scenario with several failure models, for our experiments. Using actual sensor locations and knowledge of communication loss rates among sensors, LABDATA simulates a deployment of 54 sensors recording light conditions in the Intel Research Berkeley laboratory [9]. The dataset contains around 2.3 million sensor readings. The SYNTHETIC scenario is a deployment of 600 sensors placed randomly in a 20 *ft* $\times$ 20 *ft* grid, with a base station at location $(10, 10)$. We study two failure models for SYNTHETIC: GLOBAL(p), in which *all* nodes have a message loss rate of p, and REGIONAL(p_1, p_2), in which all nodes within the rectangular region $\{(0, 0), (10, 10)\}$ of the 20×20 deployment area experience a message loss rate of p_1 while other nodes have a message loss rate of p_2.

7.2 Adaptivity of `TD-Coarse` and `TD`

To demonstrate the different ways in which our two strategies adapt to changes in network conditions, we study the SYNTHETIC scenario under the two failure models. First, we apply the GLOBAL(p) failure model with increasing values


(a) TD-Coarse & Global(0.2)


(b) TD-Coarse & Global(0.3)


(c) TD & Regional(0.3,0.05)

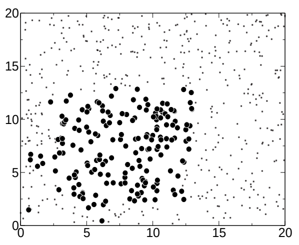

(d) TD & Regional(0.8,0.05)

Figure 4: Evolution of the TD-Coarse and TD topologies for varying loss rates. Each dot depicts a sensor located at the given coordinates in the deployment area. The larger dots comprise the delta region. The base station is at $(10, 10)$.

of p. Figure 4(a) and Figure 4(b) show snapshots of the TD-Coarse approach when the loss rates are 0.2 and 0.3 respectively. As expected, the delta region expands as the loss rate p increases—depicted by an increase in the number of larger dots from Figure 4(a) to Figure 4(b). The snapshots for the TD approach are similar, except that the delta region increases gradually, instead of expanding by all switchable nodes at a time.

Second, we apply the Regional(p_1, p_2) failure model with increasing p_1 and a fixed $p_2 = 0.05$. In TD-Coarse, because the delta region expands uniformly around the base station, all nodes near the base station are switched to multi-path, even those experiencing small message loss. In TD, this problem does not arise because the delta region expands only in the direction of the failure region. Figure 4(c) and Figure 4(d) capture pictorially the response of TD to such localized failures. Even at a high loss rate, in TD, the delta region mostly consists of nodes actually experiencing high loss rate.

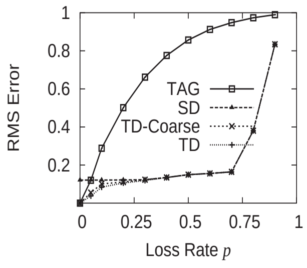
7.3 Evaluation using a Simple Aggregate

In this section we evaluate the error reduction of our two proposed approaches, TD-Coarse and TD, in varying network conditions, using the TAG and SD approaches as baselines. We restrict ourselves to simple aggregates like Count and Sum for which the partial results can fit in a single TinyDB packet for both TAG and SD. To ensure that all approaches use comparable energy levels, we disallow retransmissions (as in the original TinyDB implementation).

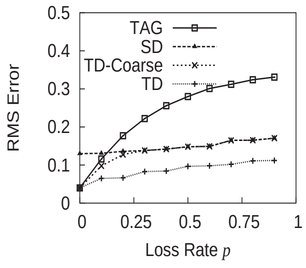
We measure the error as the relative root mean square (RMS) error—defined as $\frac{1}{V}\sqrt{\sum_{t=1}^{T}(V_t - V)^2/T}$, where V is the actual value and V_t is the aggregate computed at time t. The closer this value is to zero the closer the aggregate is to the actual value.

Real scenario. We find the RMS error in evaluating the Sum aggregate on LabData to be 0.5 for TAG and 0.12 for SD. Both TD and TD-Coarse are able to reduce the error to 0.1 by running synopsis diffusion over most of the nodes.

Synthetic scenarios. For the remainder of this section, we use Synthetic scenarios. Figure 5(a), the complete graph for Figure 2, presents the RMS error of different schemes under the Global(p) failure model. At *all* loss rates in both cases, the error for either TD-Coarse or TD is no worse than the minimum of TAG or SD. In particular, the error is reduced significantly at low loss rates $(0 \leq p \leq 0.05)$, when some tree nodes can *directly* provide exact aggregates to the base station. This effect is more pronounced in Figure 5(b) with



(a) Global(p) failure



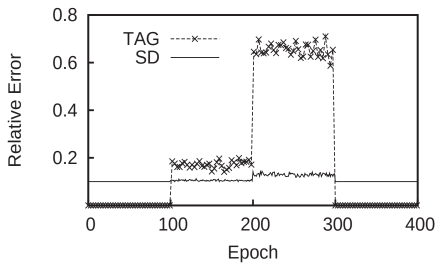
(b) Regional$(p, 0.05)$ failure

Figure 5: RMS errors and loss rates.

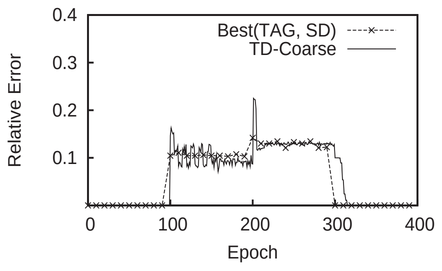
the TD strategy under a Regional$(p, 0.05)$ failure model. TD uses multi-path aggregation only in the failure region and so exact aggregation over a significant portion of nodes can be carried out using tree aggregation.

Next, to evaluate how well our Tributary-Delta schemes adapt to dynamic scenarios, starting with the Global(0) failure model, we first introduce Regional$(0.3, 0)$ at time $t = 100$. Then at $t = 200$, we switch to Global(0.3). Finally, at $t = 300$, we restore the Global(0) failure model. Figure 6 shows the relative errors of the answers provided by different schemes over time. We use relative error instead of RMS error because each data point corresponds to just a single aggregate answer.

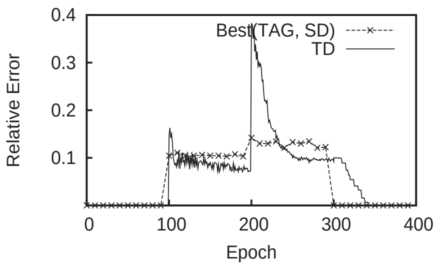
As expected (Figure 6(a)), TAG is more accurate when loss rates are low ($t \in [0, 100]$ or $t \in [300, 400]$) whereas SD is more accurate when loss rates are high ($t \in [100, 300]$). Figure 6(b) and Figure 6(c) compare the relative errors of TD-Coarse and TD with the smallest of the errors given by TAG and SD. At a high level, both TD-Coarse and TD, when converged, have at most the error given by any of the two existing approaches. However, the graphs reveal a number of subtle differences between the two Tributary-Delta schemes. First, because TD can adjust its delta region at a finer granularity, it can converge to provide a more accurate result. Second, the coarse granularity of TD-Coarse adversely affects its convergence: the delta region continues to expand and shrink around the optimal point (e.g., for $t \in [100, 150]$ in Figure 6(b)). The base station can use simple heuristics to stop the oscillation (e.g., at $t = 150$), but even then it may end up using a delta region larger than necessary. Finally, the benefits of TD come at the cost of a higher convergence time compared to TD-Coarse. As shown in Figure 6(c), TD takes around 50 epochs to converge after the network condition changes. The time can be reduced by carefully choosing some parameters (e.g., how often the topology is adapted),



(a) `TAG` and `SD`



(b) `TD-Coarse`



(c) `TD`

Figure 6: Timeline showing the relative errors of different aggregation schemes.

a full exploration of which is beyond the scope of this paper.

7.4 Evaluation With Frequent Items

We begin in Section 7.4.1 by evaluating our tree construction algorithm presented in Section 6.1.3. We then evaluate our frequent items algorithms using a SYNTHETIC scenario with the brightness data stream of the LABDATA dataset evenly partitioned among the sensors. The real-valued sensor data are discretized with a bucket-size of 5 and frequent items are computed over these discrete values. For the different versions of our frequent item algorithm, we use error margin $\epsilon = 0.1\%$ to report all items with frequency more than the *support threshold* $s = 1\%$ of the total number of item occurrences, similar to the methodology in [13, 12]. Likewise, to compensate for the elimination of small partial counts at intermediate nodes in our tree (Step 3 of Algorithm 1) as well as our multi-path algorithm (Step 3 of Algorithm 2) for computing frequent items, we report all items whose estimated counts are more than $(s - \epsilon)$ fraction of the total count. We present evaluation of MIN TOTAL-LOAD, our tree algorithm for computing frequent items, in Section 7.4.2. Then, in Section 7.4.3, we compare our multi-path algorithm against the two existing algorithms described in Section 6.2. Finally, in Section 7.4.4, we evaluate `TD-Coarse` and `TD` for the frequent items aggregate.

7.4.1 Evaluation of Tree Construction Algorithm

As mentioned in Section 6.1.3, our MIN TOTAL-LOAD algorithm has smaller overhead (by constant factors) if the aggregation tree is d-dominating for large d values. We note that, in practice, typical sensor deployments tend to have this property. For example, Figure 7(a) plots the cumulative number of nodes versus height for the aggregation tree of the LABDATA dataset. The distribution is lower bounded by the cumulative number of nodes of a balanced, regular tree with degree 2.25 (denoted as Regular(2.25)). Hence the aggregation tree of the LABDATA dataset is 2.25-dominating, resulting in a low overhead (the constant factor of $(1+\frac{2}{\sqrt{d-1}})$ evaluates to 5).

Figure 7(b) and Figure 7(c) show the domination factors of the aggregation trees in different SYNTHETIC scenarios. In Figure 7(b), while keeping the deployment area fixed at 20×20, we vary the sensor density. In Figure 7(c), we keep the sensor density fixed (1 sensor per square unit area) and vary the size of the deployment area by changing its width (the height remains 20 across all experiments). The graphs show that our tree construction algorithm (Section 6.1.3) significantly improves the domination factor d. This is particularly useful when the domination factor of the tree is low

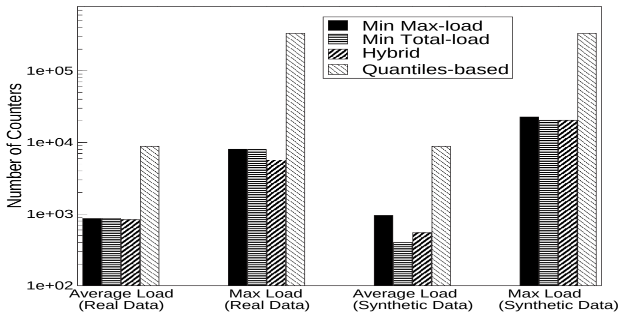


Figure 8: Average and maximum load of a sensor in a tree topology. (Note the log-scale on the y-axis.)

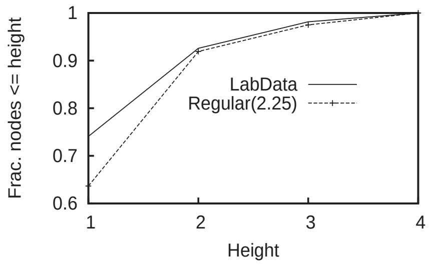
because of low sensor density or narrow deployment area, since even a slight improvement in the d value greatly reduces the constant factor in MIN TOTAL-LOAD (which is proportional to $\frac{1}{\sqrt{d-1}}$).

7.4.2 Frequent Items over Tree

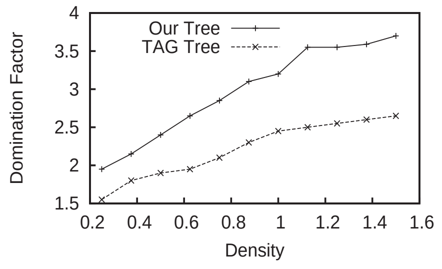
Figure 8 compares our two frequent items algorithms MIN TOTAL-LOAD (Section 6.1.2) and HYBRID, the 2-approximation algorithm presented in Section 6.1.4, against the two best known existing algorithms: MIN MAX-LOAD [12] and QUANTILES-BASED[6] [8]. We report the average and maximum load (number of integer values transmitted) of a node, under no message loss, on two sets of data. The first two sets of bars in the graph represent the results with the LABDATA dataset. The graph shows that even though the communication load of our HYBRID algorithm is only guaranteed to be within a factor of 2 of the best of MIN TOTAL-LOAD and MIN MAX-LOAD, with our real-world data set it performs significantly better. Specifically, HYBRID is 4% and 30% *better* than the best of MIN MAX-LOAD and MIN TOTAL-LOAD for average load and maximum load, respectively. It also shows that even though our MIN TOTAL-LOAD algorithm does not aim to reduce maximum load of a node, in practice it performs very close to MIN MAX-LOAD. The QUANTILES-BASED algorithm performs significantly worse than the other algorithms because it is not optimized for the bushy tree we encounter in LABDATA.

As discussed in Section 6.1, the average load for our MIN TOTAL-LOAD and HYBRID algorithms is upper bounded by a constant for any dataset (recall Lemma 3 and Lemma 4 noting that the average load is the total communication divided by the number of nodes), whereas it is logarithmic in the number of nodes for MIN MAX-LOAD and QUANTILES-

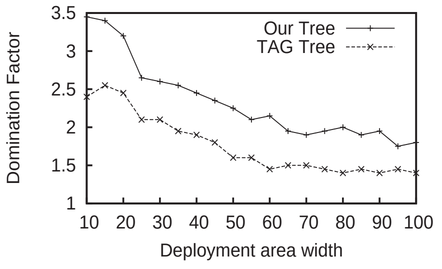
[6]Frequent items can be computed from quantiles.



(a) LabData



(b) Effect of density



(c) Effect of deployment area shape

Figure 7: Domination factors in real and synthetic aggregation trees.

Algorithm	False positive	False negative
Undercounting-based	2.3%	0
Counting-based	33.3%	30.3%
Sampling-based	15.4%	12.1%

Table 3: Average false positives and false negatives in the frequent items estimated by different multi-path algorithms.

based. Therefore, we can always construct a stream of items for which the improvement of Min Total-load and Hybrid over Min Max-load and Quantiles-based is significant (e.g., zero vs. nonzero). As an example, in the last two sets of bars in Figure 8, we present results over a synthetic dataset, in which every sensor node receives a stream of items such that (1) the same item never occurs in multiple streams and (2) within a stream the items are uniformly distributed. As shown, Min Total-load incurs only half the total communication required for the Min Max-load algorithm.

7.4.3 Frequent Items over multi-path

To understand the performance of our multi-path based frequent items algorithm presented in Section 6.2, which we denote as Undercounting-based (Step 3 of Algorithm 2 does the undercounting), we compare it with the two existing algorithms we summarized in the section. We denote the first algorithm as Counting-based and the second algorithm as Sampling-based. Since neither Counting-based nor Sampling-based have provable error guarantees, we likewise relax guarantees for our multi-path algorithm, in order to keep message overheads low. In particular, instead of using an accuracy preserving duplicate-insensitive sum operator (Definition 1), which would be necessary for providing error guarantees, we continue using the low overhead, best-effort algorithm in [7] for duplicate-insensitivity. To keep communication costs same, we use a sample size of 1000 for Sampling-based. Finally, we measure the accuracy of the algorithms with their *false positive* (number of non-frequent items reported as frequent) and *false negative* (number of frequent items not reported) rates.

Table 3 shows the false positive and the false negative rates in the frequent items estimated by different algorithms. Each data point represents the average error over 100 experiments (each experiment has a random placement of sensors, and hence potentially a different topology). As shown, our Undercounting-based algorithm has no false negative and has only a very small false positive rate. The zero false negatives and a few false positives arise because (a) we report items above $s-\epsilon$ fraction, thus sufficiently compensating for the elimination of small partial counts at intermediate nodes (Step 3 of Algorithm 2), and (b) the estimates from the best-effort algorithm in [7] for duplicate-insensitivity are usually higher than the actual value. Counting-based suffers from large false positive and false negative rates since the frequent items do not appear significantly more often than the non-frequent items, making the input a difficult one for Counting-based. The false positive and the false negative rates of Sampling-based are higher than Undercounting-based, suggesting that for this experiment, a sample size of 1000 is not sufficient for Sampling-based to match the accuracy of our Undercounting-based approach.

7.4.4 Frequent Items over Tributary-Delta

We now evaluate how well our Tributary-Delta algorithm finds frequent items. Recall from Section 6.3 that our Tributary-Delta algorithm for finding frequent items combines a tree and a multi-path algorithm. For the tree part, we use our Min Total-load algorithm, and for the multi-path part, we use our Undercounting-based algorithm. Further, our Tributary-Delta algorithm requires setting of ϵ_a, the error margin in the tree part, and ϵ_b, the error margin in the multi-path part, so that $\epsilon_a + \epsilon_b = \epsilon$. We divide $\epsilon = 0.1\%$ in the most natural way— equally between ϵ_a and ϵ_b, and leave the task of determining the best division as future work. As before, we continue reporting items above $s - \epsilon$ fraction, thus sufficiently compensating for the elimination of small partial counts at intermediate nodes in both the tree and multi-path part.

Because communication failures mean that the tree part may not meet its error margin guarantee, we likewise relax the multi-path part's error guarantees, in order to reduce message overheads. In particular, as in the previous section, we continue using the low overhead, best-effort algorithm in [7]. As mentioned in the previous section, our multi-path algorithm for computing frequent items (Undercounting-based) has a small ($< 3\%$) false positive rate with no communication failure. Communication failures further reduce the false positive rate, but introduce false negatives in the estimated results because some of the items with frequency above the support threshold s are not reported due to underestimation resulting from message loss, mostly in the tree part.

Figure 9(a) and Figure 9(b) show the false negative rates of different aggregation schemes under the Global(p) and

REGIONAL$(p, 0.05)$ failure models, respectively.[7] As in our previous results, `TD` performs as well as (for GLOBAL) or better than (for REGIONAL) the `TAG` or `SD` schemes alone.

In contrast to the Sum or Count aggregate studied in Section 7.3, with the Frequent Items aggregate, a multi-path partial result can consist of more TinyDB messages than a tree partial result (3 times on average in this experiment). To make both approaches use comparable energy while keeping the latency, which increases linearly with the number of retransmissions, acceptable, we let the tree nodes retransmit their messages twice.[8] Keeping the latency low is particularly important for many real-time monitoring and control applications [19]. The results are shown in Figure 9(c). As expected, retransmission significantly reduces the false negatives of `TAG`. Still, at loss rates greater than 0.5, the multi-path algorithm outperforms the tree algorithm and `TD` can effectively combine the benefits of both the algorithms.

7.5 Summary of Results

Our results have quantified a number of advantages of our techniques. First, we have shown that our `TD-Coarse` and `TD` constructions can run tree and multi-path based aggregation simultaneously in different parts of the network and can dynamically balance between the tree and multi-path components as the operating conditions (e.g., loss rate) change. Second, Tributary-Delta is able to provide not just the best of the accuracies provided by tree or multi-path (e.g., by running either on them in the whole network), but in fact a significant accuracy improvement over the best, across a wide range of practical loss rates $(0-40\%)$—thus demonstrating the synergies of using both in tandem. For example, in computing Count under typical loss rates $(0-40\%)$, Tributary-Delta reduces errors by up to a factor of 3 compared to the best existing approach for that rate. Third, for complex aggregates (e.g., Frequent Items), Tributary-Delta may use larger messages than a tree-based algorithm. However, Tributary-Delta can provide higher accuracy than a tree that retransmits messages on loss and hence consumes the same amount of communication energy.[9] For example, with our frequent items algorithm and within the loss rates $0-40\%$, Tributary-Delta reduces false-negatives by a factor of 4 compared to the best of `TAG` and `SD`. Thus, Tributary-Delta provides a sweet spot between communication overhead and approximation error, as shown in Table 1. Finally, we have shown that our tree-based and multi-path-based Frequent Items algorithms perform significantly better than the existing algorithms.

[7]Note that in the extreme when all nodes run either tree or multi-path algorithm, our choice of choosing ϵ_a and ϵ_b so that $\epsilon_a + \epsilon_b = \epsilon$ holds, and then reporting all items above frequency greater than $(s - \epsilon)$ fraction, reduces the number of false negatives. However, we obtain similar results, which we present in the conference version of this paper, even with the setting $\epsilon_a = \epsilon_b = \epsilon$.

[8]Note that, in practice, two retransmissions would incur more latency than a single transmission of a 3 times longer message, because each retransmission occurs after waiting for the intended receiver's acknowledgment. Other limitations of retransmission include a reduction in channel capacity (by $\approx 25\%$) and the need for bi-directional communication channels (often not available in practice) [22].

[9]Note that a tree with retransmission also has a higher response time than a Tributary-Delta without retransmission.

8. CONCLUSION

In this paper, we have presented Tributary-Delta, a novel energy-efficient approach to in-network aggregation in sensor networks. Tributary-Delta combines the advantages of the existing tree- and multi-path-based approaches by running them simultaneously in different parts of the network. We have studied this new approach and presented schemes for adjusting the balance between tributaries and deltas in response to changes in network conditions. We have also shown how a difficult aggregate for this context—finding frequent items—can be efficiently computed in the Tributary-Delta framework. Our simulation results on real-world and synthetic data showed that our techniques are greatly superior to the existing tree- and multi-path-based approaches. For example, in computing Count under realistic loss rates, our techniques can reduce errors by up to a factor of 3 compared to *any* previous technique.

9. REFERENCES

[1] Atmel AVR Microcontroller Datasheet. http://www.atmel.com/dyn/resources/prod_documents/2467s.pdf.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

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

[4] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *SIGMOD*, 2004.

[5] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

[6] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

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

[8] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, 2004.

[9] Intel Lab Data. http://berkeley.intel-research.net/labdata/.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad hoc sensor networks. In *OSDI*, 2002.

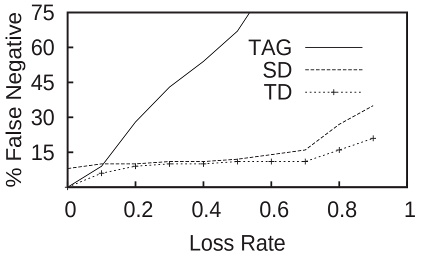
[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisational query processor for sensor networks. In *SIGMOD*, 2003.

[12] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, 2005.

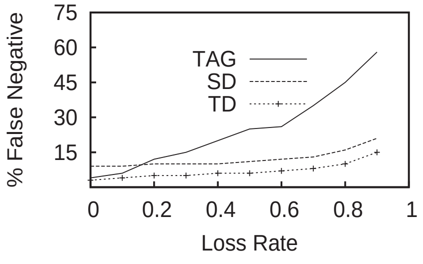
[13] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

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

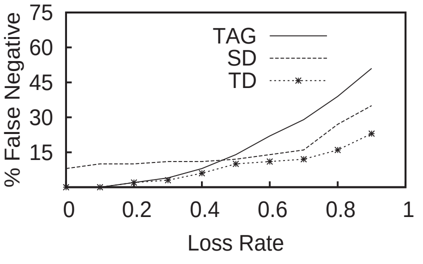
[15] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys*, 2004.



(a) Global loss



(b) Regional loss



(c) Global loss & retransmission

Figure 9: False negatives in the estimated frequent items. (False positives are $< 3\%$.)

[16] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *SIGKDD*, 2002.

[17] V. Shnayder, M. Hempstead, B.-R. Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys*, 2004.

[18] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *SenSys*, 2004.

[19] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *SenSys*, 2004.

[20] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-temporal aggegration using sketches. In *ICDE*, 2004.

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

[22] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys*, 2003.

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

APPENDIX

A. PROOF OF THEOREM 1

Before stating and proving Theorem 1, we provide bounds on the accuracy of the synopsis generated at any node (Lemma 5), the accuracy of the synopses resulting from Synopsis Fusion (Lemma 6, Lemma 7 and Lemma 8), and bounds on the communication load on a link (Lemma 9, Lemma 10 and Lemma 11). Recall that all bounds are in terms of the number of sensors m, the total number of items N, the overall user-specified error tolerance ϵ and confidence parameter δ, and the relative error parameter ϵ_c of $\oplus$, where ϵ, δ, and ϵ_c are each between 0 and 1, and $\oplus$ is the constant error duplicate-insensitive sum operator defined in Definition 1.

We start with Lemma 5 which bounds the accuracy of the synopses generated by the nodes.

Lemma 5. *With probability at least $1-\delta$, for any synopsis $\mathcal{S}$ generated at the nodes, and for any item u in $\mathcal{S}$, the following bounds hold:*

- $(1-\epsilon_c)\mathcal{S}.n \le \mathcal{S}.\tilde{n} \le (1+\epsilon_c)\mathcal{S}.n$
- $(1-\epsilon_c)\mathcal{S}.c(u) \le \mathcal{S}.\tilde{c}(u) \le (1+\epsilon_c)\mathcal{S}.c(u)$

Proof. At each of the m sensor nodes, after the Synopsis Generation process, at most $\frac{\log \mathcal{S}.n}{\epsilon}$ locally frequent items remain. The union bound suggests that setting the confidence parameter for the duplicate-insensitive sum algorithm as $\delta_c = \frac{\epsilon\delta}{m \log N}$ is sufficient to have Lemma 5 hold. $\Box$

Lemma 6. *With probability at least $1-\delta$, for any synopsis $\mathcal{S}$, $(1-\epsilon_c)\mathcal{S}.n \le \mathcal{S}.\tilde{n} \le (1+\epsilon_c)\mathcal{S}.n$. Moreover, if the class of $\mathcal{S}$ is i, $(1-\epsilon_c)2^i < \mathcal{S}.\tilde{n} \le (1+\epsilon_c)2^{i+1}$.*

Proof. Simple proof using induction. $\Box$

Lemma 7. *With probability at least $1-\delta$, for any synopsis $\mathcal{S}$, the estimated frequency $\tilde{c}(u)$ of any item u satisfies: $\mathcal{S}.\tilde{c}(u) \le (1+\epsilon_c)c(u)$.*

Proof. By induction on $\mathcal{S}.\tilde{n}$.

Base step ($\mathcal{S}.\tilde{n}$): Such synopsis can only be generated by Synopsis Generation. If the item is not pruned, Lemma 5 applies. Or else, the lemma trivially holds.

Induction step: Either the synopsis is generated locally, or the synopsis is a result of invoking Synopsis Fusion. In case of the former, the *base step* argument applies. For the latter, Lemma 7 holds because $\oplus$ is constant error duplicate-insensitive sum and because of the induction assumption. $\Box$

Lemma 8. *With probability $1-\delta$, for any synopsis $\mathcal{S}$ of class i, the estimated frequency $\tilde{c}(u)$ of any item u satisfies: $(1-\epsilon_c)(c(u) - \frac{i\epsilon\mathcal{S}.n}{\log N}) \le \mathcal{S}.\tilde{c}(u)$*

Proof. By Induction on $\mathcal{S}.\tilde{n}$.

Base step ($\mathcal{S}.\tilde{n}$ of class i): If u is pruned, $c(u) < \frac{i\epsilon\mathcal{S}.n}{\log N}$ and $\mathcal{S}.\tilde{c}(u) = 0$. Or else, Lemma 5 can be applied. In either case, Lemma 8 holds.

Induction step: If the synopsis is generated locally, the *base step* argument applies. Otherwise, it is a result of the synopsis fusion function. Let the two inputs to Synopsis Fusion be synopses $\mathcal{S}_1$ and $\mathcal{S}_2$ of class k. By induction assumption, $(1-\epsilon_c)(\mathcal{S}_1.c(u) - \frac{k\epsilon\mathcal{S}_1.n}{\log N}) \le \mathcal{S}_1.\tilde{c}(u)$ and $(1-\epsilon_c)(\mathcal{S}_2.c(u) - \frac{k\epsilon\mathcal{S}_2.n}{\log N}) \le \mathcal{S}_2.\tilde{c}(u)$. Let $\hat{c}(u)$ denote the frequency estimate of an item u before the pruning step. It follows that $(1-\epsilon_c)(\mathcal{S}_f.c(u) - \frac{k\epsilon\mathcal{S}_f.n}{\log N}) \le \hat{c}(u)$. If item u is not pruned, $\mathcal{S}_f.\tilde{c}(u) = \hat{c}(u)$ and Lemma 8 holds (irrespective of whether the class of $\mathcal{S}$ increases or not). Or else, the class of S_f is $k+1$ and $\hat{c}(u) < \frac{\epsilon(1-\epsilon_c)\mathcal{S}.\tilde{n}}{(1+\epsilon_c)\log N}$. Using Lemma 5, $\hat{c}(u) < \frac{\epsilon(1-\epsilon_c)\mathcal{S}.n}{\log N}$. Hence, Lemma 8 holds. $\Box$

Lemma 9. *For a synopsis $\mathcal{S}$ of class i, $\mathcal{S}.n < \frac{2^{i+1}}{1-\epsilon_c}$ with probability at least $(1-\delta)$.*

Proof. Follows from Step (3) of Algorithm 2. $\Box$

Lemma 10. *For any item u belonging to a class i synopsis $\mathcal{S}$, $\mathcal{S}.\tilde{c}(u) \geq \frac{(1-\epsilon_c)\epsilon 2^i}{(1+\epsilon_c)\log N}$ with probability at least $(1-\delta)$.*

Proof. After synopsis generation, $\mathcal{S}.\tilde{c}(u) \geq \frac{i\epsilon 2^i}{\log N}$. In synopsis fusion, the class of a synopsis increases only in Step (3) and $\mathcal{S}.\tilde{c}(u) \geq \frac{(1-\epsilon_c)\epsilon 2^i}{(1+\epsilon_c)\log N}$ holds immediately after that. □

Lemma 11. *The maximum number of (non-zero) frequency estimates in any class i synopsis $\mathcal{S}$ is $\frac{2(1+\epsilon_c)^2 \log N}{(1-\epsilon_c)^2\epsilon}$.*

Proof. Follows from Lemma 9 and Lemma 10. □

Proof of Theorem 1. Lemma 5, Lemma 7 and Lemma 8 provide accuracy bounds on the synopses generated by the synopsis fusion function. Since $\oplus$ is constant error duplicate-insensitive sum, the same bounds hold after synopsis evaluation too. The bound on communication load follows from Lemma 11.