Another Outlier Bites the Dust: Computing Meaningful Aggregates in Sensor Networks

Antonios Deligiannakis [#1], Yannis Kotidis, Vasilis Vassalos [*2], Vassilis Stoumpos, Alex Delis [#3]

[#]*Dept of Electronic and Computer Engineering, Dept of Informatics, Dept of Informatics & Telecommunications*
Technical University of Crete, Athens U. of Econ and Business, University of Athens
[1]adeli@softnet.tuc.gr
[2]{kotidis, vassalos}@aueb.gr
[3]{stoumpos, ad}@di.uoa.gr

***Abstract*— Recent work has demonstrated that readings provided by commodity sensor nodes are often of poor quality. In order to provide a valuable sensory infrastructure for monitoring applications, we first need to devise techniques that can withstand "dirty" and unreliable data during query processing. In this paper we present a novel aggregation framework that detects suspicious measurements by outlier nodes and refrains from incorporating such measurements in the computed aggregate values. We consider different definitions of an outlier node, based on the notion of a user-specified *minimum support*, and discuss techniques for properly routing messages in the network in order to reduce the bandwidth consumption and the energy drain during the query evaluation. In our experiments using real and synthetic traces we demonstrate that: (i) a straightforward evaluation of a user aggregate query leads to practically meaningless results due to the existence of outliers; (ii) our techniques can detect and eliminate spurious readings without any application specific knowledge of what constitutes normal behavior; (iii) the identification of outliers, when performed inside the network, significantly reduces bandwidth and energy drain compared to alternative methods that centrally collect and analyze all sensory data; and (iv) we can significantly reduce the cost of the aggregation process by utilizing simple statistics on outlier nodes and reorganizing accordingly the collection tree.**

I. Introduction

Recent advances in remote sensing equipment, computing hardware and communication technology have made the creation and deployment of large scale sensor networks easier and cheaper. Their uses in monitoring natural or artificial conditions and processes in diverse physical environments – such as wildlife monitoring, health-care, traffic monitoring, agriculture, production monitoring, battlefield surveillance – have subsequently multiplied. Sensor networks typically consist of small devices equipped with a power source, a processing unit with limited processing power and memory, one or more sensing devices that obtain readings, and a communication module for relaying these readings or the result of their processing to other *sensor nodes* nearby.

A lot of recent research has focused on the problem of efficiently answering declarative queries in such networks. These efforts primarily focus on evaluating aggregate queries, which are of great importance to surveillance applications [16], [24], and on enabling *in-network processing* by combining individual sensor readings as they are transmitted towards a *base station*. Such an in-network paradigm dramatically reduces the communication cost, often by orders of magnitude, and thus leads to prolonged network lifetime. An equally important line of research addresses the issue of data cleaning of sensor readings [10], [11], [19]. A measurement obtained by a node is only an approximation of the physical quantity observed and is constrained in accuracy and precision by the characteristics of the sensing device. Sensors are also often exposed to conditions that adversely affect their sensing devices, yielding readings of low quality. For example the humidity sensor on a MICA mote is very sensitive to rain drops. Moreover, sensor nodes often provide imprecise individual readings after a failure, i.e., they tend to *fail dirty* [10]. Thus, data processing applications using sensor networks must deal with information that is at times unreliable and unpredictable.

In this paper, we present a novel query processing framework for aggregate queries over a network consisting of inexpensive, wireless sensor nodes that are prone to generating dirty data. Our approach computes robust, or "meaningful", aggregates by identifying and excluding potentially "abnormal" readings. In our query processing model, introduced in our recent preliminary work [15], the sensor network propagates, in multiple hops towards the base station, the aggregate values, and also recognizes and reports a concise set of readings that are believed to be outliers, along with a set of characteristic values, *i.e., witnesses*, that have been used to derive the requested aggregates. In the current paper we build a comprehensive framework for identifying outliers and simultaneously computing in a resilient manner aggregate values in-network. In our framework, users are able to control the minimum amount of support that the readings of each node are required to achieve in order for the node to not be classified as an outlier. This ability is provided through a query-defined parameter, termed as *minimum support*, that regulates the number of tests that measurements have to pass in order to be included in aggregates. This way, our techniques are resilient to environments where spurious readings originate from multiple nodes at the same epoch, due to a multitude of different, and hence unpredictable, reasons. The framework presented in this paper supports a rich query model that permits grouping, and also allows for semantic constraints on the definition (and detection) of outliers. Respecting the *minimum support* for a query and the enriched query model

creates significant challenges for efficient and effective outlier detection, which we successfully address.

A key characteristic of our framework is that we do not use the same, originally constructed, collection tree to gather values throughout the life of an aggregate query, but periodically seek to readjust it based on easy to compute statistics. Using a single, monolithic collection tree constructed in advance, as in [15], [16], [23], does not take into account the existing readings and can lead to suboptimal decisions when computing and communicating outliers in the network. We overhaul the aggregation and outlier detection processes and periodically determine proper routing paths, based on simple statistics collected during query processing.

Our contributions can be summarized as follows:

1. We propose a framework (Section III) and algorithms (Section IV) for in-network aggregate query processing in the presence of multiple unreliable sensor nodes. Our computation model is based on simultaneous aggregate processing and outlier detection, and results in reporting both outlier and witness nodes in addition to the aggregates, to create increased user confidence in the produced results and to enable further investigation of suspicious readings in an efficient manner. Our framework allows the incorporation of different metrics (Section IV) for similarity testing between measurements of sensor nodes.

2. We show that the generation of outliers by sensor nodes renders raw aggregation techniques meaningless and inefficient. We thus develop a novel outlier-aware process for constructing the collection tree that takes into account the nature of our evaluation process (Section V). Our algorithms are based on a periodic reorganization of the collection tree using simple statistics of how often the measurements of two sensor nodes are similar. We show (Section VI) that the collection trees constructed by our algorithms result in substantial savings in the number of transmitted bits (up to 43%) and in energy consumption compared to existing methods that are outlier-oblivious when constructing the collection tree. The overhead of communicating the necessary statistics and running our reconstruction algorithm is comparable to the bandwidth consumption of one epoch, and less than 0.4% overall.

3. We perform an extensive experimental evaluation of our framework using real traces of sensory data (Section VI). It demonstrates significant benefits compared to alternative approaches a) in the quality of the reported aggregate computed through our aggregation framework, and b) in energy and bandwidth consumption (up to 6.5 times). We also report comparable performance, in the number of detected and reported outliers, to an out-of-network computation of outliers that uses the full set of node readings per epoch.

II. Related Work

The feasibility of using an embedded, lightweight database management system for sensor network data processing has been demonstrated by recent research [16], [24]. In particular, the focus has been on aggregate query processing [3], [6], [7], [17]. To enable efficient and effective in-network processing of aggregate queries, many techniques for computing energy-efficient data routing paths such as the aggregation tree have been proposed [9], [18]. Alternative techniques do not utilize an aggregation tree, but rather compute aggregation queries using decentralized algorithms [1], [12].

Many recent publications have also pointed out that current sensor nodes are prone to failures and often tend to transmit unreliable readings due to environmental interference and other local disturbances, and hence there is need for *data cleaning* of sensor readings. In [22] the authors propose a fuzzy approach to define the correlation among sensor readings, assign a confidence value to each of them, and perform a fused weighted average. In [10] the authors present ESP, a data cleaning framework in support of pervasive applications. ESP allows programmers to specify five pipelined cleaning stages using high-level declarative queries over data streams produced by the sensors. In [11] a probabilistic technique for cleaning RFID data streams is presented. Khoussainova et al. [13] propose a framework for correcting input data errors using integrity constraints. Our approach differs from these techniques in that we do not try to "mask" abnormal readings, but instead we promote them into first class citizens, on par with the requested aggregates, and make them available to the monitoring application, to enable further investigation.

A few recent works propose voting protocols for outlier detection. In particular, Chen et al. [2] seek to identify faulty sensors using a localized voting protocol. In Section III-A we describe how local voting schemes are prone to erroneous decisions when nodes that observe interesting events are not in direct communication. Furthermore, the proposed technique requires a periodic process: nodes marked as faulty are ignored until the process runs again. As discussed earlier, sensors can give occasional spurious readings due to, for example, environmental conditions, without being faulty, and by ignoring them we may miss important observations. Furthermore, while [2] requires a separate costly process that runs periodically to find faulty nodes, our framework allows us to capture outlier readings, produced for a variety of reasons including sensor faults, in real time during query evaluation and piggyback this information on messages used in query processing.

Another localized voting scheme is proposed in [23] that ranks sensor readings according to their validity. Ranking explores a *correlation network* that requires several epochs to be finalized and is based on readings from a starting epoch. This approach differs from ours in that it does not allow for the definition of a minimum support and does not provide techniques for building and reorganizing the collection tree. Moreover, the correlation network is built based on statistics collected at a single epoch, while our approach reorganizes the collection tree based on statistics collected throughout the entire period between consecutive reorganizations. Finally, the technique proposed in [19] computes outliers by first examining their recent history of measurements in order to verify whether the last reading could be an outlier. This means that it cannot identify the common case of nodes that fail dirty and reach a maximum value, since this maximum value will

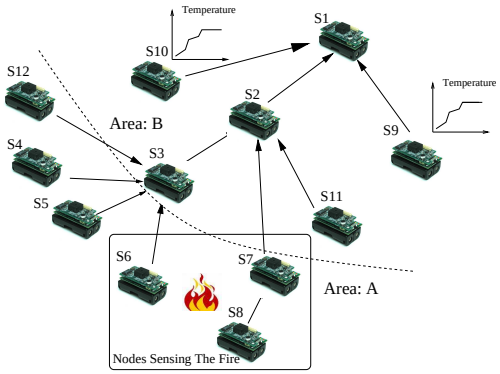

Fig. 1. Sample Collection Tree

appear multiple times in the sensor's reading.

III. A FRAMEWORK FOR MEANINGFUL AGGREGATE COMPUTATION

A. Aggregate Computation in the Real World

Consider a query that computes the average temperature in the two areas covered by the sensornet depicted in Fig. 1. The query is a simple GROUP BY query, where the grouping attribute takes the values A, B (for the two areas). Let's assume a sensor reading is termed an outlier if it isn't *supported* by the readings of at least two other nodes, in other words the desired *minimum support* is 2. For simplicity we assume that the aggregate is collected at node S_1, which acts as the base station in our example. We use x_i to denote the temperature readings provided by node S_i. The collection tree [16] is also depicted in the figure. A typical way of computing the average value from the temperature readings is for each node to compute the SUM and COUNT functions in its subtree for each area (i.e., each value of the grouping attribute) and propagate these values to its own parent [16], [24].

In our example, nodes S_6, S_7 and S_8 observe an open fire and therefore their readings are expected to be a lot higher (and fluctuate more) than, for example, those of S_2. When S_3 receives the values of its children nodes, the readings of node S_6 appear to be suspicious, since no other node in that subtree is aware of the fire. If we decide to reject the reading of S_6 (for instance using a voting protocol [2]), the monitoring application will lose a crucial observation. Techniques based on smoothing [10], [11] will also obscure the outcome, especially if many more nodes are rooted at S_3.

In our framework, we tentatively put the reading of node S_6 in a *list of outliers* O_3 communicated by node S_3 to its parent node in the tree S_2. Assuming that the measurements of nodes S_4, S_5 and S_{12} over the last few epochs are similar enough, their measurements can be combined to calculate a running aggregate value for the group-by identifier A as $A : (x_4 + x_5 + x_{12}, 3)$, where 3 denotes the number (COUNT) of measurements included in the aggregate of Area A. Moreover, one of the nodes S_4, S_5, or S_{12} will also be inserted into the *witness set*, along with its corresponding measurement, and be communicated up the collection tree, to provide evidence to the base station for the computed aggregate value. Note that in our example we assumed a minimum support value of 2 and that S_4, S_5 and S_{12} gain the required amount of support from each other when their values reach S_3. Also note that S_3 is located in a different area (i.e., group) from its children nodes. Our framework allows us to restrict the nodes whose measurements can "witness" the measurements of S_3 to nodes belonging in the same group. If we use this feature, S_3's value cannot be witnessed by any node and will be transmitted in the list of outliers communicated to its parent node S_2.

Now let us concentrate on node S_2. This node will receive from the left subtree a pair of aggregate values for Area A by node S_3, namely $A : (x_4 + x_5 + x_{12}, 3)$, an outlier list containing the values x_3, x_6 and a witness list containing one of the values x_4, x_5 or x_{12} (depending on which node was selected as the witness). Its middle subtree contains nodes S_7 and S_8. Their readings are similar, but these nodes reach only a support of 1, which is less than the desired minimum support of 2. Thus S_7 includes the readings of S_7 and S_8 in the outlier list transmitted to S_2. At this point, at node S_2, the nodes S_6, S_7 and S_8 can reach the required support in order to be included in the partial aggregate of Area A. Moreover, one of these nodes will be selected to become a witness in S_2. We also need to note that if the latest measurements by nodes S_2, S_3 and S_{11} are similar, then these nodes may also reach the required amount of support at node S_2 (since they are also characterized by the same group-by value) and provide a running aggregate for Area B.

Sensor nodes S_9 and S_{10} are two nodes that fail dirty. These nodes start reporting abnormally high readings that cannot be justified by any of their neighbors or the values that are provided in the witness list, and, as explained in the previous section, would be missed by the outlier detection technique of [19]. If the query had specified a minimum support of 1, these two nodes might, at some epochs, witness each other and, thus, their readings would end up being included in the reported aggregate. However, this cannot occur in our example with the minimum support value of 2.

Finally, we can also see from Fig. 1 that an alternative organization of the collection tree where both S_6 and S_8 select S_7 as their parent node could lead to bandwidth savings at nodes S_3 and S_7. In particular, node S_3 would have to transmit one fewer outlier to node S_2. Moreover, we note that in node S_7 these three sensors could gain enough support to be included in the computed aggregate and have all three of them be replaced by a single witness. One can also observe that if S_{12} selects S_{10} as its parent node, then the values of S_4, S_5 and S_{12} will have to be transmitted all the way to S_1 in order to gain enough support. Thus, an algorithm that properly forms and reorganizes the aggregation tree may lead to substantial bandwidth savings, when compared to a monolithic approach that forms the aggregation tree with a technique such as TAG [16]. We describe such an algorithm in Section V and show its benefits in Section VI-A.

B. Extended Query Model

We consider aggregate queries of the form:

```
SELECT groupingAttrs, AggrFun(s.value)
```

```
FROM Sensors s
WHERE cond
SAMPLE PERIOD e FOR t
GROUP BY groupingAttrs
MINIMUM SUPPORT MinSupp
[ CONSTRAIN TEST GROUP ]
```

where AggrFun() is a distributive or algebraic function such as MAX,MIN,COUNT,SUM,AVG. Our work also captures GROUP BY queries, based on the user-defined grouping attributes `groupingAttrs`. The period (e) in the above query is the *epoch duration* and determines the frequency at which data is acquired from the sensors. Parameter (t) specifies the life span of the query. The minimum support required for each node to be incorporated into the aggregate is provided by the `MinSupp` value. Finally, an optional CONSTRAIN TEST argument may limit the cases when a node may witness the measurements of another node. For ease of presentation, in this paper we describe how to restrict such witness tests amongst nodes that are characterized by the same `groupingAttrs` values. However, it is simple to extend our techniques to cases where a user-defined function specifies the nodes that can participate in the witness test.

More formally, during query evaluation, the measurements of a node S_i can *witness* (or *support*) the measurements of node S_j if: (i) The latest set of measurements of S_i and S_j, when compared with a user-defined similarity function, exhibit a similarity above a user-defined threshold; and (ii) S_i and S_j are characterized by the same grouping attributes `groupingAttrs`, if both a group-by clause and the CONSTRAINT TEST GROUP clause have been specified. Many similarity functions are symmetric; that is, when S_i witnesses S_j, then S_j also witnesses S_i. In the presentation of our algorithm we assume the use of such symmetric similarity functions. However, as we describe in Section IV-D, the extension to asymmetric similarity functions is simple. Our framework can thus handle a variety of functions used for performing the similarity test. Some examples of such functions are presented in Section IV-B. Finally, an outlier is defined as a node that is witnessed by fewer than `MinSupp` other nodes. The measurements of such nodes do not contribute to the query result, but are still communicated to the application for further analysis.

Example 1: A sample supported query is the following:

```
SELECT building, floor, AVG(s.temperature)
FROM Sensors s
SAMPLE PERIOD 30 sec FOR 1 day
GROUP BY building, floor
MINIMUM SUPPORT 2
CONSTRAIN TEST GROUP
```

The above query computes the average temperature readings of sensor nodes for each floor in each building. The aggregate value is computed for thirty seconds, and the query will be executed for one day. The measurements of each sensor node will need to be similar to those of at least two other nodes within the same building and floor, in order to be included into the aggregate.

Symbol	Description
`Root`	The node that initiates a query and which collects the relevant data of the sensor nodes
S_i	The i-th sensor node
`CacheSize`	The maximum number of epochs a measurement remains in the cache. Also determines the maximum number of measurements stored per node in the cache
`TestInterval`	The minimum number of data value pairs required to perform the witness test
$F[j][k]$	The number of successful witness tests between the measurements of S_j and S_k
`MinSupp`	The required minimum support specified by the posed query

TABLE I
SYMBOLS USED IN OUR ALGORITHMS

C. Transmitted Data

As discussed in the example of Fig. 1, during query execution, an intermediate node in the tree receives (using a protocol like TAG [16]) a list of aggregate values, one per each different set of `groupingAttrs` values, a list of witnesses and a list of outliers calculated at each of its children nodes. When we refer in this paper to the transmission of an aggregate value or the reading of a sensor node, each such value needs to be accompanied by its corresponding grouping attributes. An optimization can be performed whenever the group-by clause refers to a static predicate such as the node's identifier, or the node's location in the case of immobile sensor nodes. In such cases, at the first epoch of each query, each node may transmit its grouping attributes along with its identifier towards the `Root` node. In this way, the node's identifier can be used in the execution of the query so that other sensors in its path to the `Root` node can determine its `groupingAttrs` values, without having to transmit them at each epoch. Such an optimization is even possible in the case of dynamic predicates that are based on the sensor's current reading, since the latest reading of each sensor can also be used to determine its `groupingAttrs` values. However, in the case of other dynamic predicates such an optimization may not be possible.

Note that each transmitted witness and outlier value does not necessarily reach the `Root` node that poses the query. These values may be witnessed at some intermediate nodes and removed from the transmitted data. This observation provides the intuition for our algorithm for periodically reorganizing the collection tree. If we monitor how often the witness test between pairs of sensor nodes succeeds, then each node can select a parent in the collection tree through which it expects to find the most witnesses and at relatively short distances, in number of hops.

IV. MEANINGFUL AGGREGATE COMPUTATION

We now present `SensibleAggr-supp`, our algorithm for the computation of aggregate functions and the simultaneous management of outliers in sensor networks. `SensibleAggr-supp` works by propagating upwards in the aggregation tree partial aggregates along with sets of witnesses and outliers computed by the nodes for their subtrees. Through this process, at each epoch, we compute at the `Root` of the tree (i.e., the node that initiated the query) the aggregate for each different value of `groupingAttrs`, excluding nodes that are determined to be outliers, based on the specified minimum

support MinSupp. The Root also computes the final set of outliers and witnesses. The monitoring application may decide to use this information to further investigate suspicious behavior by the nodes. Table I summarizes the notation used in this paper.

A. Preliminaries

The *level* of a node in a sensor network denotes, given the transmission range of the sensor nodes, the minimum possible distance, in number of hops, of the node from the Root. If while traversing a path, we continuously reach nodes at a higher level (i.e., with a higher minimum distance from the Root) than the ones at the origin of each edge, then the traversed path is termed as a *descending path*. A node S_i is an *ancestor* of S_j in a given collection tree if there exists a descending path from S_i to S_j. Similarly, a node S_j is a *descendant* of S_i if S_i is an ancestor of S_j. A node S_i is a *potential ancestor* of S_j if, given the transmission range of the nodes, there exists a collection tree where S_i is an ancestor of S_j. Similarly, a node S_j is a *potential descendant* of S_i if S_i is a potential ancestor of S_j.

B. Examples of Similarity Tests between Nodes

SensibleAggr-supp frequently tests whether the recent measurements of two sensor nodes are similar. If this is the case, then each sensor can *witness* the measurements of the other. We will call this similarity test in this paper a *witness test*. We consider the following alternatives:

- **Correlation Coefficient:** If we consider the readings x_k, x_l of sensor nodes S_k, S_l respectively as random variables, the correlation coefficient $r_{k,l}$ is defined as:

$$r_{k,l} = \frac{cov(x_k, x_l)}{\sigma_{x_k}\sigma_{x_l}} = \frac{E(x_k x_l) - E(x_k)E(x_l)}{\sqrt{E(x_k^2) - E^2(x_k)}\sqrt{E(x_l^2) - E^2(x_l)}}$$

where $cov()$, σ and $E()$ stand for the covariance, standard deviation and expected value respectively. The correlation coefficient takes values in the interval [-1,1]. Given a threshold θ provided by the application and communicated to the nodes during the query initialization, the witness test succeeds when $r_{k,l} \geq \theta$.

- **Extended Jaccard Coefficient:** If we consider the readings x_k, x_l of sensor nodes S_k, S_l respectively as vectors and denote their dot product as $x_k \cdot x_l$, the extended Jaccard coefficient $j_{k,l}$ is defined as:

$$j_{k,l} = \frac{x_k \cdot x_l}{\|x_k\|^2 + \|x_l\|^2 - x_k \cdot x_l}$$

Again, given a threshold θ provided by the application and communicated to the nodes during the query initialization, the witness test succeeds when $j_{k,l} \geq \theta$.

- **Regression-Based Approximation:** If we consider the readings x_k, x_l of sensor nodes S_k, S_l respectively as random variables, we may apply approximation techniques to identify the error of approximating x_l given x_k. For example, the work in [5], [14] proposed using a linear regression model for this approximation. Using such a technique, we may determine that the witness test will succeed if the reconstruction maximum/absolute relative error for x_l is below an application-defined threshold (i.e., 2%). While we investigated this technique, we omit it from our discussion due to its poor performance in our experiments. Please note that, unlike the cases of the correlation and extended Jaccard coefficients, the regression-based approximation is an example of an asymmetric similarity function.

C. Memory and Cache Management

Since the similarity tests cannot be performed simply based on the last received measurement of the sensor nodes, but also require the knowledge of measurements from the recent past, our algorithm maintains in a small cache the latest CacheSize measurements received by descendant sensor nodes. The cache is organized as an array indexed by epoch, so accessing the recent measurements in the cache is performed using the modulo operator and the query epoch. For example, if $\#epoch$ denotes the current query epoch, then the latest measurement is stored in the position $\#epoch \ mod$ CacheSize. When no value is received for a descendant node, we store a NULL value in the cache for its measurement in the current epoch. As we will see later in this section, this happens when the node belongs to neither the set of received witnesses nor the set of received outliers.

The witness test between the readings of two nodes is performed over the latest TestInterval $\leq$ CacheSize readings of the two sensors in the cache that were *simultaneously* not NULL at the same epoch. Moreover, we only perform witness tests amongst pairs of sensor nodes for which we have received a measurement in the *current* epoch. In cases of sudden changes in the readings, this requirement does not allow the witness test between pairs of nodes to succeed based solely on the similarity of the readings of these nodes in prior, but not the current, epochs.

Notice that using a CacheSize larger than TestInterval enables us to have TestInterval recent available readings for a node, and hence to be able to use it in witness tests, even if a few recent readings are unavailable. If CacheSize $=$ TestInterval, as in [15], missing one measurement for a node means disqualifying it for witness comparisons for TestInterval $- 1$ epochs (i.e., until the cache for it is full).

Our framework also suggests that we do not need to store in the cache information for sensor nodes for which we have not received any measurement in the last CacheSize $-$ TestInterval $+ 1$ epochs. For such nodes, it is certain that the witness test with other nodes cannot succeed unless at least TestInterval measurements are received in subsequent epochs. Thus, it suffices for these nodes to remove their history and start, when necessary, with an empty buffer of measurements.

Finally, we note that many sensor nodes have very limited memory capabilities. Even though our experience with our techniques revealed that the required size for the cache is typically small, we still need a replacement strategy for sensor nodes with severe memory constraints: we evict from the cache measurements originating from those sensor nodes for which we have received the fewest (non-NULL) measurements in the

last TestInterval epochs.

Example 2: Consider that the sensor node S_1 maintains in its cache measurements for itself and for nodes S_2, S_3 and S_4, as depicted in the following table at the left.

	Cache Position					
ID	0	1	**2**	3	4	5
S_1	20.05	20.51	**20.69**	21.17	21.22	21.36
S_2	NULL	21.15	**21.21**	21.77	NULL	21.93
S_3	NULL	25.79	**25.82**	26.57	26.51	26.91
S_4	22.09	22.40	**NULL**	22.97	22.97	23.14

Cache Measurements

	NODE ID			
ID	S_1	S_2	S_3	S_4
S_1	1.000	0.992	0.975	-
S_2	0.992	1.000	0.996	-
S_3	0.975	0.996	1.000	-
S_4	-	-	-	-

Correlation Coefficients

Let the current epoch be epoch 8 and TestInterval $= 4$ (in this example CacheSize $= 6$). Thus, the measurements of the current epoch can be found at position 2 of the cache (current measurements marked in bold). Based on our discussion, the witness test cannot be performed between S_4 and any other node, since we do not have a measurement for S_4 at the current epoch. The witness test between nodes S_2 and S_3 can be performed since on 4 (=TestInterval) positions of the cache they both have non-NULL values. The witness test between nodes S_1 and S_3 will be performed over the measurements at positions 2,1,5,4. The computed correlation coefficient amongst all pairs of nodes is presented in the preceding table at the right. Since these numbers were derived from a real data set measuring the temperature in a room, one can see that the measurements of these nodes are strongly correlated. The extended Jaccard coefficient is computed in the same way.

D. Algorithm Description

Algorithm 1 presents an outline of our SensibleAggr-supp algorithm. The main steps and operations of the algorithm will be explained and analyzed throughout this section.

The SensibleAggr-supp algorithm is invoked at each node after it has received messages from its children nodes in the collection tree, in a manner similar to TAG [16]. Thus, the execution of the algorithm starts at the leaf nodes of the collection tree. Each received message starts with a bitmap containing 3 bits. Each bit that is set in this bitmap reveals the existence of a set of aggregate values, a set of witnesses, or a set of outliers, correspondingly, in the message. At least one of the bits must be set. For example, since a leaf node of the collection tree cannot witness its own measurements by itself, it characterizes itself as an outlier and transmits this information to its parent node, without including in the message an aggregate value or witnesses. As explained in Section III-C, each witness and outlier is described as an id-value pair, where the value of each node may also be accompanied or not (based on the query and whether the grouping attributes refer to static or dynamic predicates) by its corresponding grouping attributes. At a parent node, these values, along with the node's current measurement, are stored in the cache of measurements that the node maintains. We note that leaf nodes in the aggregation tree do not need to maintain this cache, as they do not perform any witness-tests.

Each node S_i first initializes its WitnessSet and OutlierSet lists to the union of the corresponding lists received by its children nodes. Since the measurements of current node S_i have not yet been compared to those of any other node, S_i needs to be inserted into the OutlierSet list. The algorithm then processes each outlier and examines whether its measurements can be witnessed by either those of other outlier nodes (Lines 7-12) or by those of existing witnesses (Lines 13-18). The witness test is performed by calling the canWitness() function with the ids of the two nodes being tested as arguments. This function operates on the available cached measurements of the two nodes and also depends on the groupingAttrs values of the two nodes, if the clause CONSTRAIN TEST GROUP has been specified.

Please note that witness tests are performed only between pairs of sensor nodes that have enough recent values stored in the cache. One can implement different similarity checks, such as those discussed in Section IV-B, by simply modifying this function. Since the focus of this paper is on symmetric functions for the witness test, the SensibleAggr-supp algorithm tries to avoid performing symmetric executions of this test (i.e., if the pair $\langle S_j, S_k \rangle$ has been tested, then the pair $\langle S_k, S_j \rangle$ is not examined). For the case of asymmetric witness functions, this optimization is not possible. It is important to emphasize that the processed lists WitnessSet and OutlierSet are of small size (often containing less than 5 entries each for small values of MinSupp). Their exact size depends on

Algorithm 1 SensibleAggr-supp(MinSupp) Subroutine

1: {S_i is the node being examined}
2: Set WitnessSet to the union of the received sets of witnesses from children nodes.
3: Set OutlierSet to the union of the received sets of outliers from children nodes. Also add S_i to OutlierSet.
4: Update the cache of S_i with the latest measurements received from nodes in the WitnessSet or the OutlierSet.
5: Set the support of each id in OutlierSet to 0
6: **for** $S_j \in OutlierSet$ **do**
7: **for** $S_k \in OutlierSet$ AND S_k lies after S_j in $OutlierSet$ **do**
8: **if** canWitness(S_j, S_k) **then**
9: Increase support of S_j and S_k by 1.
10: Increase $F[j][k]$ and $F[k][j]$ by 1 only if the latest readings of S_j and S_k were received by different children nodes of S_i.
11: **end if**
12: **end for**
13: **for** $S_k \in WitnessSet$ **do**
14: **if** canWitness(S_j, S_k) **then**
15: Increase support of S_j by 1.
16: Increase $F[j][k]$ and $F[k][j]$ by 1 only if the latest readings of S_j and S_k were received by different children nodes of S_i.
17: **end if**
18: **end for**
19: **end for**
20: **for** $S_j \in OutlierSet$ **do**
21: **if** support of S_j is greater or equal to MinSupp **then**
22: Move S_j from $OutlierSet$ to the $WitnessSet$
23: Incorporate measurement of S_j into the aggregate of its group
24: **end if**
25: **end for**
26: **for** $S_j \in WitnessSet$ **do**
27: **for** $S_k \in WitnessSet$ AND S_k lies after S_j in $WitnessSet$ **do**
28: **if** canWitness(S_j, S_k) **then**
29: Keep as witness only the node for which S_i has transmitted the most consecutive readings.
30: Increase $F[j][k]$ and $F[k][j]$ by 1 only if the latest readings of S_j and S_k were received by different children nodes of S_i and neither S_j nor S_k was an outlier at Line 3 of this algorithm.
31: **end if**
32: **end for**
33: **end for**
34: Transmit the remaining witnesses and outliers to the appropriate parent node(s) of the collection tree. Transmit the computed aggregate values (one per group) only to one of these parent nodes, if more than one are selected.
35: **if** Current epoch is last epoch prior to tree reorganization **then**
36: Add each $F[k][l]$ value computed at S_i to the value for the same pair (k, l) received by children nodes. Transmit all $F[][]$ values to the parent node that received the computed aggregate values.
37: **end if**

the value of MinSupp and on the existence (or not) of a CONSTRAINT TEST GROUP clause.

The witness list of a node S_i is further trimmed in Lines 26-33 of the SensibleAggr-supp algorithm, in order to reduce the number of witnesses transmitted to the node's parent. Let us focus on Line 29 of the SensibleAggr-supp algorithm: Whenever two witnesses are similar, our techniques trim the witness list by keeping only one of them. The node kept is the one for which S_i has transmitted the most consecutive values to its parent node (i.e., by including it into the witness or outlier sets in the preceding epochs). Such an optimization has the following characteristics and benefits: (1) It requires a single counter per each descendant node with measurements stored in the cache; (2) It increases the chances that the parent node will have, either now or in the following epochs, enough measurements (TestInterval) in its cache for the witness node to be able to participate in witness tests; and (3) It increases the chances that the memory replacement policy described in Section IV-C will not evict the node's measurements in the near future, if memory constraints exist.

A number of statistics is updated at each successful witness test between S_j and S_k. The support of each outlier is first increased by one. This is needed to monitor if the recent measurements of the outlier witness those of at least MinSupp other sensor nodes. The second statistic that is updated involves the number $F[j][k]$ of successful tests between these pairs of nodes at the sensor. We utilize these statistics in Section V for the reorganization of the collection tree. Please note that in Lines 10, 16 and 30 of the SensibleAggr-supp algorithm, these frequencies are updated only under some conditions. For example, for two outlier nodes for which we have received their latest measurements through the same child in the collection tree, the witness test between these two nodes will surely have been performed at that child node as well. Thus, we do not wish to increase the F values of these two nodes, as this increase has already been recorded.

E. Discussion of Algorithm Internals

Outlier support. Our algorithm, as presented here, does not take into account the support that a sensor may have achieved in lower levels of the collection tree while still remaining an outlier: each node in the OutlierSet has by default zero support when received. The main reason for this is bandwidth efficiency. While our algorithm requires only an $\langle id, value \rangle$ pair for each received outlier (with the possible inclusion of the grouping attributes as well, as explained in Section III-C), taking into account the support obtained at descendant nodes would have required extra information per outlier (i.e., its current support). At the same time, the benefit would be small: somewhat fewer calls to function canWitness(). Recall that the witnesses and outliers are propagated upwards in the collection tree. A successful witness test between the measurements of two sensor nodes will not be performed again at their parent node only if: (1) one of the two sensors becomes a witness at the child node and (2) that same sensor is later removed from the witness list, due to a successful witness test with another witness of the sensor (Lines 26-33 of the algorithm). As bandwidth consumption is by far a larger component of energy drain on sensors than CPU usage, our current approach is preferable. Moreover, our current approach is also more general, as it is applicable in both cases (selecting a single or multiple parents per node) discussed in Section V (as opposed to the alternative approach, where it is impossible to determine if a witness test has already been performed in lower tree levels, in the case of sensors with multiple parent nodes).

Support update. In our algorithm, every successful witness test increases the support of an outlier o by 1. Given that each witness is supported by at least MinSupp nodes, why can't we directly "translate" the support of the witness into MinSupp for o? The reason is that many useful similarity functions (all the ones presented in Section IV-B) are not transitive. The following table provides an example of the non-transitivity of the similarity function. Let us assume that three sensor nodes obtained binary (0 or 1) readings shown in the table. The table also presents their computed similarities, based on the correlation coefficient, for the sample set of readings.

	Epoch				
ID	0	1	2	3	4
S_A	0	1	0	0	1
S_B	0	1	0	1	1
S_C	0	1	0	1	0

Evaluation
r_{S_A,S_B}=0.67
r_{S_B,S_C}=0.67
r_{S_A,S_C}=0.17

For θ=0.6, node S_A is similar to S_B and node S_B is similar to S_C. Nevertheless the witness test between S_A and S_C fails. Therefore, only the support of nodes with which successful witness tests have been performed can be taken for granted. In Section VI-B (Figure 10), we evaluate different variants of our algorithm where support is updated by more than 1 and find that our standard, conservative, approach is superior.

V. Collection Tree Reorganization

As mentioned in Section IV, the collection tree is periodically reorganized. A poor construction of the collection tree could lead to many nodes finding similar measurements (i.e., support) by other sensors only at the Root node, or at nodes near the Root. This would essentially result in a computation with bandwidth requirements close to those of performing a SELECT * query on the sensor network per epoch. It is much preferable to route the witnesses and outliers towards the direction of nodes where they are expected to be "matched" (witnessed) by outliers or witnesses received through other parts of the collection tree.

Before presenting the algorithm for the reorganization of the collection tree we must answer the following questions: (1) Will all the witnesses and outliers of a node be propagated towards a single parent node, or towards multiple nodes? (2) Given the answer to the above question, how do we decide the appropriate parent node(s)? The collected statistics in table $F[][]$ will be considered at this stage.

Concerning the first question, our collection tree reorganization algorithm can produce either of the two choices, i.e., selecting a single or multiple parent nodes, depending on the setting of a single parameter. As we will shortly demonstrate, the basic concept of the algorithm is similar for

both approaches. While in this paper we present the algorithm for both approaches, we suggest using a single parent for propagating all the witnesses and outliers as this leads to fewer transmitted messages per node.[1]

The important question that remains is the choice of the appropriate parent node(s) of each sensor. The reorganization algorithm proceeds bottom-up, based on the level (see Section IV-A) of each node. Each node S_i first waits for notifications from children nodes in the collection tree. Each notification by a child node S_j of S_i is accompanied by a list of node identifiers. This list of identifiers represents descendants of S_i in the new collection tree, reachable through S_j. Until the next reorganization of the collection tree, each message by S_j to S_i may contain only witnesses and outliers within the list transmitted at the aforementioned notification. Please note that if S_j selects more than one parent node, then the list transmitted to S_i may contain only a subset of the descendants of S_j, since some nodes may be "assigned" by S_j to another (parent) node. Finally, S_i creates a list $managedIds_i$ by combining the received lists along with S_i's identifier.

Intuitively, in order to reduce the bandwidth consumption, we should seek to route the witnesses and outliers of S_i towards other nodes in the aggregation tree where witness tests involving these witnesses and outliers may be satisfied. The closer such nodes exist and the larger the probability of successful witness tests at them, the more plausible it seems to route the nodes in WitnessSet and OutlierSet towards them. Thus, for each node identifier $S_j \in managedIds_i$, S_i examines: (1) the probability with which it can be witnessed by each node S_k of the entire collection tree (or by a part of the collection tree, if the CONSTRAIN TEST GROUP clause has been specified). Please note that since many examined nodes S_k will very likely not belong to the subtree of S_i, S_i has no knowledge of the frequencies $F[j][k]$; and (2) the corresponding distance (in terms of hops) from S_i to the closest potential ancestor S_p of S_i (again, in terms of hops) that is also a potential ancestor of S_k. Let $upwardDistance_i[k]$ denote this distance. By the way we defined this metric, $upwardDistance_i[k] = upwardDistance_i[p]$ obviously holds. Moreover, if S_k is a potential descendant of S_i, then $upwardDistance_i[k]$ is 0.

The first piece of information needed by S_i is provided in Lines 35-37 of the `SensibleAggr-supp` algorithm. At the last epoch before the reorganization of the collection tree, each node receives the witness test statistics computed at its children nodes, combines (i.e., adds) them with its own statistics and transmits them towards one parent node. Please note that this process of combining statistics at a node does not increase the size of the statistics, as this is bounded by the size of the $F[][]$ table. The entire information of the collection tree is gathered at the `Root` node. The `Root` node then transmits the result to the sensor nodes with a single broadcast message. Please note that base stations often have increased capabilities (in terms of both their transmission range, and their computational and memory capabilities) compared to regular sensor nodes. Thus, this approach is viable in most sensor network settings. If the `Root` node cannot contact all the sensor nodes directly, these statistics can be propagated top-down in the existing collection tree (i.e., before the reorganization).

[1]Because of the way that in our algorithm each node selects its parent(s) in the collection tree amongst nodes with a strictly lower minimum distance from the `Root`, schedules produced as in [16] can still be used. However, when selecting multiple parents, each node will need to make more than one transmissions.

Algorithm 2 `Reorganize()` Subroutine

```
1: {S_i is the node being examined}
2: Wait for notifications from potential child nodes. Merge S_i identifier with received
   lists of identifiers in managedIds_i.
3: upwardDistance_i[k] returns the distance of the closest potential ancestor of S_i
   that is also a potential ancestor of S_k.
4: Let fathers denote the set of potential parent nodes of S_i
5: for S_j ∈ fathers do
6:    weight[j] = 0 {Reset weights for each S_j}
7: end for
8: for S_j ∈ managedIds_i do
9:    if Selecting different parents for nodes in managedIds_i then
10:      for S_p ∈ fathers do
11:         weight[p] = 0
12:      end for
13:   end if
14:   for S_k for which S_i has an entry in the upwardDistance_i[k] array AND
      upwardDistance_i[k] > 0 do
15:      weight[nextHop[k]] += F[j][k] / upwardDistance_i[k]
16:   end for
17:   if Selecting different parents for nodes in managedIds_i then
18:      Set parent[j] to the parent node with the maximum weight
19:   end if
20: end for
21: if Selecting a single parent for nodes in managedIds_i then
22:    Set as the common parent the parent node with the maximum overall weight
23: end if
24: Notify each selected parent about the ids that will be forwarded to it
```

The second piece of information that we need is to compute the value of $upwardDistance_i[k]$ for any node S_k that is not a potential descendant of S_i. This information, computed only during the initial construction of the collection tree, and not during the reorganization phases, can be obtained as follows:

- Moving bottom up, based on the level of each node, nodes transmit their potential descendants. For an intermediate node, this set of nodes is produced by the union of sets received from descendant nodes and the identifier of the node itself. Representing the list of identifiers in interval lists can help reduce the amount of transmitted information for nodes closer to the `Root`.
- After this phase, the `Root` node is aware of all identifiers. The `Root` does not need to move upwards to reach a common ancestor of any node, and thus sets the *upwardDistance* of each identifier to 0 and transmits this result.
- Moving top-down, each node sets the *upwardDistance* of an identifier as zero, if the identifier is a potential descendant of the node, or to one plus the minimum *upwardDistance* received by nodes at one level closer to the `Root`. In the second case, it also records the parent node that provided the upwardDistance as the *nextHop* of that identifier. For the nextHop selection, ties are broken randomly.

A. Algorithm Presentation and Discussion

Our overall reorganization algorithm is presented in Algorithm 2. The core of the algorithm lies in Lines 14-16. For each node S_k of the collection tree that is not a descendant of

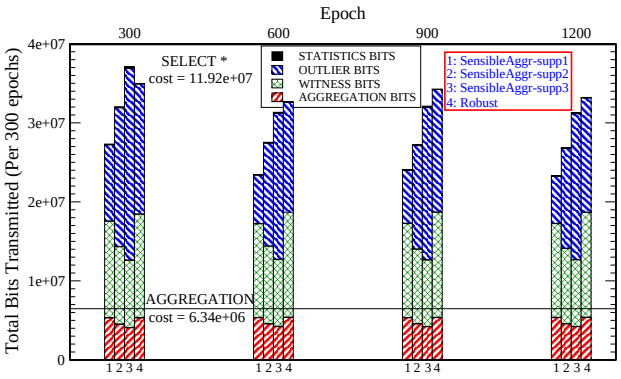

Fig. 2. Bandwidth consumption in synthetic dataset (per 300 epochs)

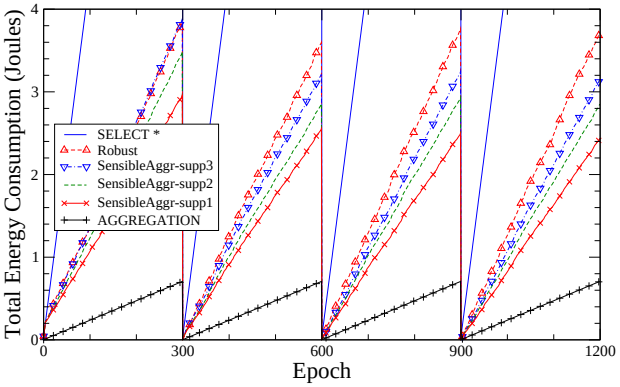

Fig. 3. Energy Consumption in synthetic dataset (per 300 epochs)

the current sensor S_i, we assign a weight only to the candidate parent node that is the $nextHop$ towards S_k. The candidate parent nodes (included in the $fathers$ list at Line 4 of the Algorithm) are the nodes that are one level closer to the `Root` node and which are within the transmission range of S_i. Since we would like to move towards nodes that are not distant and where a lot of witness tests may succeed, the weight assigned to nextHop[k] for each $S_j \in managedIds_i$ is proportional to $F[j][k]$ and inversely proportional to $upwardDistance_i[k]$ (Line 15). Finally, we note that the algorithm can select either a common or multiple parent nodes by checking a single parameter. However, as we stated earlier in this section, we suggest selecting a single parent for each sensor node.

While the intuition of the reorganization algorithm is as described above, and the algorithm can be executed in a distributed fashion, in our implementation we utilize a more bandwidth efficient centralized approach. Each node transmits towards the `Root` node the list of nodes with which it can communicate directly. Note that this information needs to be communicated *only* during the initial construction of the collection tree and not in subsequent reorganizations. Updates are required only when the set of neighbors of some sensor nodes is modified (i.e., due to link failures). In such cases, only the affected nodes need to communicate their statistics to the `Root`. After it has received the $F[][]$ values from its children nodes, the `Root` node then has all the necessary information to compute all the parameters (weights, nextHop and fathers) for each node, by simply processing the nodes bottom-up (using the connectivity information). Given `Root` nodes with increased capabilities, this approach is very practical, as the `Root` node can perform all the processing and then use individual transmissions to notify each sensor of all the computed values. In the case when a single parent per node is selected, as suggested in this paper, this simply requires transmitting to each sensor the identifier of its new parent node. Thus, Algorithm 2 is performed in our implementation exactly as described above, but at a central node, with appropriate recursive procedures, instead of top-down and bottom-up operations involving message transmissions.

B. Space Considerations

An important issue for the reorganization algorithm is the amount of information associated with the storage and transmission of the $F[][]$ statistics. Because the witness test is typically symmetric (i.e., when S_i can witness S_j, then S_j can also witness S_i), we only need $F[i][j]$ values that are above the diagonal ($i < j$). Thus, these statistics require up to $O(\frac{n\times(n-1)}{2})$ space, for a sensor network of n nodes. Our experience with the `SensibleAggr-supp` algorithm demonstrated that the F array is extremely sparse in all but a few nodes close to the `Root`, even for large sensor networks of 400 sensors. In most sensors the number of non-zero entries was often no more than 20 entries. We thus decided to not store the entire array, but only its non-zero entries at each node. When transmitting these non-zero entries during the tree reorganization (assuming that the reorganization is performed every $ReorgEpochs$ epochs), we first map the two coordinates of each of these entries to a single value, using a typical location function for arrays. Then, the information about the non-zero $F[][]$ entry can be transmitted with only $\lceil \log \frac{n\times(n-1)}{2} \rceil + \lceil \log ReorgEpochs \rceil$ bits. Note that the second summand is produced because the value of an entry in the F array cannot exceed the number of epochs between two reorganization periods, since these entries are reset to zero (i.e., removed from memory) after each reorganization.

For the case of nodes close to the `Root` node and with low memory capabilities, we propose using Count-Min sketches [4] in order to bound the amount of collected data and transmitted data. These sketches require only $O(\frac{1}{\epsilon} ln \frac{1}{\delta})$ space and $O(ln \frac{1}{\delta})$ update time, where ϵ is the L_1 error-guarantee of the sketch for point-wise estimation and δ is the probability of failure. The use of a such a sketch in nodes close to the `Root` gives several advantages such as: (1) reduced memory requirements; (2) reduced bandwidth requirements when transmitting these statistics; (3) easily composable statistics, as combining statistics by different sensors simply requires adding the sketches of these nodes; and (4) strong probabilistic guarantees on the quality of the reconstructed values.

VI. Experiments

We developed a simulator for testing the algorithms proposed in this paper under various conditions. In all experiments the sensor nodes are dispersed at random locations over a rectangular area. The maximum packet size of communication is set to 32 bytes. The energy consumption while transmitting

and receiving data is modeled according to [20]. In particular, transmitting b bits of data to a node that lies at a distance $dist$ from the current node results in an energy drain of: $(E_{TX} + E_{RF} \times dist^2) \times b$, where E_{TX} denotes the per bit power dissipation of the transmitter electronics and E_{RF} denotes the per bit and squared distance power delivered by the power amplifier. Similarly, receiving b bits of data results in an energy drain of: $E_{RX} \times b$. The values of these parameters are set similarly to [20] as: $E_{TX} = E_{RX} = 50nJ/bit$ and $E_{RF} = 100pJ/bit/m^2$. The default values for the `CacheSize` and `TestInterval` parameters was set to 10 and 6, respectively. In all runs, the maximum size of the memory cache is 4 KB.

We also model the channel loss. In our experiments, the transmitted messages have a 10% probability of requiring a retransmission due to message loss or collision. The header of each packet and the space to represent a node identifier is set to 2 bytes. The required space for the reading of a sensor and for the aggregate value is set to 4 bytes. We compare the quality of the reported aggregate and the required energy and bandwidth consumption of our techniques against a SELECT * query, which can be used to collect all measurements at the `Root` and perform the outlier elimination and aggregate computation there, and a standard aggregate query (termed as AGGREGATION) like the ones performed by TAG, without outlier detection. We also compare our techniques against the *Robust* algorithm presented in [15]. Since the *Robust* algorithm does not handle group-by queries, many of our experiments involve aggregate queries without a group-by clause. As mentioned earlier, contrary to our techniques, the *Robust* algorithm uses a fixed minimum support value of 1 for defining outliers. We ran all experiments 5 times and present here the median values.

A. Evaluating the Reorganization Algorithm

In order to better assess the benefits of our algorithm for reorganizing the collection tree, we explore the following synthetic setup. We first generate a large sensor network of 400 nodes and define 10 classes of data to control the behavior of the sensors. The readings of nodes that belong to the same class make random walks with different steps, and at the same direction. That is, whenever one node increases its value, all the nodes in the same class also observe higher measurements. The nodes are assigned to classes as follows. Each node initially belongs to the default class 0. We then generate 9 events at random locations and assign all nodes within horizontal distance 15 from the centers of the events to belong to the same class (classes 1 to 10). Thus the classes define vertical partitions of the space. In Figs. 2 and 3 we show the resulting bandwidth and energy consumption for a minimum support of 1,2 and 3. The standard deviation of the results in these two figures is about 8.3% of the presented numbers. In Fig. 2 (please note that the X axis lies at the top) the bandwidth consumption has been partitioned based on the bandwidth required to transmit the aggregates, outliers, witnesses and statistics in each algorithm. The collection tree reorganization is performed every 300 epochs and its overhead is included in the graphs (we account for this cost only in our method). We zero all the counters (bandwidth and energy consumption) immediately after each reorganization to better demonstrate the differences before and after the collection tree reorganization. From these figures we observe that:

1. During the first 300 epochs, the difference between the *Robust* and the `SensibleAggr-supp`1 algorithm is due to: (i) the flexibility of our techniques to handle missing (NULL) values in the cache; and more importantly (ii) the proper selection of witness nodes that is performed in Line 29 of the `SensibleAggr-supp` algorithm. Because the *Robust* algorithm performs the selection in a random way, many witness tests fail due to an insufficient number of readings in the cache of the nodes.

2. The collection tree gradually improves, as more statistics are collected, but mainly in the first reorganization of the collection tree. This is expected, since the classes are not modified during the query execution and the statistics are sufficient to make good decisions at the first reorganization. Fig. 2 shows that the improvement is due to the reduction in the number of the transmitted outliers. For example, the bandwidth consumption for the outlier sets in the SensibleAggr-supp1 algorithm during the initial 300 epochs is 61% higher than in the last 300 epochs. This reduction of bandwidth consumption increases (decreases) in absolute terms (in relative ratio) for larger support values. The overhead for the transmitted outliers cannot be completely eliminated due to the leaf nodes of the collection tree (this is why the bandwidth consumption for the aggregate is lower than in AGGREGATION).

3. It is important to note that *Robust* consumes, after the tree reorganization, more bandwidth than the `SensibleAggr-supp3` algorithm that reports more outliers due to the higher number for `MinSupp`. The progressive decrease in the bandwidth consumption for our algorithms results in an equally important decrease in the consumed energy by the sensor nodes (Fig. 3). For example, at the last period of 300 epochs, *Robust* consumes 43% more bandwidth than `SensibleAggr-supp`1.

4. The cost for the reorganization statistics was minimal in all cases (less than 0.4% of the total bandwidth consumption).

5. The SELECT * technique results in up to 5 times more transmitted bits and energy consumption than our techniques.

In Fig. 4 we plot the total bandwidth consumption during the last reorganization period (last 300 epochs), for a similar setting to the previous experiment, when we scale the number of sensor nodes within the same area (thus increasing the density of the network). The corresponding standard deviation rates for this experiment are about 9.2%. It is important to note that the bandwidth savings that our `SensibleAggr-supp1` algorithm achieves, when compared to the *Robust* and SELECT * algorithms, increase with the number of sensors. In fact, the *Robust* and SELECT * algorithms require up to 37% and 6.5 times, correspondingly, more bandwidth than `SensibleAggr-supp`1. The smaller absolute values of the bandwidth consumption when compared to Fig. 2 are due

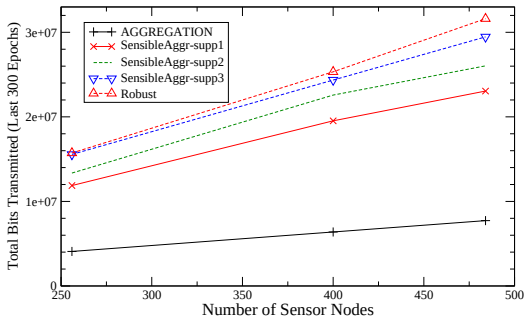

Fig. 4. Total bandwidth consumption in final reorganization period

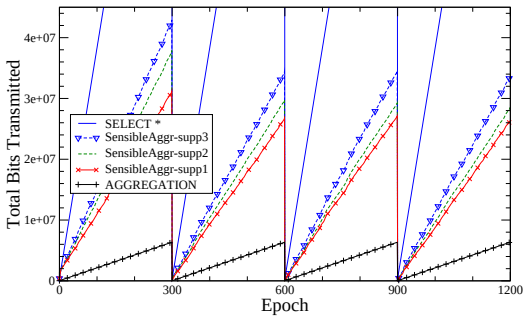

Fig. 5. Cumulative bandwidth consumption since last reorganization in synthetic dataset, group-by query

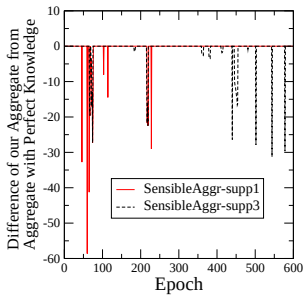

Fig. 6. Difference of Fig. 7 aggregate from aggregate with perfect knowledge

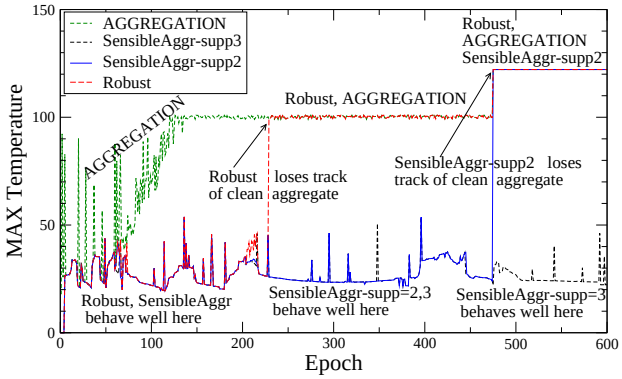

Fig. 7. Computed MAX Temp, Noisy Intel data, Extended Jaccard Coefficient

to the increased connectivity of the sensor nodes in this experiment (i.e., deployment within a smaller area).

In Fig. 5 we repeat the first experiment of this Section and plot the cumulative bandwidth consumption since the last tree reorganization (or the initial construction for the first 300 epochs). However, in this experiment we also: (i) specify a group-by clause that partitions the network into 4 quadrants; and (ii) specify the CONSTRAIN TEST GROUP clause. Please note that the *Robust* algorithm cannot be used to answer such queries. One can see that this case is qualitatively similar to Fig. 2. We also note that reorganization of the collection tree has a greater impact, especially for larger values of `MinSupp`. The reduction in the overall bandwidth consumption is up to 27%, while the reduction in the transmitted outlier bits reaches 53%. This is because a random initial selection of the parent nodes may disperse readings of nodes that belong to the same group towards different directions. Thus, reorganization is more important when group-by queries are considered. This fact is also recognized in [17].

B. Experiments with Perturbed Real Traces

We now investigate how our techniques perform in challenging scenarios where sensor nodes frequently obtain spurious measurements, or fail dirty due to the environment where they are placed. We use temperature measurements that involve 48 motes from the Intel Labs data set [8]. In this data set, one of the sensor nodes fails dirty at some point, increasing its temperature until it reaches 122 degrees. In this experiment, we increase the complexity of the data set by: (1) Specifying for each sensor a 6% probability that it will fail-dirty at some point. Each node that fails-dirty increases its measurement at an average of about 1 degree per epoch, until it reaches a maximum reading of about 100 degrees. To prevent its measurements from lying on a straight line, we also impose a noise of up to 15% at the values of a node that fails dirty; (2) Each node with probability 0.4% at each epoch obtains a spurious measurement, which we model as a random reading between 0 and 100 degrees.

In Fig. 7 we show the resulting reported aggregate for this very challenging data set. In this experiment we examine an alternative technique for computing similarity, namely the extended Jaccard coefficient [23]. We note that this change requires only the modification of the `canWitness()` function in our framework. The similarity threshold in this experiment was set to 0.8. The reported maximum temperature of our `SensibleAggr-supp1` algorithm was similar to the one of *Robust*, so we depict only the latter in the Figure, for clarity. Fig. 7 shows that *Robust*, which uses a minimum support of 1, quickly leads to disappointing results, after 220 epochs. Using our techniques, with increased minimum support, improves the situation. In this experiment, most of the abnormal readings are eliminated using a minimum support of 3.

In Fig. 6 we compare the aggregate reported by our techniques against a technique, termed *oracle*, that collects all the data at the `Root` node (using a SELECT * query) and then applies our `SensibleAggr-supp` algorithm there. Most of the times the reported aggregate of our in-network technique is exactly the same with the one reported by the oracle. Please note however that even the oracle can perform very poorly for low values of minimum support, as its results for `MinSupp`=1 are similar to the ones of *Robust* in Fig. 7. In Fig. 10 we show the F measure [21] which is calculated as: $\frac{2 \times recall \times precision}{recall + precision}$ of our in-network algorithm when using a minimum support value of 3, compared to the oracle based on the list of outliers reported. We plot the results for the standard version of our algorithm, where each successful witness test of an outlier node with a node in the WitnessSet conservatively increases the support of the outlier by 1. We also plot two more aggressive alternatives that increase the outlier's support in such cases (parameter $PropagateSupport$ in the figure) by 2 and 3, correspondingly. We see that our `SensibleAggr-supp` algorithm achieves very high values of the F measure, meaning that it discovers most of the outliers also discovered by the oracle with few misclassifications,

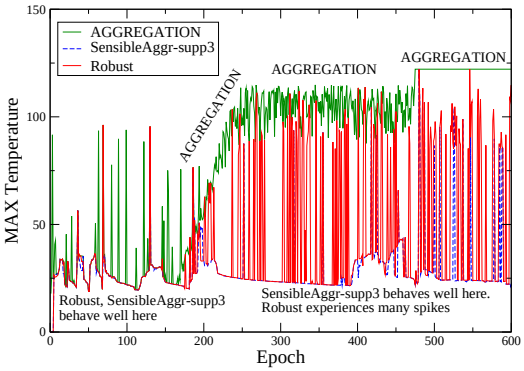

Fig. 8. Computed MAX Temp., Intel data with noise, Correlation Coefficient

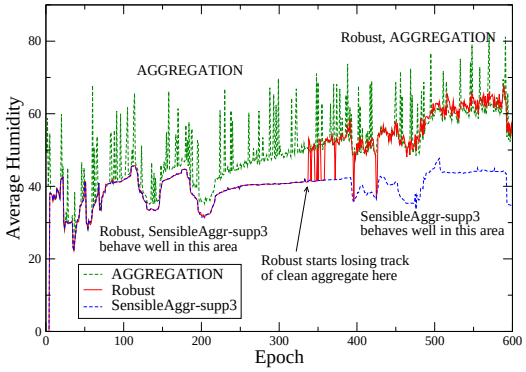

Fig. 9. Computed AVG Humidity, Intel data with noise, Correlation Coefficient

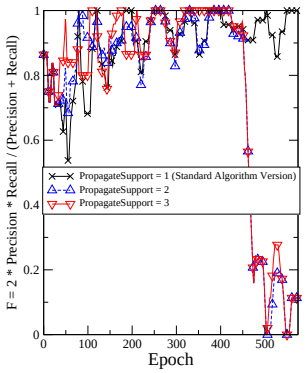

Fig. 10. F measure of SensibleAggr-supp3 in Fig. 7

while using only a fraction of the bandwidth and energy consumed by the latter (please refer to our discussion in the previous subsection where we evaluate the cost of a SELECT * query). Moreover, we observe that we are correct in utilizing our standard, conservative approach, as the most aggressive versions provide disappointing results near the end of the experiment, where the data has become more noisy.

In Fig. 8 we show the resulting reported aggregate when using the correlation coefficient. As we can see, the aggregate computed by pure in-network aggregation quickly becomes meaningless. The technique of [15] provides some improvements, but is still characterized by too many spikes. The aggregate obtained by our technique with a minimum support of 3 is significantly more accurate and manages to eliminate most spurious readings, along with the readings of nodes that fail-dirty, in all but a few cases. In Fig. 9 we experiment with perturbed humidity readings from the same real dataset. This time we compute the average humidity value. We notice that utilizing a minimum support of 3 provides very good results and manages to eliminate from the aggregate computation the measurements of the nodes that failed dirty in this experiment.

VII. Conclusions

In this paper we presented a novel aggregation framework that can tolerate outlier readings that almost always arise in sensor network applications. Our framework supports aggregation queries with group by predicates. We considered different definitions of an outlier node, based on a specified minimum support by other nodes over a period of time, and discussed techniques that optimize the routing of messages in the network in order to minimize the bandwidth and energy drain during the query evaluation while maintaining the quality of the aggregate. Our experiments with real traces establish that a straightforward evaluation of an aggregate query leads to highly inaccurate and, thus, meaningless results due to the existence of outliers. In contrast, our approach detects and eliminates spurious readings without any application specific knowledge of what constitutes normal behavior, and can indeed report comparable performance to out-of-network computation of outliers and aggregates with significant reduction in energy and bandwidth consumption.

References

[1] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating Aggregates on a Peer-to-Peer Network. Technical report, Stanford, 2003.

[2] J. Chen, S. Kher, and A. Somani. Distributed Fault Detection of Wireless Sensor Networks. In *DIWANS*, 2006.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *ICDE*, 2004.

[4] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: the Count-Min Sketch and its Applications. *J. Algorithms*, 55(1):58–75, 2005.

[5] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing Historical Information in Sensor Networks. In *ACM SIGMOD*, 2004.

[6] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical In-Network Data Aggregation with Quality Guarantees. In *EDBT*, 2004.

[7] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Bandwidth Constrained Queries in Sensor Networks. *The VLDB Journal*, 17(3):443–467, 2008.

[8] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB*, 2004.

[9] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidermann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *ICDCS*, 2002.

[10] S. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Pervasive*, 2006.

[11] S. Jeffery, M. Garofalakis, and M. Franklin. Adaptive Cleaning for RFID Data Streams. In *VLDB*, 2006.

[12] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *FOCS*, 2003.

[13] N. Khoussainova, M. Balazinska, and D. Suciu. Towards Correcting Input Data Errors Probabilistically using Integrity Constraints. In *MobiDE*, 2006.

[14] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. In *ICDE*, 2005.

[15] Y. Kotidis, A. Deligiannakis, V. Stoumpos, V. Vassalos, and A. Delis. Robust Management of Outliers in Sensor Network Aggregate Queries. In *MobiDE*, 2007.

[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.

[17] A. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks. *VLDB Journal*, 2004.

[18] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware Routing in Mobile Ad Hoc Networks. In *International Conference on Mobile Computing and Networking*, 1998.

[19] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online Outlier Detection in Sensor Data Using Non-Parametric Models. In *VLDB*, 2006.

[20] H. Tan and I. Korpeoglu. Power Efficient Data Gathering and Aggregation in Wireless Sensor Networks. *SIGMOD Record*, 32(4), 2003.

[21] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[22] Y.-J. Wen, A. M. Agogino, and K.Goebel. Fuzzy Validation and Fusion for Wireless Sensor Networks. In *ASME*, 2004.

[23] X. Xiao, W. Peng, C. Hung, and W. Lee. Using SensorRanks for In-Network Detection of Faulty Readings in Wireless Sensor Networks. In *MobiDE*, 2007.

[24] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3):9–18, 2002.