Robust Management of Outliers in Sensor Network Aggregate Queries [*]

Yannis Kotidis
Athens Univ. of Economics and Business, Athens, Greece
kotidis@aueb.gr

Antonios Deligiannakis
University of Athens
Athens, Greece
adeli@di.uoa.gr

Vassilis Stoumpos
University of Athens
Athens, Greece
stoumpos@di.uoa.gr

Vasilis Vassalos
Athens Univ. of Economics and Business, Athens, Greece
vassalos@aueb.gr

Alex Delis
University of Athens
Athens, Greece
ad@di.uoa.gr

ABSTRACT

Sensor networks are increasingly applied for monitoring diverse environments and applications. Due to their unsupervised nature of operation and inexpensive hardware used, sensor nodes may furnish readings of rather poor quality. We thus need to devise techniques that can withstand "dirty" data during query processing. In this paper we introduce a robust aggregation framework that can detect and isolate spurious measurements from computed aggregate values. Such readings are not injected in the reported aggregate, in order not to obscure the outcome, but are still maintained and returned to the user/application, which may investigate them further and take appropriate decisions. In addition, our framework provides a form of positive feedback to the user by enhancing the result with a set of nodes that contain the most characteristic values out of those included in the aggregation process. We perform an extensive experimental evaluation of our framework using real traces of sensory data and demonstrate its utility to the monitoring of applications.

1. INTRODUCTION

Sensor networks are increasingly being introduced in monitoring applications and/or conditions in diverse physical environments. These networks typically consist of primitive wireless nodes that are able to "sense" their environment, produce readings, perform simple operations and, if needed, relay their results to other sensors nearby. Such exchanges are possible through the selective use of radio frequency. A flurry of recent papers has tackled the problem of efficiently answering declarative queries in such networks. The majority of these efforts focus on answering aggregate queries that are of great importance to surveillance applications [19, 26]. By and large, these efforts try to exploit *in-network processing* by combining individual sensor readings as they are communicated towards a *base station*. This results in orders of magnitude fewer transmissions and, thus, prolonged network lifetime compared to a naive execution that collects all readings at a central point where aggregation is performed. A shorter, but equally important, line of research deals with the realization that sensor readings are inherently dirty [14, 15, 23]. A measurement obtained by a node is only an approximation of a physical quantity observed by the application and is constrained in accuracy by the limited hardware that often implements the sensing function. Moreover, due to the unattended nature of many applications and the inexpensive hardware used in the construction of the nodes, sensor nodes often provide imprecise individual readings after a failure, i.e., they tend to be *fail dirty* [14]. Thus, we are faced with the daunting challenge to build applications on top of an architecture that is, at times, unreliable and unpredictable.

In this paper, we take a step towards providing a resilient query processing platform over a network consisting of cheap, wireless devices that are prone to failures. We argue that one cannot simply accept all readings taken from the nodes, because many of them are of very poor quality. For instance, after examining real traces of sensory data from Intel Labs [9] we observed that simple aggregation queries like "find the maximum temperate by all sensors in the room" can deviate by as much 400% from the real values because of dirty readings. Thus, a naive implementation of a query processor that ignores such issues renders the network infrastructure practically ineffectual. On the other hand, it is very inefficient to gather all readings outside the network in order to identify suspicious measurements. Even if this is done only periodically, it would still require a lot of messages and would cancel out the benefits of in-network data aggregation. Moreover, we should not forget that sensor networks find their cause when they are called to observe interesting phenomena. Therefore, we should be very careful when characterizing their observations, because what may look like a spurious reading may as well be the prelude to an interesting phenomenon. For example, a surprisingly high temperature reading in the execution of the previous query may in-fact indicate a possible fire. Thus, it would be unwise to reject measurements simply because they look suspicious.

We aim at extending the notion of in-network aggregation as our proposal offers the effective means for dealing with dirty and/or unexpected measurements. We introduce a query execution model

[*]This work was supported by a European Social Funds and National Resources Pythagoras Grant with No. 56-90-7410 and the Univ. of Athens Research Foundation.

that, on par with the aggregates, also recognizes and reports to the user a concise set of readings that are believed to be outliers. We introduce a novel notion of an outlier node, motivated by real applications of sensor networks, that is not based on individual readings. Instead our framework looks for correlations between sets of readings from multiple sensors in order to properly classify them. Readings from outlier nodes are not injected in the computed aggregate, in order not to distort the outcome, but are still maintained and returned to the user/application. Users may investigate outliers further and take appropriate decisions. For instance, if a single sensor is continuously reported as an outlier, the network administrator may investigate it and determine that it has in fact failed and needs to be removed from further consideration during query processing or repaired, if possible. Our execution model, in addition to the list of outliers, further enhances the result with a set of nodes that contain the most characteristic values out of those included in the aggregation process. These nodes are called witnesses and provide a form of positive feedback to the user that may help him better comprehend the set of measurements that shape the result to its aggregate query.

Our contributions are as follows:

- We present a new resilient execution paradigm for computing aggregate queries in sensor networks. Our techniques dynamically separate outliers from the outcome of the aggregation. These unexpected readings are preserved and reported to the user that issued the query along with the estimate of the aggregate. Our enhancement of the aggregate computation allow us to return more trustworthy results to the users.

- In addition to the list of outliers, our framework may also return a set of characteristic values that have been used to derive the requested aggregates. This form of *positive feedback* allows the user to better comprehend observations made by the nodes that may otherwise be blurred during the aggregation process.

- We perform an extensive experimental evaluation of our framework using real traces of sensory data. Our experiments demonstrate the utility of our aggregation framework enhanced with outlier detection to the monitoring applications.

The identification of outliers could be of tremendous value in a number of real-life situations, such as the monitoring and detection of:

1. *faulty home sensing devices:* as home automation strives to offer a highly controlled and adaptable environment, the timely identification of isolated faulty devices plays a key role in both saving energy and reducing costs.

2. *tremors following earthquakes:* it is often the case that the follow-up activity of major earthquakes proves more devastating in terms of lives and property lost. Detecting and accurately reporting deviations in measurements taken from large extends of land after the occurrence of severe physical phenomena, may help authorities better understand their nature and deploy their rescue resources more effectively.

3. *resurgence of bush-fires:* recurring severe weather conditions frequently assist in the re-establishment of fires taking place in large extends of land. Continual monitor and reporting of further activity is a must after such a catastrophic event.

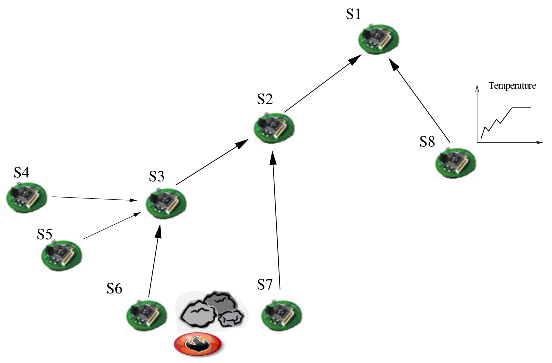

Figure 1: Sample Aggregation Tree

4. *breaches computing systems security:* organization and companies routinely monitor their key infrastructure including servers, desktops, routers, bridges, and incident detection devices. Operators often make the final decision on whether a spurious activity takes place once they review operational aggregated data reported by the individual elements of infrastructure.

This paper proceeds as follows: In Section 2 we present a motivational example for our problem and briefly describe the intuition of our techniques. In Section 3 we formally introduce our framework. Section 4 provides additional details on our algorithms and potential extensions. Section 5 includes our experimental evaluation. Section 6 presents related work, while cocluding remarks are presented in Section 7.

2. MOTIVATING EXAMPLE

Consider a query that computes the average temperature in the area covered by the sensornet depicted in Figure 1. 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 aggregation tree 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 and propagate these values to its own parent [19, 26]. For instance, node S_1 receives two pairs of aggregates from its direct children $(sum_2, 6)$ and $(sum_8, 1)$ where sum_2 is the summation of the temperature readings from nodes S_2,S_3,..., S_7, while sum_8 only contains value x_8 from sensor node S_8. Node S_1 can easily compute the requested average value from these statistics.

We notice that nodes S_6 and S_7 can both observe an open fire and, therefore, their readings are expected to be a lot higher (and fluctuate more) than for example those of node S_2. Since node S_6 is a lot closer to the fire than node S_7 we further assume that its temperature readings x_6 will be larger than those of node S_7. We also observe that nodes S_6 and S_7 do not directly communicate because of an obstacle in their path. Because of that, the readings of node S_6 are propagated to node S_2 through intermediate node S_3. When node S_3 receives the values of its children nodes, the readings x_6 of node S_6 appear to be suspicious, since no other node in that subtree is aware of the fire. If one is to reject that reading (for instance using a voting protocol [3]), the monitoring application will lose a crucial observation. Techniques based on smoothing [14, 15] will also obscure the outcome, especially if a lot more nodes are rooted at node S_3.

Our solution to this impasse is to tentatively put the reading of node S_6 in a special list of outliers. There are many ways to define the notion of an outlier. While we defer the discussion of outlier detection to the next section, we briefly note here that our framework seeks correlations among sensor readings in order to classify (or not) a particular node as an outlier. This list of outliers will be communicated from children nodes to their parents, but the readings that it contains will not participate in the computation of the aggregate.

Now let us concentrate on node S_2. This node will receive from the left subtree a pair of values $(x_3+x_4+x_5, 3)$ from node S_3 and the list of outliers containing only the value x_6. The right-subtree contains a single node S_7 that reports the pair $(x_7, 1)$. In absolute terms the value of x_7 is smaller than that of node x_6. However, an oracle could ague that both readings refer to the same physical event. Thus, node S_7 acts as a *witness* to node S_6 and vise versa. Therefore, neither of their readings should be considered as outliers and both values can be included in the aggregate computation. In addition, it would be beneficial to retain the readings from one of these two nodes into a list of *witnesses* that will play a role that is symmetrical to role of outliers: If there exists another node higher in the aggregation tree that also reports a fire, then we need to be careful as to not to characterize its readings as outliers. This could be achieved by testing its recent values against the nodes found in the list of witnesses.

In addition to values that were initially deemed outliers, but for which that decision was later overturned, the witnesses list also contains characteristic values that participate in the aggregation process. For instance when the readings from nodes S_4 and S_5 are consolidated, one of them is retained as a witness. This witness will be utilized in order to judge the readings of nodes S_3 and S_2 when the partial aggregates are computed. Thus, the list of witnesses provides supporting evidence for the aggregate as the latter is compiled at the intermediate nodes of the aggregation tree.

Back to our example, sensor node S_8 demonstrates the case of a node that fails dirty. The node starts reporting abnormal high readings that cannot be justified by any of its neighbors or the values that are provided in the witness list. Thus, the values of x_5 should be kept out of the computation of the average temperature at node S_1 but still need to be reported in order to alert the user/application.

Compared to a conventional computation of the query aggregate, one could see that in our alternative framework three main pieces of information are propagated upwards in the aggregation tree. First, the requested aggregate. Second, a list (actually a set) of witnesses that helps us properly decide whether the readings of a new node should be part of the aggregation. Last, a list of outliers, which are re-evaluated as they are forwarded upwards in the tree.

The above example raises several interesting questions:

- How may one rightfully characterize the readings of a node as outliers? As we have indicated above, local voting schemes may not return correct results if the node does not agree with the majority in its neighborhood.

- Sensor readings may look different simply because of the nature of the observations. Node S_6 that is closer to the fire reports higher temperature than node S_7. In another example of acoustic readings, the values will also depend on the distance of each node from the source. Thus, we need to devise more intuitive solutions than, for example, relying on simple norm computation like $|x_i - x_j| \geq \epsilon$ [23].

- The judgment on the readings returned by the nodes may change as more knowledge is accumulated at the higher levels of the aggregation tree. In the previous example, the reading of node S_6 was initially regarded as abnormal but that decision was later retracted.

- It is crucial that detection and bookkeeping of the lists of witnesses and outliers should not impose significant overhead on the participating nodes and that these lists remain compact, so as not to increase communication cost.

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
x_i	The current reading of the i-th sensor node
W_i	The set of witnesses that node S_i transmits to its parent node
O_i	The set of outliers that node S_i transmits to its parent node
T_i	The set of nodes contained in the subtree routed at node S_i, including S_i
a_i	The partial aggregate from subtree T_i excluding the outliers in O_i

Table 1: Symbols Used in our Algorithm

3. FRAMEWORK DESCRIPTION

We now present more details on our query execution model. The notation used is described in Table 1. The queries we consider are aggregation queries of the form:

```
SELECT AggrFun(s.value)
FROM Sensors s
WHERE cond
SAMPLE PERIOD e FOR t
```

As in [19], the aggregate functions that we consider are distributive or algebraic functions such as MAX,MIN,COUNT,SUM,AVG. The period (e) in the above query determines the frequency at which data is acquired from the sensors. This is often referred to as the epoch duration. Parameter (t) specifies the life span of the query. Unless it is explicitly terminated by the user, the query will run for t time units. We assume that the aggregation tree has been established using techniques already presented in the literature [12]. During the query execution, each node S_i in the tree receives, using a protocol like TAG [19], partial aggregates from its children nodes. These values are combined with its own reading (if the node contributes to the query) and result in a new partial aggregate that is communicated to its own parent. This is a paradigm of in-network computation and has been well established by prior work [7, 19, 26].

In this paper, we extend the in-network computation framework as follows. In addition to the partial aggregate a_i computed over its subtree, node S_i also transmits two sets:

W_i: This is a set of witnesses taken from nodes that provide measurements for the query. These nodes must belong to the subtree rooted at node S_i.

O_i: This set contains nodes that provide measurements for the query and are deemed to be outliers or faulty. These nodes must belong to the subtree rooted at node S_i.

Both W_i and O_i are subsets of T_i (the set of nodes under the subtree containing node S_i) and their intersection is the empty set. For instance, in our running example of Figure 1, set W_3 contains only one of the nodes S_3, S_4 and S_5, while the set O_3 contains node S_6.

Our definition of an outlier is that of a node whose readings are not like any other node that participates in the query. More formally, let $\mathcal{S}$ be the set of nodes that provide their measurements for the query and let $P(S_k, S_l)$ be a predicate that evaluates to true when the readings from nodes S_k and S_l are similar. We define node S_k as an outlier when $P(S_k, S_l)$=*false*, $\forall S_l \in (\mathcal{S} - \{S_k\})$. Details on predicate P will be given in Section 4.1. As it will be explained shortly, predicate P requires a few readings from nodes S_k and S_l. These readings are communicated along with the id of nodes in sets W_i and O_i.

Let us assume that node S_j is a child node of S_i in the aggregation tree. In our proposed framework, S_j will process readings from children nodes (and its own) and communicate to its parent node S_i a triplet (a_j, W_j, O_j). We distinguish the following cases:

- Node S_j is a leaf node. In this case, it needs to reports its own measurement as an outlier since there is no evidence of other nodes with similar readings. Of course, this selection may change later by nodes along its path to the root. There is no partial aggregate a_j or witnesses W_j to report at this stage. Thus, the node transmits a triplet containing (*null*,$\emptyset$,$\{S_i\}$) to its parent node S_i.
- Node S_j is an intermediate node in the aggregation tree. In that case, it transmits a triplet (a_j,W_j,O_j) to node S_i where a_j is the partial aggregate for the subtree rooted at S_j, excluding nodes in O_j that are believed to be outliers.

Upon receiving all triplets (a_j,W_j,O_j) from its children, node S_i first initializes W_i to $\cup_j(W_j)$, the union of all witnesses received by its children nodes. Then, the node needs to decide which of the readings in $\cup_j(O_j)$ are outliers or not, considering its whole subtree, and appropriately compute the aggregate value a_i. First, for each child node S_j, it examines the set of nodes contained in O_j. Let S_k be a node in O_j. If there exist S_l in

$$\{S_i\} \cup (\cup_{m \neq j}(W_m)) \cup (\cup_{m \neq j}(O_m))$$

such that $P(S_k, S_l)$=*true*, then node S_k is inserted to set W_i, otherwise, the node remains an outlier and is included in O_i. This means that a node that was originally deemed as an outlier by S_j (or its descendants) may be added in the aggregate if it is similar to at least one node in T_i. We do not need to check the nodes in T_j because this check has already been performed by S_j when S_k was inserted in the set O_j. A similar test is performed by S_i for its own measurement: it checks whether there is at least one other node in $\cup_j(W_j)$ that is similar to S_i.

The partial aggregate a_i that is computed by S_i needs to contain[1]:

- All partial aggregates a_j that have been reported by its children. Null values are ignored.
- The values of nodes $S_k \in O_j$ that have been decided that are not outliers by the previous process.
- Its own measurement, if S_i contributes to the query and there exists at least another node with a similar behavior in its measurements.

[1]In case of AVG or more complex aggregates, a_i may be a struct, not a single scalar value.

In the end of this process, node S_i transmits (a_i, W_i, O_i) to its own parent and this process continues towards the Root node. It is easy to see that if a node S_k does not belong to the list of outliers O_i then its measurement has been included in the partial aggregate a_i by a node S_m that is either S_i or some other node in T_i. In either case, S_k was among the lists of outliers received by the children of S_m (or S_k=S_m) and was found similar to a node in T_m. Thus, if a node contributes to the aggregate a_i then it is not an outlier in T_i.

In the previous discussion we assumed that the set W_i is initially compiled from the union of the nodes in sets W_j, where node S_j is a child of node S_i in the aggregation tree. This simplification introduces some redundancy as it is quite possible that two nodes S_k, S_m that belong to sets W_{j1} and W_{j2}, respectively, are in-fact similar ($P(S_k, S_m)$=*true*). In that case, we do not need both nodes as witnesses and it is sufficient to include only one of them in W_i. Thus, when inserting a node to set W_i we make a test whether there is another similar node. If this is true then no action is taken. Note that this is not required when inserting nodes to the set of outliers O_i, since by definition two outlier nodes cannot be similar. Furthermore, when a node is moved from a set of outliers O_j because of a successful similarity test with a node in W_i, then this node should not be added to the set W_i, since another similar node already exists there.

4. OPERATIONAL ISSUES

4.1 Detection of Outliers

One can provide several definitions of an outlier, depending of the application. For example in [23], an outlier is defined as an observation that is sufficiently far from most other observations in the data set. We already saw that such a definition is inappropriate for physical measurements whose absolute values depend on the distance of the sensor from the source of the event that triggers the measurements. In our framework, we have separated the logic used for detecting an outlier from the mechanisms of our query evaluation protocol. This separation may allow one to consider different principles for detecting outliers with only minor modifications in our framework.

In this paper we take note of several recent publications that have shown that many of the physical quantities observed by sensor nodes in monitoring applications are naturally correlated. This observation has been exploited in order to reduce energy drain during data gathering by either compressing the readings taken from each node [5] or reducing the need to access many of the nodes in the network by exploiting model-based retrieval techniques [9, 18]. Motivated by this observation, we propose here a simple yet powerful technique based on the *correlation coefficient* among the readings of two sensor nodes. 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

$$r_{k,l} = \frac{cov(x_k, x_l)}{\sigma_{x_k}\sigma_{x_l}} = \frac{E(x_k x_l) - E(x_k)E(x_l)}{\sqrt{E(x_k^2) - E^2(x_k)}\sqrt{E(x_l^2) - E^2(x_l)}} \quad (1)$$

where $cov()$, σ and $E()$ stand for the covariance, standard deviation and expected value respectively. If x_k and x_l are independent then the correlation coefficient $r_{k,l}$ is zero. The closer $r_{k,l}$ is to 1, the stronger the positive correlation between the two readings. Since, in our framework, the nodes observe the same physical quantity, is it natural to expect that the values should not be independent, unless an unusual event is observed by one of the nodes or one of them is faulty. Given a threshold θ provided by the application and communicated to the nodes during query initialization, we define predicate P that is used to separate outliers from normal readings as

$$P(S_k, S_l) = (r_{k,l} \geq \theta) \quad (2)$$

4.2 Implementation Aspects

In order to compute coefficient $r_{k,l}$, we need to have available recent measurements from nodes S_k and S_l. The most recent measurement of a node in sets W_j and O_j is communicated along with the id of the node in each epoch. Node S_i that receives these sets, needs to store these measurements in its memory while the query is running. We note that one would not want to use the complete history of measurements from both nodes in order to compute the correlation coefficient, since we tend to focus more on recent readings than older observations. In our implementation, we use the last M readings from each of the nodes in order to determine the value of the coefficient. Parameter M depends on the sensors used. Since memory capacity in modern sensors tends to increase rapidly, it is reasonable to assume that for small values of M (i.e., 8, as in our experiments) this approach is feasible even in todays low-end sensors.

Furthermore, one would want to have a limit on the size of sets W_j and O_j, not only to reduce memory usage, but also in order to reduce communication cost. In most radio models used in todays sensors, the energy drain during transmission/reception of a message is proportional to the number of bits included [25]. Thus, we can limit the communication cost by allowing each set to contain up to k pairs of the form (S_{id},x_{id}). A natural question is how to handle situations when more that k witnesses/outliers are needed. A naive solution is to only report the first-k nodes in each set and ignore the remaining ones. The drawback of this approach it that we may lose important observations. A workaround is to transmit these "overflow" values in subsequent epochs. Recall that aggregate queries in sensor networks do not seek an one-time result. On the contrary they are continuous queries that run for predefined periods or until they are explicitly terminated. Thus, if the size of say set W_j exceeds k we can transmit the first k values in the current epoch, and opportunistically patch the remaining values in subsequent epochs in which fewer than k values need to be transmitted. In order to properly correlate aggregates with witnesses and outliers we may want to include epoch-ids along with transmitted values that are delayed. Furthermore, "overflow" nodes in both sets that appear both in the current epoch and previous ones may only be reported once in order to reduce the number of bits transmitted.

4.3 Extensions

Detecting faulty patterns: Thus far, we have made the implicit assumption that measurements from sensor nodes that have failed dirty follow a random pattern and, thus, are expected to behave as white noise. The consequence is that readings from two or more sensors that fail dirty are statistically independent and will, thus, correctly be included in the list of outliers. Depending on the hardware used in implementing the sensing element of a node, this assumption may not hold in practice. For instance, the temperature sensor used in the MICA2 motes shows a predictable pattern of failure: its temperature readings increase until they reach a maximum value and then stay there (see node S_8 in Figure 1). In such cases, our model may incorrectly characterize two or more faulty sensors as statistically correlated and include their values in the computed aggregate. In order to avoid such issues, we propose the following extension to our framework. Each node is instructed with a special predicate that will be used to detect readings from faulty sensors; this is orthogonal to the predicate used in detecting outliers. The new predicate may test the latest readings from a node against a predefined pattern, such as the one described above for temperature sensors. It may also perform simple sanity checks such as asserting that a reading is above or below predefined thresholds such as minimum and maximum expected temperature. Nodes that fail these tests are immediately put in the set of outliers and flagged appropriately so that they cannot be removed from the set by other nodes in the path to the root.

Limiting Scope of a Witness. A possible extension to our framework is to restrict the set of candidate nodes that can play the role of a witness to sensor node S_i. For instance, assuming that sensor nodes are used in a tower building to measure temperature we may want to consider a node S_j as a witness to node S_i only if both nodes are in the same floor or in adjacent rooms etc. These extensions can be easily handled in our framework without changing the core of our algorithms.

Using *MinSupport* for Outlier Detection: Based on our definition of an outlier a node needs a single witness so at not to be considered an outlier. Depending on the application we may want to control the number of witnesses required through a *MinSupport* parameter and extend the definition of an outlier to that of a node that has fewer than *MinSupport* witnesses (our previous definition is equivalent to using *MinSupport* with a value of 1). This extension only requires a simple modification to our framework. Along with the id S_k of a sensor node in O_i, we also include the number of nodes n_k that are currently witnessing S_k. When n_k reaches or exceeds *MinSupport* the node is removed from the list of outliers.

5. EXPERIMENTS

We have developed a simulator for testing various types of aggregate queries [6]. For the experiments that we present in this paper, we fed the simulator with a trace of readings from sensors in the Intel Research, Berkeley lab, collecting light, humidity and temperature readings [9].[2] We used trace data from 48 sensors for a period of 7 hours. For setting up the aggregation tree, we used the aggregate connectivity data available with this trace. Unless specified otherwise, in all of our experiments we used a threshold value of 0.9 for the correlation coefficient and a buffer of 8 readings for computing this coefficient (i.e., a sensor could witness another node if the correlation coefficient of their last 8 readings was at least 0.9). The size of the buffer needs to be of moderate size. A small buffer size (i.e., 2) does not include enough history about the sensor's past measurements and would often cause many sensors to exhibit a constant value within the buffer. On the other hand, a very large buffer size would result in increased memory and processing

[2]We would like to thank the authors for making their data publicly available.

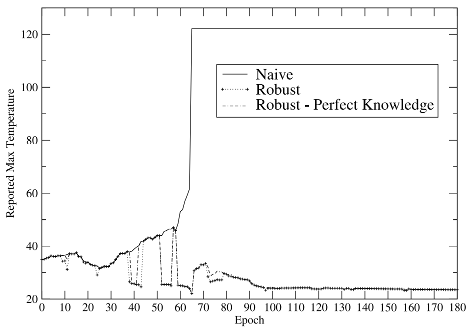

Figure 2: Reported Max Temperature over Time

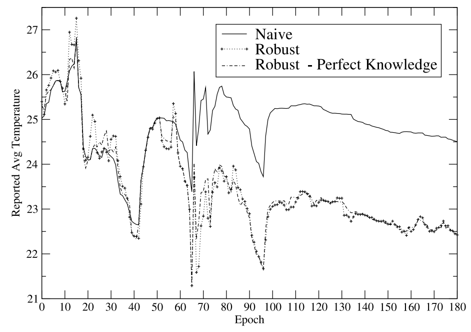

Figure 3: Reported Avg Temperature over Time

cost. By performing a sensitivity analysis, we discovered that using a buffer size between 4 and 10 provided qualitatively similar results.

In all experiments we report the reported aggregate value for three algorithms. The *Naive* algorithm denotes the typical in-network aggregation performed by protocols like TAG [19]. The *Robust* algorithm denotes our technique for computing and reporting the aggregate value along with a set of witnesses and outliers. Finally, the *Robust-perfect knowledge* algorithm denotes an ideal version of our algorithm that is performed on the base station and which has perfect knowledge of all the current and previous readings from all the sensor nodes.

In Figures 2 and 3 we depict the reported maximum and average temperature readings collected by the sensor nodes for all algorithms. As it is evident from these figures, our algorithms managed to identify and eliminate from the computation of the aggregate sensors with unique patterns, such as a faulty sensor node that increased its reading up to a maximum value of about 122 degrees Celsius. What is more important is that the identification of such nodes is performed inside the network without any application specific knowledge of the *normal* range (which is not always easy to define - i.e., consider the case when a fire may erupt inside a lab or a machinery part overheats) of values for the readings of the sensor nodes. In Figure 4 we plot the corresponding minimum humidity reading collected by the sensor nodes for all algorithms. Results

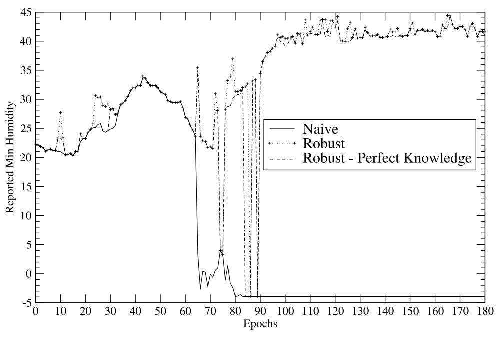

Figure 4: Reported Min Humidity over Time

	Threshold Value						
Measure	0.60	0.65	0.70	0.75	0.80	0.85	0.90
MAX Temperature	3	1	1	1	0	0	0
MIN Humidity	8	8	5	3	2	2	2

Table 2: Epoch lag to Eliminate Faulty Sensor Readings from Aggregate Varying the Threshold for the Correlation Coefficient

for the average readings are similar as before and are omitted due to space constraints. Again, our algorithms managed to detect and eliminate from the computation of the aggregate abnormal humidity readings, in all but a few epochs.

A question that naturally arises is how does our method perform when the threshold used for the correlation coefficient is lowered. As expected, the lower the value of the threshold, the more likely that a node can witness some other sensor and, thus, the lower the number of identified outliers. If a sensor starts generating a unique pattern of behavior, it may take a few epochs for our techniques to identify it as outlier, as a single reading may not be sufficient for the witness test with other nodes to fail. The lower the value of the threshold, the larger the number of readings involving the new unique pattern that are required for the algorithm to identify and eliminate the readings of the node. In Table 2 we present the number of epochs that our algorithms required in order to eliminate the apparently wrong readings, depicted by the sudden shifts in the reported aggregate by the Naive algorithm in Figures 2 and 4, for different values of the threshold for the correlation coefficient. The presented results are easily explained based on our above discussion. We can clearly observe from Table 2 that a threshold value of 0.75 to O.8 results in a rapid elimination from the computed aggregate of the abnormal sensor readings. Please observe that the number of wrong reported aggregates does not exceed the required number of measurements for computing the correlation coefficient amongst the readings of two sensor nodes.

In Table 3 we present the number of transmitted bytes per sensor node at each epoch for the calculation of the average temperature. For our algorithm we report four values, based on the limit that we imposed, in each case, on the maximum number (1, 2 ,3 or ∞) of transmitted outliers and witnesses by each node. The header of each packet was set to 8 bytes, while the size of each node identifier and of each quantity (count, sum) required for the calculation of the aggregate was set to 4 bytes. We can observe that even in

	Algorithm					
	Robust Aggregation				NAIVE	SELECT *
Threshold	1	2	3	Infinite		
0.60	22.0	23.2	23.8	24.1	16	35.2
0.65	22.4	23.6	24.1	24.5	16	35.2
0.70	22.9	24.0	24.5	24.9	16	35.2
0.75	23.4	24.3	24.8	25.1	16	35.2
0.80	23.9	24.9	25.4	25.7	16	35.2
0.85	24.9	25.8	26.2	26.4	16	35.2
0.90	25.9	26.7	27.1	27.4	16	35.2

Table 3: Bandwidth Consumption in Bytes per Sensor and Epoch by our Algorithms

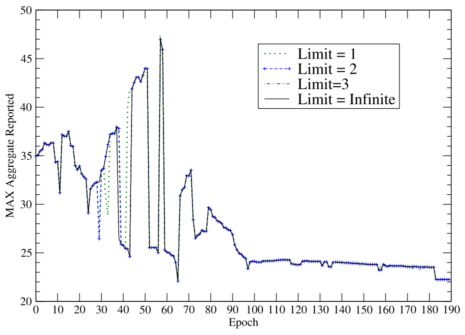

Figure 5: Reported Max Temperature Varying the Maximum Number of Transmitted Witnesses/Outliers

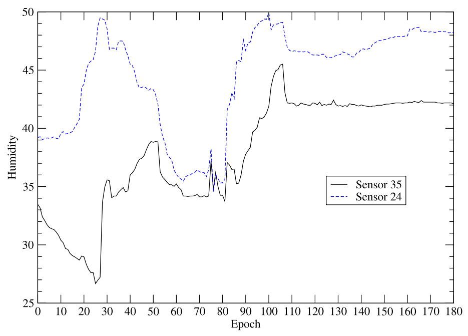

Figure 6: Humidity Readings by Sensors that are Often Witnesses

this small number of sensor nodes, where the SELECT * query imposed only a small increase in the bandwidth consumption over the NAIVE aggregation, our algorithms achieved significant gains in the bandwidth consumption. In Figure 5 we depict the reported max aggregate for the same four versions of our algorithm. In this small sensor network, using a limit of 3 transmitted witnesses and outliers per node yields about the same results as using an infinite limit. For a limit of 1 or 2, a few additional spikes in the reported aggregate are observed.

A final note involves the *positive feedback* that witnesses provide to the base station. In Figure 6 we depict the humidity readings of the two sensor that were most often reported as witnesses. We first notice that their behavior differs significantly. When the readings of one sensor rise, the readings of the other sensor often decline. The user thus becomes aware of these different behaviors, something which could have been missed by simply looking at the overall aggregate value. Please note that amongst nodes with similar behavior only one can be picked as a witness, so this dissimilar behavior by the two most common witnesses is expected.

6. RELATED WORK

Recent work has demonstrated the feasibility of using an embedded database management system for data acquisition in sensor networks [26, 19]. Using a declarative SQL-like query language for posing monitoring queries in such sensor networks provides far greater flexibility than injecting hand-coded programs at each sensor node [20]. Among the various types of queries that have been discussed in the literature, aggregate queries have received the bulk of attention [7, 19, 21, 26]. In [19] the nodes are first organized in a tree topology, termed the aggregation tree. During the query execution, each epoch is subdivided into intervals and parent nodes in the aggregation tree listen for messages containing partial aggregates from their children nodes during pre-defined time-slots. The rest of the time the nodes may power-down their radios in order to reduce their energy and bandwidth consumption. Another notable method for synchronizing the transmission periods of nodes is the recently proposed wave scheduling approach of [8]. Alternative techniques do not utilize an aggregation tree, but rather compute aggregation queries using decentralized algorithms [1, 16].

In the networking community there is substantial effort in resolving the networking aspects of wireless sensor networks that need to operate unattended for long periods of time. Thus, nodes must be able to self-configure [2], discover their surrounding nodes [10, 11] and compute energy-efficient data routing paths (such as the aggregation tree [12]) [22, 24]. All these techniques are complementary to our work, as they provide the necessary network primitives that we need to build our messaging scheme. In [4], a novel framework that uses sketching techniques for compensating for packet loss and node failures during query evaluation is proposed. Complementing our framework in order to take advantage of the work in [4] is an interesting future direction.

Recently, techniques that use either global [9] or local [18] data models in order to avoid gathering all readings from nodes that participate in a posed query have been proposed. Such techniques can dramatically decrease the cost of running a continuous aggregate query in large networks. Our techniques can benefit from data modeling. For instance one can implement our framework directly over the network snapshot proposed in [18]. The work in data modeling has demonstrated that many of the physical quantities observed by sensor nodes in monitoring applications are naturally correlated. This observation has been exploited in compression schemes that encode such correlations using linear regression [5]. In our work we capitalize on these observations and propose a novel technique for capturing outliers based on the *correlation coefficient* amongst the readings of sensor nodes.

Many recent publications have 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. In [13] 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 [14] 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 [15] a probabilistic technique for cleaning RFID data streams is presented. The work in [17] discusses a framework for correcting input data errors using integrity constraints. Our methods differ from these techniques in that we do not try to "mask" abnormal readings, but instead promote them into first class citizens, on par with the requested aggregates, and make them available to the monitoring application. The work in [3] addresses the problem of identifying faulty sensors using a localized voting protocol. In section 2 we saw that local voting schemes are prone to errors when nodes that observe interesting events are not in direct communication. Furthermore, while [3] requires a separate costly process that needs to run periodically in order to evaluate the network, our framework allows us to capture faulty as well as outlier nodes in real time during query evaluation and piggyback this information on messages used in query processing. Similarly, the work in [23] computes outliers in sensor networks in a manner that is decoupled from query processing and can thus lead to significant energy drain on the nodes.

7. CONCLUSIONS

In this paper we introduced a robust aggregation framework that can tolerate outlier readings that often arise in sensor network applications. Unlike prior work, our techniques do not attempt to hide or clean outliers, which can often be the prelude to an interesting phenomenon, but rather maintain and present them to the user, along with a set of witnesses that take part in the computed aggregate result. What is important is that the identification of such readings is performed inside the network, without any application specific knowledge of what constitutes normal behavior. Our experiments with real traces from sensor nodes demonstrate that our method is very effective in detecting outlier nodes which, if left untreated, would significantly blur the outcome of the aggregation process.

8. 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] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In *INFOCOM*, 2002.

[3] J. Chen, S. Kher, and A. Somani. Distributed Fault Detection of Wireless Sensor Networks. In *DIWANS*, pages 65–72, 2006.

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

[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*, 2007.

[8] Alan Demers, Johannes Gehrke, Rajmohan Rajaraman, Niki Trigoni, and Yong Yao. The Cougar Project: A Work In Progress Report. *SIGMOD Record*, 32(4):53–59, 2003.

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

[10] D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *MobiCOM*, 1999.

[11] J. Heidermann, F. Silva, C. Intanagonwiwat, R. Govindanand D. Estrin, and D. Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *SOSP*, 2001.

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

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

[14] S.R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Pervasive*, pages 83–100, 2006.

[15] S.R. Jeffery, M.N. Garofalakis, and M.J. Franklin. Adaptive Cleaning for RFID Data Streams. In *VLDB*, pages 163–174, 2006.

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

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

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

[19] 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.

[20] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *ACM SIGMOD*, 2003.

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

[22] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *ACM/IEEE International Conference on Mobile Computing and Networking*, 1998.

[23] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online Outlier Detection in Sensor Data Using Non-Parametric Models. In *VLDB*, pages 187–198, 2006.

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

[25] N. Trigoni, Y. Yao, A.J. Demers, J. Gehrke, and R. Rajaraman. Multi-query Optimization for Sensor Networks. In *DCOSS*, 2005.

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