An Energy-efficient Querying Framework In Sensor Networks For Detecting Node Similarities

Daniela Tulone
MIT CSAIL
tulone@csail.mit.edu

Samuel Madden
MIT CSAIL
madden@csail.mit.edu

ABSTRACT

We propose an energy-efficient framework, called SAF, for approximate querying and clustering of nodes in a sensor network. SAF uses simple time series forecasting models to predict sensor readings. The idea is to build these local models at each node, transmit them to the root of the network (the "sink"), and use them to approximately answer user queries. Our approach dramatically reduces communication relative to previous approaches for querying sensor networks by exploiting properties of these local models, since each sensor communicates with the sink only when its local model varies due to changes in the underlying data distribution. In our experimental results performed on a trace of real data, we observed on average about 150 message transmissions from each sensor over a week (including the learning phase) to correctly predict temperatures to within +/- 0.5°C.

SAF also provides a mechanism to detect *data similarities* between nodes and organize nodes into clusters at the sink at *no additional communication cost*. This is again achieved by exploiting properties of our local time series models, and by means of a novel definition of data similarity between nodes that is based not on raw data but on the prediction values. Our clustering algorithm is both provably optimal in the number of clusters and very efficient. Our clusters have several interesting features: first, they can capture similarity between far away nodes that are not geographically adjacent; second, cluster membership to variations in sensors' local models; third, nodes within a cluster are not required to track the membership of other nodes in the cluster. We present a number of simulation-based experimental results that demonstrate these properties of SAF.

Categories and Subject Descriptors: C.2.4 [Distributed Systems]: Distributed Databases

General Terms: Management, Performance

Keywords: Data collection, sensor networks, energy, data stream, query, models.

MSWiM'06, October 2–6, 2006, Torremolinos, Malaga, Spain.

1. INTRODUCTION

There has been a great deal of interest in recent years in developing systems to collect data from wireless sensor networks. Applications include environmental monitoring [21], agriculture [5], industrial monitoring, and process control [1]. Several systems (e.g., Cougar [22], TinyDB [20], and directed diffusion [16]) for querying such networks have been developed. These existing systems typically collect data from all of the nodes in a network at a regular rate to a *sink node*, where readings are combined and processed just as in a standard streaming database system.

Much recent work has applied statistical modeling techniques to approximate query answering in sensor networks [11, 8, 17, 25]. However, most of these approaches [11, 8] require a large amount of communication between the sink and each sensor during an initial learning phase where models are built. This expensive learning phase means that these models do not adapt well to variations in the statistical distribution of data at a node, since variations may require the model to be recomputed. Unfortunately, such variations commonly occur in physical phenomena of the sort sensor networks are often used to monitor. In contrast, in this paper we propose an approximate querying framework based on time series models that have an inexpensive learning phase, making our approach very adaptable to changes in the distribution of data at sensors.

In our framework, called SAF (for Similarity-based Adaptive Framework), queries are answered using lightweight linear time series models built by each node from a small number of readings (enabling models to be quickly re-learned) and stored at the sink. Sensor nodes and the sink only communicate occasionally to exchange models or answer queries that require more accuracy than the stored models can provide. We show that SAF is capable of detecting outlier values (isolated data anomalies), periods of data instability, and data similarities among sensor nodes. Our approach to similarity detection works even under dynamic conditions (e.g., node mobility, data distribution variation, or unstable communication channels).

SAF uses simple linear time series models that consists of a time-varying function T_r, called *trend component*, and a *stationary autoregressive* (AR) component $X(t)$ representing the divergence of the phenomenon from T_r over time. We chose this model because, as shown in our experimental results, it is capable of predicting data produced by real-world sensors measuring physical phenomena like ambient light, temperature, and humidity that evolve slowly over time, is

computationally tractable on modern-generation sensor networks, and is cheap to learn. In contrast with other approaches [11, 8], nodes learn these models locally (requiring no communication), whenever a novel *monitoring* algorithm we have developed detects that the local model is no longer a good fit for the data – that is, it is *invalid*. When this occurs, the sensor relearns the model and transmits its coefficients to the sink. SAF When the AR models SAF uses are valid, the distribution of sensor readings is stationary (invariant over time), which means that SAF does not require periodic readings to provide a given accuracy as in PAQ [25]. Stationarity allows SAF to provide other interesting properties such as the ability to detect outliers and periods of *data instability* (highly noisy data).

In addition, we can use SAF to detect data similarity between sensors, and group them into clusters using the model coefficients stored at the sink. Clustering has a number of applications in sensor networks, including intrusion and anomaly detection (when one sensor is not behaving like others it was previously clustered with), redundancy analysis (determining that multiple sensors in a particular location are providing no additional information as they are always similar to each other), and various kinds of physical modeling (observing, for example, that temperatures or soil moisture levels in a particular region are similar over a range of time may be useful to domain scientists.) Grouping sensor nodes into clusters under dynamic conditions is a challenging problem requiring continuous monitoring and adaptation of cluster membership. This problem has been studied also in the context of query answering [25, 8]. However, in these proposals the communication cost involved in computing and maintaining clusters can be high, especially under dynamic conditions. Unlike previous methods, our approach to detecting node similarities requires *no additional* communication. Our method detects similarities at the sink and not at the sensor nodes, and it relies on a novel definition of data similarity, which is based not on raw data as in previous approaches but on prediction values. More precisely, we detect node similarity based on the bounds derived from the models stored at the sink.

We propose a very efficient and *provably correct* clustering algorithm based on our definition of similarity. The main merit of our method consists of transforming the complex problem of computing and maintaining clusters that evolve over time into the simple 1–dimensional problem of grouping intervals of equal width. Our algorithm is *provably optimal* in the number of clusters and it has running time $O(n \lg n)$, where n is the number of nodes. This clustering algorithm has the further benefit that it does not require nodes in the same cluster to be geographically co-located, and does not require nodes to communicate at all with each other, thus making the clusters highly adaptable, at no additional cost. These features make our approach well-suited to *mobile networks*, offering a simple solution to the difficult task of computing and maintaining clusters despite the rapidly changing network topology that typically characterizes mobile networks.

In summary, the main contributions of our work are:

- We propose a query framework based on time series forecasting for approximately answering user queries and detecting *outliers* and *data inconsistency*. This is achieved using novel bounds, and enhanced models and algorithms with respect to PAQ. We present a detailed simulation-based study of our methods running on a trace of data collected from a real sensor network.
- We study the problem of detecting and tracking data similarities under dynamic conditions. We propose a simple, provably correct and optimal method that is able to detect node similarities under highly dynamic conditions at no additional communication cost. Our method is built on top of a novel definition of data similarity based on data models and on a clustering algorithm that is provably optimal in the number of clusters and very efficient.

Summary of the paper. We describe our system model in Section 2, and illustrate our time series model and the monitoring and adaptation algorithms in Section 3. We illustrate and analyze our centralized query framework in Section 4, and introduce our notion of data similarity and propose and analyze our clustering algorithm in Section 5. In Section 6 we present detailed simulation results regarding the stability and the efficiency of our centralized query framework and our clustering algorithm. In Section 7 we compare our work with previous query approaches in sensor networks and other related work, and conclude. We have omitted proofs from this paper due to space limitations; we refer the reader to [24] for details.

2. SYSTEM MODEL

A SAF network consists of a collection $\mathcal{S}$ of sensor nodes and one or more sink nodes. Each node is equipped with some sensing capability, performing readings on m physical sensors, $F_1, F_2, \ldots, F_m$, each of which evolves over time. For example, we might say that F_1 =temperature, and F_2 =light. We assume that each node obtains a reading from each F_i once every Γ time units. We have designed our framework to work with a range of sensor nodes, including low-end devices like the Berkeley Motes [10], with just a few kilobytes of memory and slow, 8-bit processors. Note that sensor clocks are not required to be synchronized since nodes work asynchronously. Nodes in a sensor network are organized in a multi-hop topology, with several radio hops between any pair of nodes in the network. We organize the network into a *routing tree*, rooted at the *sink*, which is connected to a gateway PC that can be used to communicate with the user. In this paper, we focus on answering queries of the form, which are issued from this gateway:

```
SELECT sensorlist
WHERE P(F_1, ..., F_m) ERROR x
```

where $\mathrm{P}(F_1, \ldots, F_m)$ is a predicate over $F_1, \ldots, F_m$ consisting of atoms of types $F_i \in [a, b]$, $F_i > a$, and $F_i < b$ where a, b are user-specified. Here `ERROR` x denotes that the user is tolerant to a maximum absolute error in the query result. For example, the user might issue the query `SELECT nodeid, temp WHERE temp` $> 25^{\circ}$`C ERROR` $.1^{\circ}$`C`, which would report the temperature at each node to within $.1^{\circ}$C. To simplify our presentation we consider queries over one measurement F, though our results generalize in a straightforward manner.

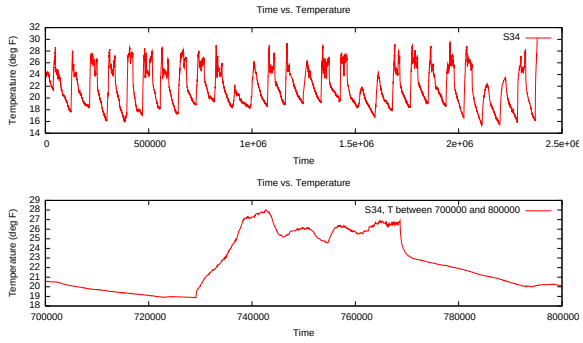

Figure 1: Temperature at sensor 34 over 26 days, and details from one day.

3. LOCAL FORECASTING MODEL

In this section we describe our model, which is computed and maintained at each sensor node, and the algorithms run by each node for learning its model and monitoring its quality.

3.1 Our model

Physical phenomena are usually complex and require sophisticated prediction models with a long learning phase in order to produce an accurate prediction. However, it is impractical for many low-end sensor network nodes to build such sophisticated models, due to their high computational cost and memory requirements. Our idea is not to model the physical phenomena in its entirety but only within a time interval W of a few hours, using a simple model. This relies on the observation that in most cases physical phenomena change relatively slowly within a time window W of few hours. Hence, our choice of the model is driven by the following metrics: (1) *efficiency* in terms of computational cost and memory requirements; (2) *duration of the learning phase*; (3) *accuracy* of the predictions; (4) model *stability* (duration of the time window W).

Our sensing model. We model F as a stationary time series model. Specifically, we consider an *autoregressive* model AR(q), a subclass of ARMA models whose prediction is given by a linear combination of the previous q values. A time series is *weakly stationary* if its mean $E(X_t)$ does not depend on t, and its covariance $Cov(F_{t+h}, F_t)$ is independent of t for any $h \in \mathbf{Z}$. The reader is referred to Brockwell and Davis [4] for a detailed discussion of ARMA modeling.

We assume that each node reads F every Γ time units, and denote the history of these values as $v_1, \ldots, v_i, \ldots$. In our previous work [25] we modeled F as a weakly stationary zero-mean AR model plus a constant (the mean value over the training sample). This model is suitable for cases in which the phenomenon does not vary rapidly. In case of a rapid increase or decrease of the phenomenon's values within an hour, such a model requires frequent updates at higher communication cost. For example, Figure 1 shows the temperature read during 26 days by a sensor node placed inside a building in the presence of air conditioning, and a snapshot of 5 hours. Note that the activation/deactivation of the air conditioning causes rapid changes in the temperature – for example, around time = 750,000, the temperature suddenly drops as a result of AC activation. To reduce the need to relearn models in the presence of such instability, we enhance our previous model by including a temporal function that models a linear trend in the data during a time window W. As a result, we model phenomenon F as follows:

$$F(t) = m_t + X(t)$$

where $F(t)$ denotes the real value of F at time t, m_t denotes the value of the trend component at time t, and $X(t)$ is a *weakly stationary* AR(q) time series [4]. In our discussion we consider a linear trend component $m_t = a + b \cdot t$, where a and b are real constants. $X(t)$ is a weakly stationary zero-mean AR(q) time series, with a short prediction window q, which ensures cheap learning/re-learning and low memory requirements. In addition, using a small prediction window allows us to neglect seasonal components [4]. For simplicity of presentation, we assume $q = 3$ in the remainder of this paper. Our stationary component $X(t)$ is an AR(3) time series with *Gaussian white noise* of zero-mean and maximum standard deviation $b(\Gamma)$ between readings, as follows:

$$X(t) = \alpha\, X(t-1) + \beta\, X(t-2) + \gamma\, X(t-3) + b(\omega)N(0,1) \quad (1)$$

where α, β, γ are real constants. Since $X(t)$ is stationary, then $\alpha + \beta + \gamma < 1$. Note that function $b(\omega)$ represents the standard deviation of the white noise, and as a result it provides a measurement of the quality of our prediction model. It increases with the elapsed time ω since the previous reading, since the accuracy of the prediction usually degrades over time.

The predictor $P(t)$ of value $F(t)$ at time t is given by the value of the current trend m_t plus the predictor of $X(t)$, which is a linear combination of the increments or decrements of the last three readings with respect to their trend component (at time $t_{i-1}, t_{i-2}, t_{i-3}$). More precisely, the prediction of $F(t)$ at time $t_{i-1} < t < t_i$ is given as follows:

$$P(t) = m_t + \alpha(v_{i-1} - m_{t_{i-1}}) + \beta(v_{i-2} - m_{t_{i-2}}) + \gamma(v_{i-3} - m_{t_{i-3}})$$

We call $v_t - P(t)$ the *prediction error* at time t, provided v_t is the sensor reading at time t. The following lemma computes the error bound and error probability associated with $P(t)$. It follows from our assumption of a Gaussian white noise of zero mean and maximum standard deviation $b(\Gamma)$. We refer the reader to [24] for the analysis and further details.

LEMMA 1. Let $P(t)$ be the prediction of F at time t associated with model (1), and let $\varepsilon = \nu\ b(\Gamma)$, where ν is a real-valued constant larger than 1. Then, the actual value at time t is contained in $[P(t) - \varepsilon, P(t) + \varepsilon]$ with error probability at most $\frac{1}{\nu^2}$.

As a result, readings with a prediction error larger than ε are classified as anomalies with error probability smaller than ν^{-2}, and they are handled specially. This feature is crucial for monitoring and adapting the model, as we discuss below.

3.2 Monitoring and adapting algorithms

During the learning phase the sensor node performs a reading every Γ time units, and inserts each reading into a queue V of recent readings. After performing N readings, the node computes the coefficients a and b of the trend component based on the data contained in $V = \{v_{t_1}, \ldots, v_{t_N}\}$

by applying *least square regression*. That is, it computes the parameters a and b that minimize $\sum_{i=1}^{N}(v_{t_i} - m_{t_i})^2$. Then, it computes a queue D_v containing the difference of each value in V with from its predicted trend value, that is $x_{t_i} = v_{t_i} - m_{t_i}$ for each $x_{t_i} \in D_v$. The node uses the data contained in D_v to compute the coefficients α, β and γ of $X(t)$ by applying least-squares regression, and it computes the standard deviation of the white noise $b(\Gamma)$ between readings. Therefore, at the end of the learning phase the node has computed the model coefficients $a, b, \alpha, \beta, \gamma, b(\Gamma)$ that uniquely identify the model, and transmits them to the sink. The least-square regression, computation amounts to solving two systems of two and three unknowns (respectively) and is tractable on low-end sensor hardware. The impact of the size of the training sample V will be discussed in Section 6.

As mentioned above, our linear model must be adaptable in order to effectively predict non-linear phenomena. Hence, the node periodically monitors its local model and updates it as needed. This design allows the node to detect *anomalies* relative to the previous history of sensed data. We classify anomalies as either *outliers*, which are transient mispredictions that the model simply does not account for, or *distribution changes*, which are persistent mispredictions suggesting that the model needs to be re-learned, either because of a faulty sensor or a fundamental shift in the data being sensed. Note that nodes can locally re-learn and update their models without communicating with the sink or other nodes. In addition, this re-learning requires a small amount of data and has low computational and memory requirements.

Monitoring algorithm. We use a monitoring algorithm to track the quality of the predictions of the model. Each node starts monitoring its model just after the learning phase. It takes a reading every Γ time units and updates a queue V, which contains the most recent N values. The main tasks of the monitoring algorithm consist of detecting (1) variations in the data distribution, and (2) outlier values.

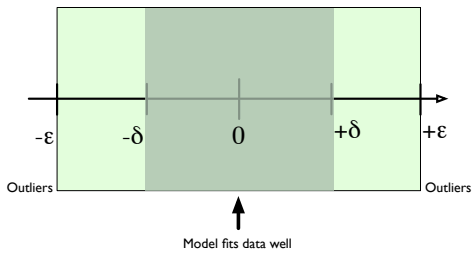

Figure 2: Monitoring prediction error.

Figure 2 graphically illustrates the monitoring algorithm, which relies on the assumption that the white noise follows a Gaussian distribution with zero-mean and standard deviation $b(\Gamma)$, and on Lemma 1. This lemma proves that a value whose prediction error (in its absolute value) exceeds ε does not belong to the data distribution with error probability smaller than ν^{-2}. However, in order to have a model capable of predicting the current data distribution and of detecting outliers, we consider an additional threshold δ where $1.2\ b(\Gamma) < \delta < \nu b(\Gamma)$. This parameter is used to detect when the model is beginning to be a poor fit for the data, as shown in Figure 2. If the prediction error falls outside $[-\varepsilon, \varepsilon]$ and no anomalies have been reported recently (i.e., the prediction error of previous readings was contained in $[-\delta, \delta]$), we classify that value as an *outlier* with error probability smaller than ν^{-2} according to Lemma 1, and monitor future readings because the model may no longer be a good fit. If the prediction error falls outside $[-\varepsilon, \varepsilon]$ and it is not an isolated anomaly, the node re-learns the model since it is probably no longer a good fit for the data distribution. For simplicity of presentation we avoid several technical details (see for more details [24]), and illustrate the three main cases that occur when monitoring the prediction error:

1. If the error prediction is contained in $[-\delta, \delta]$ the model is a good fit for the current data distribution. In this case the node simply updates its local queue V.

2. If it falls within $[-\delta, -\varepsilon] \cup [\delta, \varepsilon]$ then the model might mispredict the data. To determine that, the node opens a *monitor window* of size MW (if it is not open) during which it monitors the occurrences of these "partial anomalies". At the end of the monitor window the node relearns the model only if the number of these partial anomalies exceeds a given threshold (e.g., $\frac{MW}{2}$). The node computes the new model based on its local queue V as done at the end of the learning phase, and transmits the new coefficients to the sink.

3. If the prediction error falls outside $[-\varepsilon, \varepsilon]$, it is either an outlier valuer (isolated anomaly), or the data distribution has changed. If the model has been a good fit for the data so far, the node classifies that value as a suspected outlier, and opens a monitor window if it is not open. The node relearns the model at the second occurrence of an outlier and transmits the new model coefficients to the sink. Note that the node also follows these steps if it detects that the model is not stationary (a condition will be discussed in Section 4).

Adapting the model. As mentioned above, the monitoring algorithm can trigger a model update. In this case, the node computes the model coefficients based on its local queue V similarly to the initial learning phase. However, there are cases, although rare, in which the node is unable to compute a stationary component. This occurs if the data contained in V is inconsistent or if the data distribution has an abrupt change (e.g., in our data trace we reported occasional discontinuities up to 1-2°C between consecutive readings despite the typical differences being only a few tenths of a degree). In this case the data stored in V is inconsistent and it is not possible to compute a stationary $X(t)$ component. If the node is unable to compute a stationary model, it performs the following steps to improve model stability: (1) it removes data noise by applying an EWMA filter techniques to smooth outliers, and (2) it dynamically enlarges the size of the training set to decrease the effect of any outlier readings on the model.

If the model is still invalid, the node sends an invalid notification to the sink, and attempts to re-learn the model after a time interval called *invalid monitor window*. Note that during the invalid time window the local model is invalid, therefore the sink cannot use it to answer user queries, and if required it has to communicate directly with the node. However, in conditions of data inconsistency where values

differ unpredictably in a short time period, it might be more relevant for an application to know that the node is under unstable conditions (information provided by SAF) than to know the sensor values. For instance, knowing when data inconsistency occurred can be relevant for scientific studies to determine the cause of an event and possible correlations. In Section 6 we will discuss in depth the behavior of the invalid time and how the size of the invalid monitoring window affects the invalid time.

The following lemma provides a link between the model computed at the sensor node and the copy maintained at the sink, and it is useful to prove the correctness of our centralized query system. It follows from the monitoring algorithm and Lemma 1.

LEMMA 2. *The copy of the local model of node i stored at the sink is such that $v_t \in [P(t) - \varepsilon, P(t) + \varepsilon]$ with error probability at most ν^{-2}, provided the model is valid.*

4. OUR QUERY FRAMEWORK

In this section we describe our centralized framework for approximately answering user queries at the sink. This is achieved by ensuring consistency between the model in use at each sensor and its copy stored at the sink, and by providing a bound on the current sensor value. As discussed in Section 3, the time series $X(t)$ is a *stationary* zero-mean AR time series with Gaussian white noise. $X(t)$ represents the variation of the sensor reading at time t with respect to its trend component at time t. The stationarity of $X(t)$ allows us to bound $X(t)$ by applying Chebychev inequality since the mean of $X(t)$ and its standard deviation does not change over time (see [4]). Since sensor models are different, we denote the standard deviation of $X(t)$ at node S_i by σ_i, the standard deviation of the white noise at S_i by $b^i(\Gamma)$, and the trend value at node S_i at time t by m_t^i. The following lemma provides a bound of the value sensed by a sensor node at a given point in time.

LEMMA 3. *The reading of sensor S_i at time t is contained in $[m_t^i - (\varepsilon_i + \omega_i), m_t^i + (\varepsilon_i + \omega_i)]$, where $\omega_i = \kappa\sigma_i$ and $\varepsilon_i = \nu b^i(\Gamma)$, with error probability smaller than $\nu^{-2} + \kappa^{-2}$.*

The bound provided by Lemma 3 is crucial to prove the correctness of the query algorithm, and it allows the sink to approximately answer queries without communicating with the sensors or requiring periodic readings from each sensor to compute their prediction. As described in Section 3, each sensor S_i transmits its model coefficients to the sink at the end of the learning phase. The local models stored at the sink correspond to the current data distribution of each sensor, since each sensor immediately reports any change in its model to the sink by transmitting the new coefficients or an invalidation message. The sink responds to an invalid notification by adding the sensor node into a list U of nodes whose model is unstable. It removes that node from list U upon receiving the coefficients of the new model. The sink answers query Q of the form:

```
SELECT sensorlist
WHERE Q ERROR Q.e
```

For each node S_i, the sink checks if the node's model is stable and if the uncertainty (error) specified in the query $Q.e$ is larger than the current uncertainty provided by the model at S_i. If this is the case, the sink checks if the current trend value m_t^i at S_i satisfies condition $Q.cond$ and, if so, adds the predicted value to the list N of query answers. If the sensor is unstable or the accuracy provided by its model does not satisfy the query Q, the sink forwards the query request directly to the node.

Note that in order to make our centralized query answering scalable with respect to the number of sensor nodes, each node must send a notification to the sink each time the prediction error of a reading exceeds $\pm\varepsilon$ (including outlier values). In fact, if nodes do not notify outlier values, then the error probability associated with the query answer grows with the number of sensor nodes because of Lemma 3. Note that parameters ν and κ represent the stability of the model, since the node recomputes its model if the prediction error exceeds $\varepsilon_i = \nu\, b^i(\Gamma)$ or if $|X(t)| \geq \omega_i$ where $\omega_i = \kappa\sigma_i$. Therefore, larger values of parameters κ and ν correspond to more stable models. Clearly, there is a trade–off between stability and accuracy since the error bound increases as κ and ν increase. However, it is possible to use smaller parameters ν and κ in case the application requires high accuracy since each node monitors if $|X(t)| \leq \omega_i$ and $|v_t - P(t)| \leq \varepsilon_i$. As a result, our model can be used in two different ways:
(1) the parameters κ and ν can be fixed *a priori*, which implies that the model has a variable error bound ε_i that increases dynamically in the presence of data instability;
(2) the parameters κ and ν can be computed dynamically such that $\varepsilon_i + \omega_i$ meets the user accuracy bound, that is $\varepsilon_i + \omega_i$ is equal to the maximum query uncertainty.

The following lemma proves the correctness of our centralized framework.

LEMMA 4. *For each user query Q the query algorithm returns a list L of sensor nodes that correctly answer Q provided the nodes contained in L are alive.*

Note that since each node communicates rarely with the sink, nodes may fail silently without the sink noticing for some time. This issue can be addressed by having the sink periodically monitoring the status of each sensor node. It is important to note that SAF does not require *synchrony* among sensor nodes in contrast with [11, 8, 25] since the sink contacts only the nodes whose model is not sufficiently accurate. As a result, sensor clocks do not need to be synchronized.

5. DETECTING NODE SIMILARITIES

In this section, we describe and analyze our approach to clustering nodes using a novel definition of data similarity based on local models. Our notion of data similarity based on predicted values allows us to transform the difficult problem of computing and maintaining clusters under dynamic conditions into the simple problem of grouping intervals of equal width into a larger interval of width θ. Clustering has a number of uses in sensor applications including:

- Detecting redundant sensors, that is, sensors that can be moved to a new location or removed from the network because they provide little additional information over other sensors in their cluster.
- Intrusion and anomaly detection, where one sensor that was previously in a cluster with others is no longer in

the same cluster, indicating that some event has occurred that may merit the attention of the user.

- Modeling of physical phenomenon, such as soil moisture or temperature isolines, which are used in a variety of scientific domains.

We note that there is a large amount of prior work on data similarity and on time series similarity (see [15] for a survey of recent methods). However, previous work detects similarities using raw data. Such methods are undesirable in sensor networks since they require a large number of data transmissions from each node to the sink.

In contrast, our goal is to derive data similarities based on the local models which are stored at the sink, since this information is already maintained by the sink. To the best of our knowledge, detecting data stream similarities by means of their forecasting models, and in particular by means of their prediction values, is novel. We rely on this definition for building clusters at the sink, and for defining the compound AR model associated with each cluster.

5.1 Model-similarity definition

Before introducing our notion of data similarity between two sensor nodes, we define a temporal distance function d, called *prediction distance*, and our notion of θ-similarity based on d, where θ is a positive real constant.

DEFINITION 1. *The* prediction distance *between any two sensor nodes in S at some point in time, is a function $d : \mathcal{S} \times \mathcal{S} \times \mathcal{T} \rightarrow \mathbf{R}$ such that $d(S_i, S_j, t) = |P^i(t) - P^j(t)| \ \ \forall S_i, S_j \in \mathcal{S} \ \forall \ t \in \mathcal{T}$, where $P^i(t)$ and $P^j(t)$ are the predicted values of sensors i and j at time t, respectively.*

DEFINITION 2. *Sensor nodes S_j and S_k are* θ–similar *at time t, where θ is a positive real constant, if and only if $d(S_i, S_j, t) \leq \theta$.*

A subset of nodes $A \subseteq S$ is *θ–similar* at time t if any two nodes in A are θ–similar at time t. We group nodes that are similar into *θ–clusters*. Since clusters change over time according to the local data distribution, we define $\mathcal{C}_\theta(t)$ to be a set of subsets of $\mathcal{S}$, called *θ-clusters*, such that each $C \in \mathcal{C}_\theta(t)$ is θ-similar at time t, $\bigcup_{C \in \mathcal{C}_\theta(t)} C = \mathcal{S}$. Lemma 5 provides a criterion for detecting θ–similarities between two nodes at some point in time that relies on Lemma 3. We denote the lower bound $m_t^i - \omega_i$ of $P^i(t)$ at sensor node S_i, as l^i, and the upper bound $m_t^i + \omega_i$ as L^i. Moreover, we denote $\vartheta = \max\{L^1 - l^1, \ldots, L^n - l^n\}$. The following lemma says that given two nodes S_i and S_j, if the maximum distance between any point in $[l^i, L^i]$ and any point in $[l^j, L^j]$ does not exceed θ, then S_i and S_j are θ–similar.

LEMMA 5. *Given nodes S_i and S_j, if $|L^i - L^j| + \vartheta \leq \theta$, then S_i and S_j are θ–similar at time t.*

5.2 The clustering algorithm

The clustering algorithm, illustrated in Figure 3, is run at the sink and returns a list of θ-clusters. It is based on the fact that $P(t) \in [l_i, L_i]$ as shown in the proof of Lemma 3, and uses a provably-optimal greedy approach. The algorithm operates over a list of nodes' upper bounds of $P(t)$ sorted in decreasing order, $L^1 \geq \ldots \geq L^n$. As we show below, ordering the sensors in this way allows us to perform clustering in an efficient and optimal manner. The algorithm also maintains a set C of records, one per cluster. The record for the kth cluster, C_k, contains two fields: an upper bound, $C_k.v$, denoting the largest L value of any sensor in the cluster, and a set $C_k.set$ of the cluster's constituent members. We call $C_k.v$ the *pivot* of the cluster, and maintain it as we are processing the sorted list of upper bounds. We call the sensor node associated with $C_k.v$ the *pivot sensor*. For each *pivot sensor* the algorithm computes a maximal θ-cluster containing it.

```
clusters(θ, AR[1,...,n])
1)  D[1,...,n] ← parseData(AR[1,...,n])
2)  ϑ ← max_{1≤i≤n}{D[i].u − D[i].l}
3)  k ← 1
4)  C_k.v ← D[1].u
5)  C_k.set ← {D[1].id}
6)  for j = 2 to n do
7)      if (C_k.v − D[j].u + ϑ ≤ θ)
8)          C_k.set ← C_k.set ∪ {D[j].id}
9)      else
10)         k ← k + 1
11)         C_k.v ← D[j].u
12)         C_k.set ← {D[j].id}
13) return C_[1,...,k]
```

Figure 3: Clustering algorithm.

Specifically, the clustering algorithm receives as input the parameter θ, and an array $AR[1, \ldots, n]$ of n records such that each $AR[i]$ contains the coefficients of the AR model at node S_i. The algorithm returns a set of θ-clusters denoted by $C_{[1,\ldots,k]}$. It begins by invoking the procedure `parseData` (Fig 3:1) which returns an array $D[1, \ldots, n]$ of records, such that each record $D[i]$ contains the upper bound $D[i].u$, and the sensor node identifier $D[i].id$. `parseData` computes each upper bound L^i as $m_t^i + \omega_i$. It then creates the array $D[1, \ldots, n]$ and sorts it with respect to the upper bound values in decreasing order.

The cluster sets are progressively inserted into a list of records $C_{[1,\ldots,k]}$, where k indicates the number of clusters computed so far. $C_k.v$ is initialized when the cluster is created and $C_k.set$ is maintained as nodes in D are scanned. The pivot value $C_k.v$ is used to detect (in a greedy fashion) if the current sensor node (in the for loop) is θ-similar to each node in $C_k.set$. This is done by checking if the pivot value diverges from the lower value of the current node by no more than θ units, as in Figure 3:7. If a sensor node is not similar to the pivot sensor, then its upper bound becomes the new pivot value $C_k.v$ and its cluster's pivot $C_k.set$ is set to its identifier, as shown in Figure 3:10-12.

We have analyzed the clustering algorithm, proving its correctness and its optimality with respect to the number of clusters it computes, and providing bounds on its computational cost. We refer the reader to [24] for the analysis. The following corollary of Lemma 5 is used in proving the correctness of the clustering algorithm.

COROLLARY 1. *Let us suppose that $L^{i_1} \geq \ldots \geq L^{i_n}$ is the sorted list of $L^1, \ldots, L^n$ in decreasing order. If $L^{i_j} -$*

$L^{i_k} + \vartheta \leq \theta$, for $j < k$, then sensor nodes S_{i_j} and S_{i_k} are θ–similar provided their AR coefficients do not change and $\theta > 2 \max\{\varepsilon_1, \ldots, \varepsilon_n\}$.

LEMMA 6. (Correctness and Optimality) The clustering algorithm returns a list of θ-clusters. It is optimal in the number of clusters with respect to our criterion.

The computational cost of computing clusters is dominated by sorting the vector of upper bounds, and therefore is $O(n \log n)$ where n is the current number of sensor nodes.

5.3 Model vs. geographic clusters.

Other clustering approaches have been proposed for sensor networks (e.g., PAQ [25] and Ken [8]). Those approaches, however, grouped together geographically co-located sensor nodes (e.g., within their radio broadcast), rather than forming clusters irrespective of node location. We call those clusters *geographic clusters*. Our θ-clusters offer a number of advantages that are hard to achieved when using geographic clusters. We mention below some of these benefits:

Our model clusters are more general and capture stronger data similarities among sensors than geographical clusters as they detect similarities among far away sensors that are not geographically adjacent. This leads to a smaller number of clusters, reducing the amount of communication between sensors and sink.

Adapting local clusters to variations in the data distribution or sensor failures is a difficult and communication-intensive task for a distributed system which requires coordination among nodes. Our approach avoids these problems by relying on local models and computing clusters at the sink. This design allows sensors to be unaware of (1) their cluster membership, (2) variations in the cluster members' data distribution, and (3) variations in the user parameter θ. This allows the sink to quickly adapt to changes in any sensor's model.

Note that model clusters are resilient to *node mobility* since the notion of model-similarity does not depend on the geographic position of the sensors. Moreover, since they capture sensor similarities across the network, our clusters can be used to study *data correlations* among different geographical subregions in a large sensor network.

6. PERFORMANCE EVALUATION

In this section we discuss our experimental results based on a trace of real data, and we compare SAF and its features with other approximate frameworks [11, 8, 17, 25].

6.1 Simulation results

Our simulations were performed on a trace of real data from the Intel, Berkeley research lab (`http://db.csail.mit.edu/labdata/labdata.html`), which consists of about a month's worth of light, temperature, humidity, and voltage readings collected at a centralized base-station approximately every thirty seconds from 50 sensors. We have implemented our model and the algorithms described in Sections 3–5, and we have simulated the asynchronous behavior of the sensor nodes based on data points derived from that trace. The system and the simulator are written in Java.

Note that the sensors used to produce the data trace were placed inside a building equipped with air conditioning, whose activations caused inconsistencies in the data distribution (with an average duration of 20-30 minutes) and abrupt changes in the distribution of the data (e.g., rapid inversion in the trend). We choose this data trace since the irregularity of this data distribution provides a valid test for our framework. Note that the trace we used include a large number of missing readings because data was collected at a centralized sink and the Berkeley Motes [10] used to capture the readings were communicating over a lossy radio channel, with more than 60% missing data. Indeed we observed intervals of up to 7 hours of continuous missing data from some sensors. In order to simulate the periodic sensing performed at each node we applied linear interpolation with small random noise to infer the values of these missing readings. Note that several of those 50 sensors failed within a week, and only few of them lasted over a period of 26 days. In our experiments, when focusing on the behavior of the local model we chose primarily long-lived sensors (e.g., sensor 45), and considered sensor readings performed every 1 minute. We experimented with several other sensors and found the results to be similar to those shown here for sensor 45. The experiments reported here focus on the temperature values. Since the quality of the centralized model depends on the quality of our local models, we analyze in depth the behavior of our local model under the following metrics: (1) the total number of transmissions over the node lifetime; (2) the percentage of time during which a local model was invalid; (3) the accuracy of the model; (4) the stability of the model, that is the duration of the time interval during which the model is valid; (5) the number of similarity clusters computed by the sink.

SAF offers a number of trade-offs that can be tuned to meet application requirements, such as the trade-off between stability and error shown in Lemmas 1 and 3, or between transmissions and accuracy, or between model stability and invalid time in the presence of unstable data. We analyze our local models and discuss some of the most relevant trade-offs here. As mentioned in Section 4 our model can be used in two ways: the local model can have a fixed user-defined error, or a variable error that increases in the presence of data instability. To simplify our presentation we focus here on our model with a fixed error bound. We analyze the main factors that influence our model in the light of the metrics discussed above.

Training set. The training set usually has a clear impact on the quality of the statistical model: a larger training set typically leads to a more accurate model. However, this is not true in our model since it is designed to model a few hours of data, as discussed in the Introduction and in Section 3. In fact, a learning phase larger than 1 hour ($N > 60$) is usually not a good choice for our model (e.g., data collected over 2 hours can show seasonal components). This is because the monitoring algorithm will trigger more frequent model adaptations with larger training sets, and in the presence of data inconsistencies the model is likely to become invalid for a longer time when the training set is large (although our algorithm dynamically adjusts the learning queue in the presence of inconsistent data). In addition, we have observed that the error bound $\varepsilon + \omega$ of the model increases as the duration of the learning phase grows, at least for our mote data modeling temperatures. Figure 4 shows the variation in the total number of transmissions performed by node 45 over its

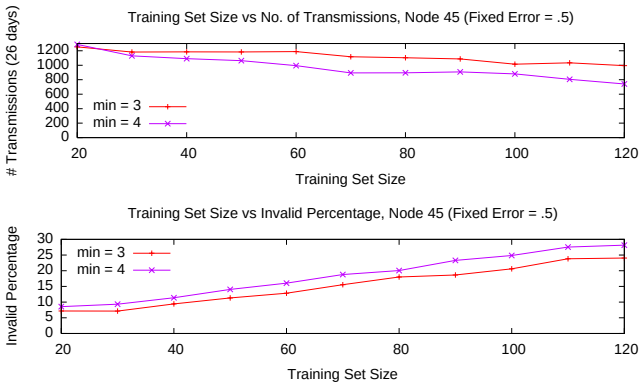

Figure 4: Training set vs. Invalid time and Transmissions.

system lifetime consisting of 26 days with respect to the size of the training set N, and the variation of the percentage of the total invalid time of the model with respect to the node lifetime. Note that N corresponds to the duration of the learning phase in minutes. These experiments were performed using a fixed error of 0.5 degrees, and an invalid time window of 5 minutes. The percentage of total invalid time grows as N grows, and this confirms what discussed above, while the total number of transmissions slightly decreases due to the increase in the invalid time during which there are no transmissions. Our analysis suggests that $N = 40$ is a good choice for such a this data, which we use in future experiments.

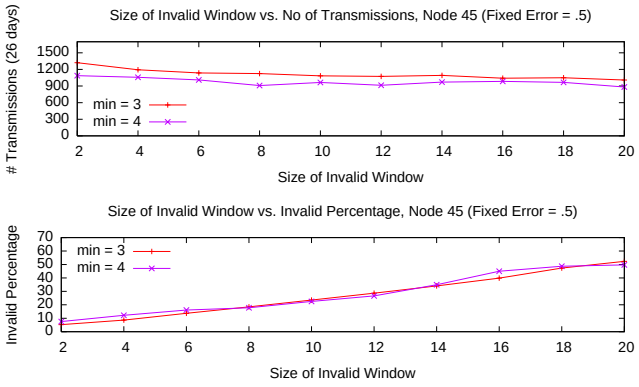

Figure 5: Invalid window vs. Invalid time, transmissions.

Invalid window. The size of the invalid window becomes relevant in case of data inconsistency (when the node is unable to adapt its model). If the node is unable to compute the model it waits for the duration of the invalid window and then attempts to recompute the model. As shown by our experimental results, illustrated in Figure 5 relative to sensor 45, the size of the invalid window has a clear impact on the percentage of the total invalid time of the model over the node lifetime. Figure 5 shows that the percentage of the total invalid time increases as the size of the invalid window increases. Also, in this case the increase of the invalid time window leads to a slight reduction in the total number of transmissions over 26 days. Our experiments suggest that a good choice for the invalid time window is 5, which we use in later experiments.

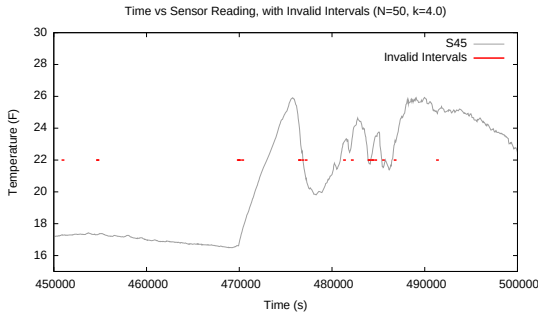

Figure 6: Invalid intervals overlaid on top of data. Invalid periods are indicated by the dark (red) dots along the central horizontal axis of the figure.

Intervals during which the model is invalid. Figure 6 shows the occurrence and the duration of time intervals during which the model could not be computed because of inconsistent data over a 42 hour period. We plotted these intervals during a 42 hour period on top of the temperature during that interval. To make the phenomenon more evident we showed a period of high data inconsistency and used a high stability parameter which increases the duration of the invalid time under inconsistent data. Note that instability typically occurs when the value of the sensor changes abruptly.

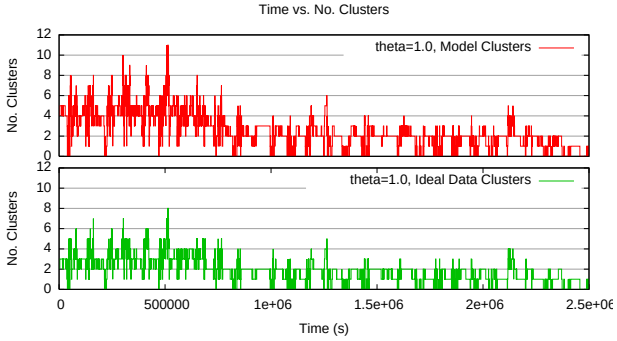

Figure 7: Ideal vs. model clusters, for $\theta = 1.0$.

Clustering data similarity. To assess the performance of our clustering algorithm, we compare the number of θ-clusters computed using our model-based approach to the *ideal* number of clusters based on the true data values of the sensors sent to the sink every minute. We ran our optimal clustering algorithm over raw sensor data and over model bounds from all 50 sensors over the entire set of readings (about 27 days). Figure 7 shows that our clustering approach based on data models performs well. In fact, over 26 days the number of cluster models is slightly above the

ideal number of clusters that could be obtained by directly querying each node (i.e., most of the time the number of model clusters exceeds the ideal clusters by $1-2$). Note that during moments of instability or rapid variation in the data distribution, the number of model clusters increases noticeably as the error of the model increases. However, this does not affect the average the performance of our approach. For example, over the first week our model-based approach clusters 35 sensors into 4.2 clusters on average, whereas the average of the ideal clusters is equal to 2.4 (note that the maximum divergence of sensor readings varies from 2 to 8 degrees). These experimental results show the efficiency of our clustering approach since it allows us to detect data similarity at *no additional* communication cost, both providing excellent energy conservation and good accuracy.

7. RELATED WORK

We group related work into approximate query approaches for sensor networks, work on time series forecasting in sensor networks and work on data stream mining.

Sensor Network Querying. There has been some work on the use of probabilistic and time series models in sensor networks. As in SAF, [17, 8, 18, 25] rely on a keeping local and global probabilistic in sync, to reduce the amount of communications between sensors and sink. We briefly compare our framework with some previous approximate approaches and show that our approach provides a good trade-off between energy consumption and data accuracy, as well as several other attractive properties. As shown in Figure 8, SAF inherits the desirable features provided by several different approaches. In fact, statistical models such as those employed in Ken [8] have the ability to detect outlier values, time intervals during which the data is inconsistent, and correlations among different measurements. However, these models (used in Ken [8] and BBQ [11]) require a large and expensive training set of days or weeks at a high communication cost during which each sensor node has to transmit periodic readings to the sink (e.g., every 30–60 seconds). This training phase consumes valuable energy source and limits the benefits of the probabilistic approach. This cost must be paid each time the model is computed or adapted.

	SAF	BBQ	Ken	Kalman	PAQ
Adaptability	yes	no	yes	yes	Yes
Duration of Learning	< 1 hour	days to weeks	days to weeks	-	< 1 hour
Outlier Detection	yes	no	yes	no	yes
Avg. transmissions per week per sensor (@ 1 min/sample)	150-200	<= 10,000 if learning else < 100.	<= 10,000 if learning/ adapting else < 100.	150-200	~10,000 per cluster head
Cost of similarity	No additional cost	No additional cost	Costly clustering protocol	-	Medium cost clustering protocol
Detection of inconsistent periods	yes	yes	yes	-	-
Robustness to comm. failure	high	low	low	high	low/medium

Figure 8: Comparing features.

In contrast, SAF has a short learning phase (i.e., $<$ 1 hour) during which each node does not have to communicate with the sink. In addition, BBQ and Ken are vulnerable to communication failures during training since the model at the sink is built based on the data sent periodically by each node. That may cause inconsistencies between the copies maintained at nodes and at the sink. This problem is present to a small extent also in PAQ [25] since each cluster head transmits periodic readings to the sink, but is not evident in SAF.

Kalman-based approaches (e.g., the method proposed by Jain et al. [17]) are highly adaptable and require low communication between the sink and the sensor node, like SAF. In our experiments, we compared our local model with Kalman filters [17] over our lab sensor data and found that the total number of transmissions performed by the sensor when using our model is comparable to the method of Jain et al. [17]. However, Kalman filters are unable to provide strong properties regarding the data distribution, nor can they detect anomalies. In addition, their weak properties do not allow nodes to turn off their radio during periods of data stability in contrast with SAF. SAF also improves on PAQ [25], in that it uses a different model (including a trend component), and a greatly improved clustering and data similarity detection algorithm.

The snapshot queries approach proposed by Kotidis [18] is also similar to ours in that it exploits local models and correlations, but it provides weaker guarantees. Other work, such as the work by Olston *et al.* [6] shows how to approximate answers to queries in distributed environments with a fixed bound on the error; these approaches, though simple, have the potential to offer far less reduction in communication than model-based approaches such as ours and those discussed above. Han *et al.* [14] show how similar (non-probabilistic) techniques can be adapted to the sensor network domain in an energy efficient way. Cormode and Garofalakis [9] present a method for estimating the answers to queries inside a sensor network using sketch-based methods to focus on computing complex aggregate queries inside of a sensor network.

Time series forecasting in sensor networks. Our approach is similar to the one proposed in [23] for using autoregressive models built at each sensor node to reduce communications in the context of time synchronization. That work did not focus on querying or clustering issues, however. Lazaridis and Mehrotra [19] use a different time-series method to create a piecewise linear approximation of signals generated by sensors, and send those approximations out of the network. Their approach differs from ours in that they capture a large time series and approximate it, rather than building a model that can be used for prediction outside of the network. Other time series approximation methods, based, for instance, on wavelets [7, 12] have been proposed; these too strive to reduce communication but do not offer the same predictive power as our methods. AR models have been widely used outside the wireless sensor network domain as a way to approximate and summarize time series with applications in finance, communication, weather prediction, and a variety of other domains. Brockwell and Davis [4] provide an excellent introduction to time series.

Data stream processing. One challenge associated with analyzing data streams is dealing with the large quantity of high-rate data. This has led to a number of summarization

techniques (e.g., sampling, synopsis data structures, and aggregation [15, 2]). However, these techniques are not well-suited for sensor networks because of their computational complexity and memory requirements. The AR model provides a small and cheap fingerprint of the data stream produced at the sensor, and it performs well in presence of temporal fluctuating data (as we showed in [25].) There has also been a much work on data stream clustering [15, 3, 13]. However, these algorithms are not well–suited to sensor networks due to their memory requirements and running time which depends on the number of data stream points. Our clustering algorithm (tailor–fit for our problem) is more efficient because it groups subsequences of data streams according to their AR models and has runtime that scales with the number of nodes rather than the number of readings.

8. CONCLUSIONS

We have proposed an energy–efficient approximate query framework which dramatically reduces the amount of communication in sensor networks. Our scheme works by detecting data similarities among sensor nodes by comparing their local models rather than their raw data. This definition of similarity, coupled with the linearity of our models, allowed us to derive an efficient clustering algorithm that is provably *optimal* in the number of clusters formed by the network. Our clusters have several interesting features: they can change regularly with little overhead as nodes do not keep track of their cluster membership, they can capture similarity between far away sensors, and are well suited to *mobile networks*. We have validated our approach both analytically and through simulation-based experimental results on a trace of real data. Our simulations confirm the analytical results and offer encouraging preliminary proof of the benefits of our approach.

Acknowledgments

This work was supported by the National Science Foundation under NSF Grants 0448124 and IIS-0325525.

9. REFERENCES

[1] R. Adler, P. Buonadonna, J. Chabra, M. Flanigan, L. Krishnamurthy, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: Experiences from the north sea and a semiconductor plant. In *SenSys*, 2005.

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

[3] B. Babcock, M. Datar, R. Motwani, and L. O'Callangham. Maintaining variance and k-medians over data stream windows. In *PODS*, 2003.

[4] P. Brockwell and R. Davis. *Introduction to Time Series and Forecasting*. Springer, 1994.

[5] T. Brooke and J. Burrell. From ethnography to design in a vineyard. In *Proceeedings of the Design User Experiences (DUX) Conference*, June 2003.

[6] J. C. Olston. Best effort cache sychronization with source cooperation. In *SIGMOD*, 2002.

[7] H. Chen, J. Li, and P. Mohapatra. RACE: Time series compression with rate adaptivity and error bound for sensor networks. In *Proceedings of MASS*, 2004.

[8] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *ICDE*, April 2006. To Appear.

[9] G. Cormode and M. Garofalakis. Sketching streams through the net: distributed approximate query tracking. In *VLDB*, pages 13–24, 2005.

[10] I. Crossbow. Wireless sensor networks. `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`.

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

[12] D. Ganesan, D. Estrin, and J. Heidemann. Dimensions: Why do we need a new data handling architecture for sensor networks? In *HotNets*, 2002.

[13] S. Guha, N. Mishra, R. Motwani, and L. O. Callangham. Clustering data streams. In *FOCS*, 2000.

[14] Q. Han, S. Mehrotra, and N. Venkatasubramanian. Energy efficient data collection in distributed sensor environments. In *Proceedings of ICDCS*, 2004.

[15] M. L. Hetland. *Data Mining in Time Series Databases*. World Scientific, 2005.

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

[17] A. Jain, E. Y. Chang, and Y. Wanf. Adaptive stream management using kalman filters. In *SIGMOD*, 2004.

[18] Y. Kotidis. Snapshot queries: towards data-centric sensor networks. In *ICDE*, 2005.

[19] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *Proceedings of ICDE*, 2003.

[20] S. Madden, W. Hong, J. M. Hellerstein, and M. Franklin. TinyDB web page. `http://telegraph.cs.berkeley.edu/tinydb`.

[21] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. In *WSNA*, 2002.

[22] P.Bonnet, J.Gehrke, and P.Seshadri. Towards sensor database systems. In *MDM*, Jan 2001.

[23] D. Tulone. Resource–efficient time estimation for wireless sensor networks. In *DIALM-POMC*, 2004.

[24] D. Tulone. *Mechanisms for energy conservation in wireless sensor networks.* PhD thesis, Dept of Computer Science, Univ. of Pisa, December 2005.

[25] D. Tulone and S. Madden. PAQ: Time series forecasting for approximate query answering in sensor networks. In *EWSN*, 2006.