PAQ: Time series forecasting for approximate query answering in sensor networks

Daniela Tulone[1,2] and Samuel Madden[1]

[1] MIT Computer Science and Artificial Intelligence Laboratory
{tulone,madden}@csail.mit.edu
[2] Computer Science Department, University of Pisa

Abstract. In this paper, we present a method for approximating the values of sensors in a wireless sensor network based on *time series forecasting*. More specifically, our approach relies on *autoregressive* models built at each sensor to predict local readings. Nodes transmit these local models to a sink node, which uses them to predict sensor values without directly communicating with sensors. When needed, nodes send information about outlier readings and model updates to the sink. We show that this approach can dramatically reduce the amount of communication required to monitor the readings of all sensors in a network, and demonstrate that our approach provides provably-correct, user-controllable error bounds on the predicted values of each sensor.

1 Introduction

Wireless sensor networks offer the potential to collect large amounts of high-fidelity information about a variety of remote locations. Recent deployments have demonstrated their utility in environmental monitoring [18], agriculture [3], and industrial monitoring and process control [1]. Most of these deployments have a similar character – data is collected at a regular rate to some centralized basestation (or *sink*), where it is stored on disk and analyzed using conventional data processing tools (e.g., databases, mathematical analysis packages, and GIS software.) One major focus of sensor network research has been on building tools to facilitate this collection of data. Researchers have proposed a variety of abstractions to enable such applications to be rapidly built, ranging from database query languages (as in TinyDB [17]) to parallel programming systems (e.g., Regions [24]), to power conserving and failure-masking network layers (e.g., Directed Diffusion [13]).

In this paper we focus on improving the performance of these data collection applications using a probabilistic approach. More precisely, we employ a class of statistical techniques broadly known as *time series forecasting*. These techniques apply to phenomena evolving over time, and use the recent history of readings to predict the most likely future values. In this paper, we propose a general framework to efficiently answer queries at the sink based on a particular type of time-series model called an *autoregressive models* (AR), the simplest time series model. We chose this model because it is computationally tractable on modern-generation sensor networks (unlike the fully general ARMA models, for example [2]) and because, as we show, it can offer a substantial reduction in communication and improvement in loss rates over existing data collection approaches.

We evaluate our AR model both analytically and through simulation results that show it can properly model physical phenomena and accurately predict future values.

We also show that it has low computational cost and memory usage, suggesting its suitability for a wide range of hardware. Our system, called PAQ (for *Probabilistic Adaptable Query system*) uses a combination of AR models to probabilistically answer queries. The model is used both *globally*, at the sink, to predict the readings of individual sensors, and *locally*, at each sensor, to detect when the sensor produces outlier readings or when the model ceases to properly fit the data (allowing the sensor to re-learn the model and notify the sink of the new model parameters.) Our approach has the following advantages over previous deterministic query systems:

- It significantly reduces the amount of communication required to report the value of every sensor at the sink.
- It allows the detection sensor readings that are "outliers", in the sense that they are not consistent with recent history or have malfunctioned.
- It is adaptive to dynamic changes in the distribution of data produced by sensors, and tolerant of missing sensor data.
- It does not require a large amount of training data or a priori knowledge of the probability distribution of sensor values and can work with any of the previously mentioned abstractions (e.g., TinyDB, Directed Diffusion) for data collion.

There has been some recent interest in applying statistical modeling techniques to sensor network query systems [9, 5]. Our approach is similar to [9] in its general probabilistic approach: the sink answers queries within a user–specified error bound by computing a prediction for those values without communicating with the sensors, thus avoiding message transmissions and sensing operations. However, there are substantial differences between our approach and previous probabilistic approaches to query answering. First, these existing approaches typically build a model *centrally* at the sink, using an expensive learning phase where each sensor transmits many readings to the sink. One reason for this is that these previous approaches have used relatively complex probabilistic models (e.g., multi-variate Gaussians [9] or generalized graphical models) which are too complex to build or maintain on many current classes of sensor network hardware (e.g., Berkeley/Crossbow Motes [8]). These approaches cannot adapt to changes in the underlying distribution of sensor data without re–running this expensive learning phase. In contrast, our framework relies mostly on *local* probabilistic models computed and maintained at each sensor. In order to adapt the local model to variations in the data distribution, each sensor continuously maintains its local model, and notifies the sink only of significant changes. This allows both the sensors and sink to adapt to changes in the underlying distribution without the addition of a complex decision process that tries to decide when to invoke an expensive re-training phase.

We further limit communication from the sink to each of the nodes by exploiting *data similarities* between sensors that are geographically nearby. Our PAQ system relies on *geographic clusters* of sensors that are similar at a given point in time and that are computed by sensors that are near each other. Therefore, the sink maintains only the models (coefficients) of a few designated sensors, called *cluster leaders*, and uses them for prediction. Before discussing the details of our algorithm, we briefly review related work.

2 Related Work

There has been some work on the use of probabilistic and time series models in sensor networks. As in PAQ, [14, 6, 15] rely on a combination of local and global probabilistic

models which are kept in synch to reduce communication between sensor nodes and the network sink. For instance, Jain et al. [14] propose a query framework based on Kalman filters: both the sink and sensors activate a Kalman filter with user-specified accuracy when a new query is received. However, this strategy does not support multiple queries with variable precision or clustering, and the local models do not adjust to non–linear phenomena. Recent work by Chu et al. [6] is similar to ours in that it also exploits temporal/spatial correlation. However, it has a heavyweight learning phase that does not work well for non-stationary data. Neither work [14, 6] provides a provable bound on the maximum error or on the error probability of answers provided at the sink. The snapshot queries approach proposed by Kotidis [15] is also similar to ours in that it exploits local models and correlations, but it provides weaker guarantees. Cheng *et al.* [5] and Deshpande *et al.* [9] have shown that generative-model based approaches can significantly reduce the communication burden in sensor networks. However, these approaches require a relatively sophisticated user who can describe the appropriate model for his or her domain and usually involve a complex centralized learning phase that must be re-run if the data distribution changes. In contrast, our approach is predicated on lightweight models that can be learned by the individual nodes in the network and rapidly retrained when confronted with non-stationary distributions.

Other work, such as the work by Olston *et al.* [20] 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.* [12] show how similar (non-probabilistic) techniques can be adapted to the sensor network domain in an energy efficient way.

Our approach is similar to the one proposed by Tulone [23] for using autoregressive models built at each sensor node to reduce communications in the context of time syncrhonization. That work did not focus on querying or clustering issues, however. Lazaridis and Mehrotra [16] 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 [4, 10] have been proposed; these too strive to reduce communication but do not offer the same predictive power as our methods. Autoregressive time series 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 [2] provide an excellent introduction to time series and their applications.

3 Preliminaries

In this section, we introduce the notation and terminology we will use throughout the paper, and provide an overview of time series techniques.

3.1 System model

Our network consists of a dynamic set $\mathcal{S}$ of sensor nodes and one or more sink nodes. Each node is equipped with some sensing capability, performing readings on m physical phenomena (metrics) $F_1, F_2, \ldots, F_m$ each of which evolves over time. For

example, we might say that F_1 =temperature, and F_2 =light. We assume that each sensor performs a reading on each F_i every Γ time units. We have designed PAQ to work with the lowest-end of today's sensor nodes, including Berkeley Motes [8], with just a few kilobytes of memory and slow, 8-bit processors without floating point or dedicated signal processing hardware.

3.2 Queries

Queries are submitted at the sink. In this paper, we focus on queries of the form:

SELECT sensorlist WHERE P($F_1, \ldots, F_m$) ERROR x CONFIDENCE k%.

where P($F_1, \ldots, F_m$) is a predicate over $F_1, \ldots, F_m$ consisting of atoms $F_i \in [a, b]$, $F_i > a$, and $F_i < b$ where a, b are user-specified. For instance, a valid predicate could be $[(F_2 \in [a, b] \vee F_3 \in [c, d]) \wedge F_m > g]$. Here ERROR x indicates that the user is tolerant to a maximum absolute error in the query result, and the CONFIDENCE clause indicates that at least k% of the readings should be within x of their true value. For example, the user might issue the query "SELECT nodeid, temp WHERE temp $>$ 25°C ERROR .1°C CONFIDENCE 95%", which would report the temperature at each node to within .1°C, a property which would be satisfied by 95% of the readings.

3.3 Time series forecasting

In general, a *time series* is a set of observations x_t, each of which is recorded at time t. An important part of the analysis of time series is the description of a suitable uncertainty model for the data. To allow for the possibly unpredictable nature of future observations, it is natural to suppose that each observation x_t is a sample of a random variable X_t (often denoted as $X(t)$).

Definition 1. *A time series model for the observed data* $\{x_t\}$ *is a specification of the joint distributions of the random variables* $\{X_t\}$ *of which* $\{x_t\}$ *is a sample.*

Clearly, if we wish to make predictions, then we must assume that some part of this model does not vary with time. Therefore, an important component of time series modeling is to remove *trend* and *seasonal* components to get a (weakly) *stationary* time series. Informally, a time series $\{X_t\}$ is stationary if it has statistical properties similar to those of the time–shifted series $\{X_{t+h}\}$ for each integer h. More precisely, $\{X_t\}$ is *weakly stationary* if its mean function $\mu_X(t)$ and its covariance function $\gamma_X(t + h, t)$ are independent of t for each h.

Linear time series models, which includes the class of *autoregressive moving–average* (ARMA) models, provide a general framework for studying stationary processes. The ARMA processes are defined by linear difference equations with constant coefficients. One of the key properties is the existence and uniqueness of stationary solutions of the defining equations [2].

Definition 2. $\{X_t\}$ *is an ARMA(p, q) process if* $\{X_t\}$ *is stationary and for all* t*,*

$$X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}$$

where $\{Z_t\} \sim WN(0, \sigma^2)$ *and the polynomials* $(1 - \phi_1 z - \ldots - \phi_p z^p)$ *and* $(1 - \theta_1 z - \ldots - \theta_q z^q)$ *have no common factors.*

Here, $\{Z_t\}$ is a series of *uncorrelated* random variables, each with zero mean and σ^2 variance. Such a sequence is referred as *white noise* and denoted by $WN(0, \sigma^2)$. An autoregressive model of degree p, denoted by $AR(p)$, is a particular type of ARMA model with $q = 0$ (i.e., the right hand side of Definition 2 contains just a single term.) Such models are simply referred to as *autoregressive* or AR models. We will adopt this model to predict the value of F_i read at time t by a sensor. This choice is motivated by its simplicity, which leads to lower computational cost and memory requirements, making such models practical on many current-generation sensor networks. Clearly, there is an inherent trade–off between efficiency and precision since there exist more sophisticated models (e.g., Hidden Markov chains [7], or recursive neural networks [19]) that are able to capture non-linear data distributions but are much more computationally expensive. We will show that AR models offer a good balance between simplicity and accuracy in sensor networks.

4 PAQ system overview

As mentioned above, we employ a combination of statistical models with live data acquisition. Each sensor in PAQ maintains a local AR model, and samples its values once every Γ seconds. It uses recent readings to predict future local readings. When a reading is not properly predicted by the model, the node may mark the reading as an outlier or choose to re-learn the model, depending on the extent to which the reading disagrees with the model and the number of recent outliers that have been detected. This design is motivated by the need to (1) *monitor* changes in the physical phenomena and detect outliers, and (2) reduce communication between the local sensors and the sink. Except when a node is a *cluster leader* (see below), it does not need to communicate while monitoring the model or during its learning phase, but only when computing and adjusting its cluster, as described in Section 7.2. We discuss local modeling more in detail in Section 4.1 and the process of marking readings as outliers and deciding when to re-learn in Section 5.

Parameter	Description
	Parameters Used in Basic Model
α, β, γ	Coefficients of AR(3) model
Γ	Time elapsed between two consecutive readings
ν	Confidence parameter on predicted sensor readings
ε	Error bound on predicted sensor readings; will have error $> \varepsilon$ with probability $1/\nu^2$
	Parameters Used in Dynamic Model
δ	Error threshold above which model re-computation is triggered
a	Fraction of readings that must be wrong before model re-computation is triggered
Λ	Number of readings during the *monitor window* in monitor() algorithm
Δ	Time that cluster leader waits in cluster formation algorithm
θ	Data similarity parameter

Fig. 1. Notation used in this paper.

The sink maintains one AR model per *geographic cluster*. A cluster is a subset of sensors within communication range of each other whose values differ from each other by at most by a constant value θ. Intuitively, it should be possible to cluster sensors in this way, since we expect that nearby sensors will often produce similar readings. Clearly, clusters are dynamic sets that can vary in number since the local models are dynamic.

One sensor in each cluster is designated the *leader*. It is responsible for communicating with the sink on behalf of its cluster. The leader's AR model, called the *cluster model*, is used to predict the values of all sensors in the cluster with an error of at most θ over the member sensor's local models, and with the same confidence. The sink maintains the coefficients associated with each of the leaders' models and receives periodic readings from them. It also maintains a list of the current clusters. The leaders' models and the cluster sets stored at the sink allow the sink to answer queries over all sensors using just the cluster models, avoiding a large number of message transmissions. To reduce communications, clusters are computed locally by the cluster leaders and members of each cluster. Each leader is responsible for notifying the sink for changes in its model (in the coefficients) or in its cluster members, and for transmitting periodic readings to the sink. Details of the cluster formation and query algorithms are given in Section 7.

4.1 Local AR model

The probabilistic model maintained by the sensor must be light-weight, in terms of both the computational and storage requirements because of limitations of the sensor. Our local models are designed with these energy restrictions in mind.

Trends and seasonal components. Since physical phenomena are typically not stationary, a time series is usually decomposed into a *trend component* that grows very slowly over time, a *seasonal component* with some periodicity, and a *stationary component*. However, maintaining the trend and seasonal components substantially increases the complexity of both learning and adapting the model. In order to simplify our model and ignore trend and seasonal components, we consider an autoregressive model AR(p) with a narrow *prediction window*, such as $p = 3$, similarly to [23]. As discussed in [2], if the time elapsed since the last reading is relatively short, it is reasonable to neglect those components.

However, an AR(p) model is unlikely to be a good fit for non–linear physical phenomena. In particular, we have observed that sensor network data is typically locally linear, but there are periodic non-linearities that are not well-predicted by AR(p) models. To solve this problem, we enhance our linear models with *dynamic updates* that are detected and performed locally – the idea is to detect when the model is no longer a good fit for the data being sensed, and dynamically re-learn the model coefficients when this happens. The efficiency of this approach comes from the fact that learning and updating the AR model is cheap compared to the costs of learning and maintaining a non–linear model.

Multivariate vs. univariate models. In sensor networks, each sensor device typically has multiple sensors. To handle multiple sensors, we can compute a multivariate AR model with m components, one for each physical measurement or we can create m univariate models. Clearly, for $m > 2$ the computational cost of learning the multivariate model is higher than the cost associated with m univariate models. To show this, we compare the number of unknowns that have to be computed in both cases during the learning phase: for a multivariate model, the sensor has to compute qm^2 coefficients, while in case of m univariate models it computes mq coefficients. In terms of storage space the multivariate model requires $qm^2 + (q + 2)m$ memory compared to $2m(q + 1)$ required by m univariate AR models, which is a savings of $qm(m - 2)$. For these reasons, it is more efficient for the sensor to compute m *univariate* AR models, although the multivariate model provides additional predictive power as it is

able to capture correlations between measurements. For simplicity of presentation we consider only one measurement, F, in the rest of this paper. However, in general, our techniques apply to either multiple univariate models or a single multivariate model.

Sensing model. We assume that each measurement reads F every Γ time units, and denote the history of these values up to time t as $v_1, \ldots, v_i, \ldots, v_t$. Although our proposal is independent of the size of the parameter vector q, we consider $q = 3$ because it allows us simplify the model and ignore seasonal and trend components, and has low computational and storage costs. Therefore, each sensor S_j models F as a *dynamic* AR(3) time series with Gaussian white noise of zero mean and standard deviation $b(\omega)$. In case of a time series $F(t)$ with non–zero mean η, we study the time series $X(t) = F(t) - \eta$, as in [2]

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

with $\alpha, \beta, \gamma \in \mathcal{R}$. Therefore, the predictor $P(t)$ of F at time t is given by its mean η plus a linear combination of the increments or decrements of the last three readings with respect to η. More precisely, the prediction at time $t > t_{i-1}$ is given as

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

Function $b(\omega)$ represents the standard deviation of the white noise. Here, we assume that the distribution of the noise does not vary over time. The following lemma computes the error bound associated with the prediction $P(t)$ of $F(t)$ at time t.

Lemma 1. *Let $P(t)$ be the prediction of F at time t associated with model (1), and let $\varepsilon = \nu\ b(\omega)$, 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 $\frac{1}{\nu^2}$.*

Proof. Since $X(t)$ is driven by a Gaussian white noise with zero mean and standard deviation $b(\omega)$, at any time t the prediction error $v_t - P(t)$ is normally distributed with zero mean and standard deviation $b(\omega)$. The proof follows by applying the Chebychev inequality to obtain the prediction error:

$$P(|F(t) - P(t)| \geq \varepsilon) \leq \frac{b(\omega)^2}{\varepsilon^2} = \frac{b(\omega)^2}{\nu^2 b(\omega)^2} = \frac{1}{\nu^2}$$

We choose ν such that the error probability ν^{-2} can be considered negligible; e.g., for a typical value of ν we use 6 or 7. As a result, readings with an error larger than ε are classified as anomalies that are handled specially as described in Section 5.2.

4.2 Learning phase

In this section we illustrate the steps taken by each sensor to learn its local model as defined by (1).

Data structures. There are two parameters that principally affect the efficiency and accuracy of the learning phase: the number of readings, N, collected during the learning phase, and Γ, the time interval between two consecutive readings. During our experiments, we study different values of these parameters. Given these parameters, each sensor builds the following data structures during the learning phase:

– a (initially empty) queue V, containing the most recent N readings.
– the coefficients α, β, γ, and the mean η;
– the standard deviation $b(\Gamma)$ of the white noise during Γ time units.

learn():

1) $\nu \leftarrow 0$
2) for $k = 1$ to N
3) read value v_k
4) enqueue v_k into V
5) wait for Γ time units
6) $\langle \alpha, \beta, \gamma \rangle \leftarrow$ **solveSys**(V)
7) $b(\Gamma) \leftarrow$**compVar**(V)

Fig. 2. Learning phase.

The learning algorithm is illustrated in Figure 2. During the learning phase the sensor performs a reading every Γ time units, and inserts it into V (Figure 2 lines 3–5.) After performing N readings, it invokes the function **solveSys** which computes the mean η from the N readings, and the coefficients α, β and γ. Notice that we do not recursively compute α, β, γ as in the Durban-Levine algorithm [2]), but only at the end for efficiency reasons, as in [23]. The coefficients are computed by calculating the *minimum squared error* between the readings contained in V and the predicted values via least-squares regression.

Least-squares regression works as follows: suppose that $v_1, \ldots, v_N$ are the values read during the learning phase, and that $\bar{v}_1, \ldots, \bar{v}_N$ are such that $\bar{v}_i = v_i - \eta$ for $i = 1, \ldots, N$. Then, α, β, γ correspond to the coefficients of the *best linear predictor* and are obtained by minimizing the function $Q(\alpha, \beta, \gamma)$

$$Q(\alpha, \beta, \gamma) = \sum_{i=4}^{N} (\bar{v}_i - (\alpha\ \bar{v}_{i-1} + \beta\ \bar{v}_{i-2} + \gamma\ \bar{v}_{i-3}))^2$$

The coefficients α, β, γ can be computed by setting the partial derivatives of the minimum squared error to zero and solving a linear system of three equations (we omit the details of this computation and instead direct the reader towards a standard linear algebra text, such as [11]).

solveSys computes the solution of these equations. After computing the mean η and coefficients α, β, γ, the sensor computes the variance of the white noise during Γ time units by invoking **compVar**, Fig 2 line 7. This is done by computing the prediction error $e_i = P_i - \bar{v}_i$ for $i = 1, \ldots, N$, and the mean $\bar{e}$ of $e_1, \ldots, e_N$. Hence, the variance of the white noise during Γ time units is:

$$b(\Gamma) = \sqrt{\frac{\sum_{i=1}^{N} (e_i - \bar{e})^2}{N - 1}}$$

Thus, the parameters $\eta, \alpha, \beta, \gamma$ and $b(\Gamma)$ uniquely describe the AR model for a given set of learning data $\{v_1, \ldots, v_N\}$.

Costs of the learning phase. The computational cost of the learning phase is the cost of reading N values, plus the cost of computing matrices A and B of the linear system $A\ X = B$ in 3 unknowns, which involves $12(N - 3)$ sum and product operations, plus the cost of solving the linear system. Therefore, the total cost is equal to $l + (s + 12)N - 36$, with s cost per reading and l cost for solving a linear system with 3 unknowns. In terms of memory, it requires a $(N + 5)$–vector plus a 3×4 real–valued matrix.

5 A dynamic local AR model

As discussed in Section 4.1 the local AR model must be dynamic. Hence, the sensor periodically monitors its local model and updates it as needed. This design offers several benefits with respect to having the sink or another specialized node monitor the validity of the sensor's model:

1. It allows the sensor to detect *data anomalies* with respect to previous history, where a data anomaly is a sensor value that the model does not predict to within the user-specified error bound. These anomalies can be classified into *outlier values*, which are transient mispredictions that the model simply does not account for, or *distribution changes*, which are persistent mispredictions that suggest the model needs to be re-learned, either because of a faulty sensor or a fundamental shift in the data being sensed.
2. It requires no communication during learning and updating (in contrast with the approach taken in BBQ [9], for example). This is possible because of the simplicity of our local model, which requires relatively small learning history and low computational cost and memory storage.

However, the efficiency of this approach is related to the efficiency of monitoring and updating the AR model, which we discuss in the remainder of this Section. For the purpose of this discussion, we assume that each cluster consists of just one sensor that is responsible for transmitting changes to its model and/or outlier values to the sink. We discuss forming geographic clusters in Section 7.1; once clusters have been formed, only cluster leaders transmit their values to the sink.

5.1 Choosing to Re-learn the Model

In order to save energy, we believe the model should be updated only when its readings diverge *consistently* from its model. We classify data anomalies based on the model's *prediction error*. We consider two thresholds: δ and the maximum error $\varepsilon = \nu b(\Gamma)$ (as defined in Lemma 1.) These thresholds are chosen such that if the absolute value of the prediction error falls in $[0, \delta]$, then the model is a good predictor of the data. If it falls in $[\delta, \varepsilon]$ the data is still within the user specified error bound but the model might need to be updated. Finally, if the error prediction exceeds ε, then the data is an outlier because of Lemma 1. Since ν is chosen such that fewer than a fraction ν^{-2} of the values will be mispredicted (if the stationarity assumption holds) a single outlier value can be neglected while still satisfying the user specified confidence bound. However, though this might be an isolated anomaly that requires no action, it might correspond to an abrupt change in the data distribution (signifying that the data was not stationary), in which case the node should update the model and send the updated model parameters to the sink.

5.2 Monitoring algorithm

We can now describe the algorithm that is used to monitor the quality of the model, based on the considerations in the previous section. The monitoring algorithm is illustrated in Figure 3. Each sensor starts monitoring its model right after the learning phase. It takes a reading every Γ time units and updates its queue V, which contains the most recent N values (see Figure 3 lines 1-3.) If the prediction error exceeds δ it begins monitoring the next Λ readings, if it is not already doing so. While monitoring,

the sensor keeps track of the number of times the prediction error is in the range $[\delta \dots \varepsilon]$ (denoted in Figure 3 by the counter variable `upd`), as well as the number of times the error exceeds ε (denoted by the variable `out`).

This is shown in Figure 3 lines 6-9. The sensor sends a notification to the sink as soon as it detects a variation in the data distribution (Figure 3 lines 12-14). This occurs if `out` + `upd` exceeds a parameter a, expressed as a fraction of the number of readings during the monitor window, denoted by Λ, (e.g. 50% of the readings performed). Notice that the sensor node can employ different strategies in reporting outlier values to the sink, depending on application requirements. For instance, it could send only the first outlier value occurred during a monitoring window, or it could avoid transmitting outliers since it promptly reports variations in its data distribution to the sink, or it could simply report every outlier to the sink. The `notify()` operation implements the strategy used by the sensor, (Figure 3 line 10). After Λ readings, if the sensor has detected variations in the data distribution, it recomputes $\eta, \alpha, \beta, \gamma$ based on the values stored in V (Figure 3 lines 15-16.) Then, it sends the new coefficients to the sink and optionally a list of the outlier values, (Figure 3 lines 17.)

monitor():

```
1) every Γ time units do
2) read v
3) update V
4) if |v − P(t)| ≥ δ
5)     mark-if-not()
6)     if δ ≤ |v − P(t)| ≤ ν
7)         upd++
8)     else if |v − P(t)| ≥ ν
9)         out++
10)        if notify(out)
11)            send ⟨v⟩
12)    if (upd > a) || (upd + out ≥ a + 1)
13)        change ← true
14)        send ⟨change⟩
15)}if (end-of-monitor) ∧ (changes)
16)        solveSys()
17)        send ⟨η, α, β, γ,outlierList⟩
```

Fig. 3. Monitoring algorithm.

5.3 Discussion

Clearly, the accuracy and efficiency of the dynamic update depends on the parameters N, Λ, δ, ε, and a. N represents the length of the history that is used to compute the model. The computational cost of re–learning increases linearly with N. However, our experiments suggest that the accuracy does not necessarily improve as N grows. For instance, if the data distribution is irregular (e.g., not well fit by a linear model), then a larger value of N will not result in a better fit.

The choice of error bound ε presents a trade–off between accuracy and error probability (ability to meet a user specified confidence bound) which is inversely proportional to ν^2. Moreover, the choice of ν has an impact also on the number of readings marked as outliers, as shown in Section 6.

Another trade–off between accuracy and efficiency is presented by the threshold δ which defines, along with ε and a, the conditions under which the model should be updated. Clearly, it is desirable to keep the number of updates low, since updates incur additional learning and communication costs. However, making the interval $[\delta, \varepsilon]$ too small will not result in a energy improvement since the model will not properly fit the data and will thus flag more readings as outliers over time. Finally, the duration of the monitor window, Λ, presents another tradeoff: we want to keep the monitor window relatively short to update the model as fast as possible, but making it too short can result in excessively frequent updates.

6 Simulation results

To study the tradeoffs discussed in the previous section, we implemented the PAQ dynamic model and evaluated it on real data. We used a trace of sensor 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 approximately every thirty seconds from 54 sensors. In particular, we have focused on the temperature read by 5 sensors with different characteristics: some of them with many high spikes, others with irregularity, and some with a relatively smooth distribution (this corresponds to sensors 6, 7, 22, 32, and 45 in the aforementioned sensor data.) We selected these particular sensors to provide a sense of the performance of PAQ under different conditions. The traces we used from these sensors include a number of missing readings because the Berkeley Motes [8] used to capture the readings were communicating over a lossy radio channel. Overall, about 25% of the 30 second intervals do not have readings associated with them.

We ran experiments where each node used its model to predict its next value, and measured the error, the frequency with which the model needed to be re-learned, and the number of messages that a node would send to the sink during execution. In Section 7.1 we discuss how the sink uses these local models; in that Section we prove that the sink can maintain a centralized model that introduces a user-specifiable constant error over the error bounds shown here.

We varied several parameters throughout our experiments, though we only report on experiments with a small number of settings here due to space constraints. In general, we did not notice a large degree of sensitivity to the settings of various parameters, except in a few cases which we discuss more below.

Figure 4 shows the total number of messages transmitted by the nodes versus the user specified error threshold ν (here, we use communication overhead as a proxy for total energy consumption, as communication tends to be the most expensive aspect of operation in wireless sensor networks [22].) This includes transmissions as a result of learning new model parameters as well as transmission of outliers. In these experiments, approximately 25% of transmissions are a result of outliers being transmitted; transmitting the coefficients accounts for the remaining communication overhead. In our implementation, we transmit all outlier readings; if users are not concerned with receiving every outlier report, our approach could use somewhat fewer messages. The number of times the model is re-learned varies from point to point; at the lowest error rates (.1), the model is re-learned on about 20% of the monitoring windows. Notice that some sensors have a higher cost; these correspond to sensors that have "noisier" signals. Figure 5 compares raw data from sensor 6 (low cost) to sensor 45 (high cost); notice that sensor 45 has more noise than sensor 6.

We also compared our algorithm to an algorithm similar to that proposed in [20] which transmits a value to the sink whenever the current sensor value is more than the user-defined error threshold away from the value last transmitted to the sink. To make the comparison fair, we use a version of this algorithm that uses a monitoring window of the same size as in our algorithm, and that transmits new values at most once per monitoring period. We call this method "approximate caching" and show the number of transmissions it requires for two of the sensors in Figure 4. Notice that our approach substantially outperforms the approximate caching approach.

Figure 6 shows how the error varies with the user specified error threshold. Here, the X axis is the value of ε; the confidence is set to 95%. The Y axis shows the average

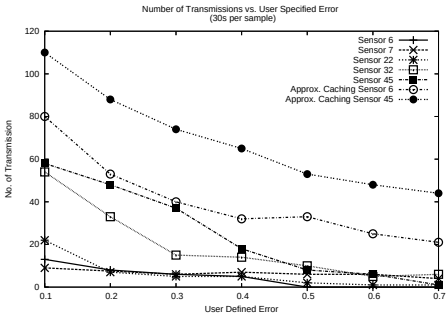

Fig. 4. Figure showing number of messages sent (either outlier or new model messages) for varying error thresholds. Parameters are as in Figure 6.

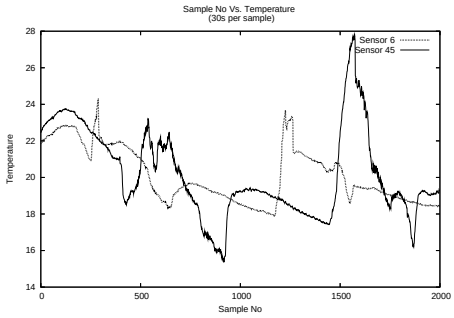

Fig. 5. Figure comparing the raw temperature readings from sensor 6 to sensor 45.

prediction error, which increases slightly with increasing ε but not dramatically since our error bounds are relatively conservative (notice that the actual error is well below the user-specified allowable error line defined by $y = x$). Because all errors are below the user specified threshold, it is not surprising that there is little variation between sensors; for example, even though sensor 45 requires substantially more re-learning phases and outlier transmissions, its average error is still low. We do not show errors for the "approximate caching" method, as it always meets the error bound.

The parameters that most affected our results were Λ, the size of learning window, and a and δ the thresholds that affect when readings are flagged as outliers and when the model is rebuilt. We noticed that the accuracy does not grow with the size of the learning sample N. In most cases it seems that the best prediction error occurs for $N = 60$, or, for sensor 45 (which tends to be "spikier" than the other data) with $N = 120$, while a learning phase consisting of $N = 20$ values implies a high number of model updates. For these reasons we chose to report results where $N = 60$.

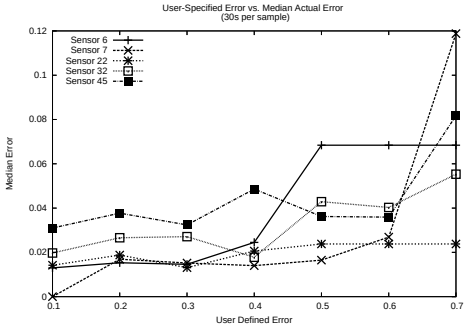

Fig. 6. Figure showing the actual error rate for different user-input error thresholds. Parameter settings are as follows: $N = 60$, $\Lambda = 15$, $\nu = 6$, $a = 8$, $\Gamma = 30$, $\delta = 1.8$

Clearly, irregular readings (e.g., sensor 45 in Figure 6) affect the accuracy of the prediction. However, the overall error is not affected noticeably by frequent missing data as shown by our results which are based on real data with a high number of irregular readings, though more irregular sensors do incur a higher overall cost as more outliers must be transmitted out of the network.

With respect to predicting values at the sink, our analysis suggests the maximum prediction error is equal to ν^{-2}; the graphs above suggest, at least for our data set, that this bound does in fact hold. We will see in the next section that this error grows by at most a user-defined constant θ when predicting the values of sensors that are a member of a cluster from that cluster's leader.

7 An Efficient Centralized Model

In the algorithms we have described thus far each sensor sends the sink the coefficients of its local model after the learning phase. However, to use the AR model, the sink requires periodic readings from the sensors, which would require energy-consuming communication on the part of each sensor at every Γ time units. As described above, we can reduce the extent of these communications by exploiting *geographic similarities* between sensors that are within direct radio communication with each other. This is based on the observation that sensors located near each other are likely to have similar readings. Once sensors are organized into clusters, only the leader sends its periodic readings to the sink. Other sensors continue to run the `monitor()` algorithm to update their local models, and detect group membership changes.

7.1 Sensor clusters

In this section we define data similarity among sensor readings, and show how we can group similar sensors into cluster sets. Let us suppose that θ, the *similarity constant* is a user-provided positive real value.

Definition 3. *Two sensors are similar at time t if their readings differ by at most the similarity constant, θ.*

Using Definition 3, we group sensors that are withing communication range of each other and are similar into clusters. Clusters will change over time since the AR model is dynamic. Hence, we define a set of clusters $\mathcal{C}(t)$ at time t as follows:

Definition 4. *A cluster set $\mathcal{C}(t)$ at time t is a set of subsets of $\mathcal{S}$ with the following properties:*

1. *any cluster $C \in \mathcal{C}(t)$ contains sensors within radio broadcast range of each other that are similar at time t;*
2. $\bigcup_{C \in \mathcal{C}(t)} C = \mathcal{S}$.

Each cluster C has a *leader*, a specified sensor in C which is elected locally by choosing, for example, the sensor with the lowest ID (or via any other function that can be locally evaluated at each node.) At any time the AR model associated with the cluster is the same as the local AR model of the current leader. The following Lemma computes the maximum error performed by predicting the value of a sensor in a cluster using the cluster model.

Lemma 2. *The maximum prediction error associated with the value of a sensor in a cluster is at most $\varepsilon + \theta$, with error probability at most ν^{-2}.*

Proof. Suppose we have a cluster C at time t such that $C = \{S_{i_1}, \ldots, S_{i_k}\}$, and node S_{i_1} is its leader. Therefore, the model of C at time t is represented by the model of S_{i_1}. For any sensor S_{i_j} in C, then $|v_t^j - P^{i_1}(t)| \leq |v_t^j - v^{i_1}(t)| + |v^{i_1}(t) - P^{i_1}(t)|$ and given Definitions 3 and 4, then $|v_t - P^{i_1}(t)| \leq \varepsilon + \theta$. The error probability is at most ν^{-2} because of Lemma 1.

In the next section, we sketch our approach for forming geographic clusters. This is a basic protocol whose correctness relies on reliable communication and symmetric radio links. We also assume that the round–trip communication time between the sink and any sensor is bounded. Though symmetry and reliability assumptions are not entirely realistic, recent publications suggest that careful neighborhood management and retransmissions can provide loss rates as low as 1-2 percent in static sensor networks [21], which should be sufficient for our purposes.

7.2 Building and maintaining clusters

The clustering protocol involves two major steps: first, at the end of the learning phase, sensors compute clusters; second, each sensor monitors and maintains its cluster-membership dynamically. Figures 7 and 8 present the high-level description of the clustering protocol. Due to space limitations we omit some details.

Building clusters. At the end of the learning phase each sensor runs a protocol to compute its cluster. The protocol is illustrated in Figure 7. Here, id denotes the identification number of the sensor running the protocol, and v is its current reading. The sensor node sets its cluster set CL to its identification number, and its leader L to zero since each id is a positive number (Figure 7 lines 1-2.) Each sensor node broadcasts its current value v and listens for Δ time units for other sensor broadcasts in order to detect similarities among its neighbors (Figure 7 lines 3-4.) During these Δ time units it listens for broadcasts, for each, and checks if the received value $\bar{v}$ diverges by θ or less from its value v. In this case it inserts the sensor identification into CL. After Δ time units it chooses the sensor with minimum identification number as the cluster leader of CL (Figure 7 line 5.) If the sensor has been chosen as the cluster leader, it sends a *leader notification message* that contains the cluster set CL and its model coefficients to its neighbors and to the sink (Figure 7 line 6-7.) Figure 8 illustrates the steps taken by a sensor when receiving a leader notification. If the sensor belongs to cluster cl and its cluster leader has not been set yet, it sets its leader to l (Figure 8 lines 1-2.) Otherwise, it keeps track of the other leaders within its radio broadcast by adding them into list OL (Figure 8 lines 3-4.) Notice that since CL is computed locally based on the messages received from its neighbors, a sensor might belong to two different clusters and might receive different leader notifications. The sensor follows the first leader notification. Since data changes over time, the leader periodically transmits its current value to its neighbors (Figure 7 lines 8-9.) The other sensors in the cluster verify that their value is θ similar to the leader's value.

```
1) L ← 0
2) CL ← {id}
3) broadcast(v)
4)    wait for Δ time units
         upon event receive(⟨v̄, i⟩) do
            if |v̄ − v| ≤ θ
               CL ← CL ∪ {i}

5) L ← min CL
6) if L = id
7)    send ⟨leader, L, CL, η, α, β, γ⟩

8)    every Γ time units do
9)       send ⟨v, L⟩
```

Fig. 7. Building clusters.

```
leader notification: receive(l, cl)
1) if (id ∈ cl) ∧ (L = 0)
2)    L ← l
3) else
4)    OL ← OL ∪ {l}

periodic validations: receive(⟨v̄, l⟩)
1) if (l = L) ∧ (|v̄ − v| > θ)
2)    if (s =nextLeader(OL)> 0)
3)       broadcast(⟨join, s⟩)
4)    else
5)       CL ← {id}
6)       L ← id
7)       send (⟨leader, L, CL, η, α, β, γ⟩)
```

Fig. 8. Notification protocols.

Maintaining clusters. Two factors can trigger changes in the cluster: (1) a sensor in the cluster can become dissimilar from the leadre, and (2) a leader can fail.

Let us consider the first case. Upon receiving the leader value, each sensor checks if its current value is within θ units from the leader value. If this condition does not hold, the sensor checks if its value is at most θ units away from the current values

of the other leaders within its radio broadcast (Figure 8 lines 1-2.) If it detects a leader whose value diverges by at most θ units from its current value, it broadcasts a `join` request, Figure 8 line 3. Otherwise, it creates a new cluster and notifies its neighbors and the sink of this change (Figure 8 lines 4-7). The leader that receives a `join` request updates its cluster set CL and notifies the sink, while the previous leader removes that sensor node from its cluster set.

Leader failures are detected by sensors listening for periodic sensor value messages. When several broadcasts are missed, a sensor infers that the leader may have failed and computes the new leader based on the remaining sensors in the group-member list. If a sensor detects that it should be the current leader (e.g., because it has the lowest id), it will broadcast a leader-change message as in the initial phase.

7.3 Answering queries

The sink locally stores the cluster models and the sensors belonging to each cluster. It uses this information to answer queries without additional communication. More precisely, it predicts the sensor values in a cluster using the model of the cluster leader, and verifies if this prediction satisfies the error-bounds associated with the query. Since the correctness of this query framework is strictly related to the validity of the cluster models at the sink, our query algorithm has to take into account the latency occurred in transmitting variations in the cluster to the sink. This is done by delaying the reporting of answers from the sink for a time equal to the maximum network latency. The system must also guarantee the error bound in the case of abrupt changes in the local data distributions at individual sensors. This is ensured by the outlier-transmission protocol described in the previous section. Such notifications are also delayed by at most the maximum latency of the network. Hence, the sink learns of new models and outliers within bounded time, and is able to answer queries with error at most $\varepsilon + \theta$.

Although this approach conserves energy by reducing the amount of communication between the sensors and sink it works well in relatively high density networks since it detects data similarity for sensors which are one–hop away from each other. Second, it can require a many transmissions when data distributions change or nodes fail frequently. We are currently exploring other clustering techniques that are able to overcome these limitations.

8 Conclusions

In this paper, we showed that AR models have the potential to dramatically reduce the amount of communication required to accurately monitor the values of sensors in a wireless sensor network. Compared to existing approaches based on centralized probabilistic models built at the sink, our approach is much lighter weight, allowing sensors to build models locally and monitor their values for significant changes or deviations from the model while still providing substantial communications reductions. This means that our approach is able to detect outliers or fundamental changes in model parameters. We also presented a simple clustering algorithm that allows us to further reduce communication from sensors to the sink while still providing a provable bound on the maximum predicted error at each sensor. Hence, we are optimistic that our approach will be important in future sensor network monitoring systems, and we look forward to extending our work with a complete implementation and evaluation.

Bibliography

[1] R. Adler, P. Buonadonna, J. Chhabra, 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] P. Brockwell and R. Davis. *Introduction to Time Series and Forecasting*. Springer, 1994.

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

[4] 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.

[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of SIGMOD*, 2003.

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

[7] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Spinger, New York, 1999.

[8] I. Crossbow. Wireless sensor networks (mica motes). `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`.

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

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

[11] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins, 1989.

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

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

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

[15] Y. Kotidis. Snapshot queries: towards data-centric sensor networks. In *Proc. of the 21th Intl. Conf. on Data Engineering*, April 2005.

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

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

[18] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. In *ACM Workshop on Sensor Networks and Applications*, 2002.

[19] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[20] C. Olston and J.Widom. Best effort cache sychronization with source cooperation. In *Proceedings of SIGMOD*, 2002.

[21] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of SenSys*, 2004.

[22] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51 – 58, May 2000.

[23] D. Tulone. A resource–efficient time estimation for wireless sensor networks. In *Proc. of the 4th Workshop of Principles of Mobile Computing, pp. 52–59*, 2004.

[24] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *NSDI*, March 2004.