

Model Based Error Correction for Wireless Sensor Networks

Shoubhik Mukhopadhyay, Debashis Panigrahi, Sujit Dey

Dept. of Electrical and Computer Engineering, University of California, San Diego

{shoubhik, dpani, dey}@ece.ucsd.edu

Abstract—One of the main challenges in wireless sensor networks is to provide low-cost, low-energy reliable data collection. Reliability against transient errors in sensor data can be provided using the model-based error correction described in [1], in which temporal correlation in the data is used to correct errors without any overheads at the sensor nodes. In the above work it is assumed that a perfect model of the data is available. However, as variations in the physical process are context-dependent and time-varying in a real sensor network, it is infeasible to have an accurate model of the data properties a priori, thus leading to reduced correction efficiency. In this paper, we address this issue by presenting a scalable methodology for improving the accuracy of data modeling through on-line estimation and model updates. Additionally, we propose enhancements to the data correction algorithm to incorporate robustness against dynamic model changes and potential modeling errors. We evaluate our system through simulations using real sensor data collected from different sources. Experimental results demonstrate that the proposed enhancements lead to an improvement of up to a factor of 10 over the earlier approach.

I. INTRODUCTION

The convergence of techniques for sensing, communication, and processing has led to the emergence of wireless sensor networks. One of the goals of the wireless sensor networks is to enable *reliable data collection* to meet the goals of the applications. Providing reliability is an important issue to address because majority of the sensor networks are remotely operated with very little human intervention once deployed and the maintenance/repair is also infeasible at times. Additionally, the sensor network is inherently exposed to several sources of unreliabilities such as errors from hardware noise, communication errors, errors in sensors, *etc.*, necessitating the need for reliability mechanisms.

One of the important factors to be considered in providing reliability in large sensor networked systems is the overall deployment cost. The deployment cost can be reduced by using low-cost sensor nodes, however that leads to constrained computational resources available on the sensor nodes. Another factor that affects the deployment of sensor networks is the lifetime of operation is primarily governed by the limited energy resources available. Hence, reliable sensor data collection should be provided using low-cost sensor nodes while consuming very low energy to enable proliferation of large-scale sensor networks.

It has been observed that low-cost reliable sensor data collection can be provided by exploiting the properties of the process being sensed. In [1], we had introduced a technique

which uses the temporal correlation of the data to correct transient errors in the received data using a data prediction model and a-posteriori information about future data. The suggested correction technique, however, assumes a perfect knowledge of the data properties and uses a pre-characterized data model to assist the correction process. However, building a perfect model of data offline is practically infeasible at times because of the following reasons. In a real sensor network, the data properties are context dependent. For example, temperature variation in a sensor being deployed outside in the field is different than one deployed inside an apartment. Moreover, the data properties vary over the lifetime of the sensor application necessitating run time changes in the data model.

In order to address the above issue, we enhance the framework presented earlier by introducing a scalable data modeling methodology. In the proposed methodology, the data modeling step is partitioned into two phases: (1) *Process Characterization*: identifies the basic model to be used for any given training data sets of the process being modeled, and (2) *Parameter Estimation*: computes parameters of the model on line using the data samples being collected. In addition, it is possible that the parameter estimation of the data model is not going to be accurate for the lifetime of the sensory application necessitating parameter changes as needed. Hence, we use a model tracking algorithm which monitors the accuracy of the current model and initiates the parameter estimation phase when needed to update the parameter values. Additionally, the correction algorithm presented in [1], assumes perfect knowledge of the data model. Hence, as shown in this paper, the correction efficiency of the algorithm reduces with on line update of the data model and inherent modeling errors. In this paper, we present two new correction algorithms to enhance the correction algorithm to be robust against model changes and inherent modeling errors.

We evaluated the presented framework in a C/Matlab simulation environment using data samples collected from an indoor environment, a commercial data center, and environmental sensor data available from CDEC [2]. Our experimental results show that the residual error level of the data can be significantly reduced by our framework without a large overhead on sensor nodes of the network.

The rest of the paper is organized as follows. In the next section, we present background material on sensor networks' unreliabilities and network architecture. We describe our overall approach in section III. The data modeling algorithm and the

correction algorithm are presented in section IV and section V respectively. Next, we present an experimental evaluation of the proposed technique followed by a comparison of our proposed technique with related work. Finally, we conclude with a summary of the proposed approach and scope for future extensions.

II. BACKGROUND

In this section, we first present a brief overview of the sources of unreliability in wireless sensor networks followed by the description of a network architecture assumed in this paper.

A. Unreliability in Sensor Networks

A wireless network of sensor nodes is inherently exposed to various sources of unreliability, such as unreliable communication channels, node failures, malicious tampering of nodes and eavesdropping. The sources of unreliability can be classified into three types based on the location of errors, *i.e.* sensor-level, node-level and network-level. Additionally, based on the properties of errors, the sources of unreliability can be divided into two categories: (1) faults that change behavior permanently, and (2) failures that lead to transient deviations from normal behavior, termed as soft failures in this paper.

The permanent faults include failures due to unavailability of energy resource, calibration errors after prolonged use, and loss of wireless coverage. Soft failures occur in wireless channels as transient errors, caused by noise from various sources, such as thermal noise at the receiver, channel interference and multi-path fading effects. Additionally, the use of aggressive design technologies such as deep-sub-micron (DSM) and ultra-deep-sub-micron (UDSM), to reduce the cost of each node further exposes the nodes to different types of transient errors in computations and sensing. The primary sources of such transient errors in nodes are ionizing particle strikes and electromagnetic noise from various sources such as crosstalk and transmission line effects. In this paper, we specifically target soft-failures that occur at the sensor and node levels. In our formulation and evaluation, we have modeled the effects of such failures on the sensor data as inversion of random bits. In our experiments we have assumed that the distribution of such errors can be described by a Bernoulli process.

B. Network Architecture

The network architecture which we have considered in this research consists of a hierarchical topology with two different types of nodes, as shown in Fig. 1. The first level contains sensor nodes connected in clusters, the cluster-heads being second-level nodes called micro-gateways (μ -gateway), which are connected amongst themselves as well as to external base stations. While the sensor nodes are very low cost devices possessing minimal computing capability, the μ -gateways have more resources available for computation, storage and communication. The μ -gateways are responsible for implementing

the proposed reliability mechanisms, as well as other functions like data collection and aggregation from sensors, distributed data processing and storage, and communication with base stations.

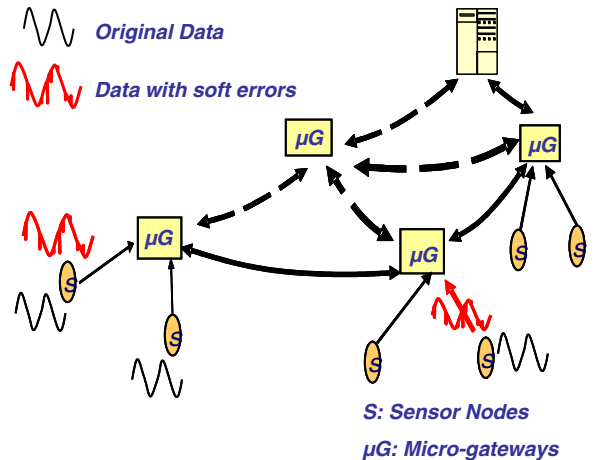


Fig. 1. Hierarchical sensor network showing sources of soft failures

We will use this architecture in this research and develop reliability techniques that can be deployed in the micro-gateways, imposing minimal resource or cost over-head on the sensor nodes. This will enable use of cheap, off-the-shelf sensor nodes, allowing rapid, cost-effective deployment, easier upgrades and better interoperability.

III. MODEL-BASED ERROR CORRECTION

In this section, we present an overview of the model-based error correction approach which was initially proposed in [1]. The main idea behind the approach is that a-priori knowledge of sensor data variation captured in a data prediction model can be used to detect and correct errors in the received data. The overall approach is illustrated in Fig. 2.

We use a model of the temporal variation in the sensor data (X_s) to predict the value of future samples given the past history of data. After a sample is corrected (X_c), the data model is used to predict the value of the next sample (X_p). Upon receiving the the next observed sample (X) and the predicted value by the model, the data correction block is used to decide the likelihood of the observed data to be erroneous. If the observed data is detected to be erroneous, we report the predicted value as the corrected output, if not, the observed value is used.

In our model-based error correction approach, we make a few assumptions about the properties of the sensing applications and the sensor data. First, it is assumed that representative sensor data of the process being monitored is available to be used for data model identification. Hence, the approach is most applicable for monitoring and tracking sensor applications, such as [3], where the sensed value is used to verify the expected behavior or departure from known

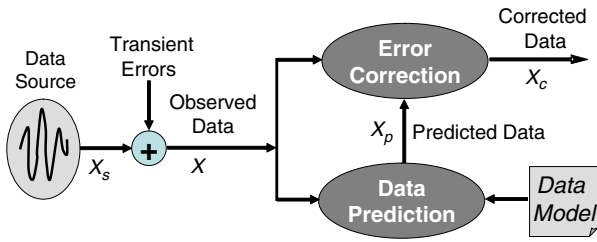


Fig. 2. Overall scheme for model-based error correction

pattern. Other applications, such as sensor applications where the goal is to learn about a physical process using the sampled information is out of the scope of this approach. Second, we assume that the application can tolerate a delay of the order of a few sample periods in receiving corrected sensor data. This is due to the inherent property of the correction algorithm to use a-posteriori information. However, the correction algorithm is configurable to meet any allowable delay requirements of the application. Finally, we assume that some primitive forms of protected data reporting mechanism is available at the sensor nodes, such as oversampling, duplicate sensor data. The protected data reporting is needed during the process of adjusting data model parameters on-line if needed.

It is important to observe that the presented approach does not require any additional computational resources or transmission overhead at the sensor nodes, and thus reduces the cost of sensor nodes and the energy consumption significantly. We implement all our data correction and modeling functions at the receiver of the data, *i.e.* the μ -gateways. At present, we only utilize the temporal correlation within the sensor data from individual sensor nodes, and hence can only correct errors that are not correlated temporally. In future, the framework can be extended to exploit spatial correlation to correct semi-transient errors as well. In the following sections, we first present the issues in the data modeling process, followed by the details of the correction process and the algorithms used.

IV. DATA MODELING

As mentioned in the previous section, in our approach of model-based correction we capture the temporal correlation of the sensor data in a model which is in turn used to predict future samples based on the past. The success of the correction thus depends largely on the accuracy of the modeling of the sensor data. The challenge mainly lies in estimating an accurate enough model by using the limited amount of data available at runtime. To address this problem, we assume that the basic nature of the model remains invariant over the application's lifetime, but some parameters of the model may show short term variations. On the basis of this assumption, we separate the data modeling process into two parts : (i) offline process characterization, including identification of the type and order of model used, and (ii) run-time estimation of model parameters. In the rest of this section, we describe each of the above steps in detail, identify the issues faced and

assumptions made, and explore the trade offs involved.

A. Process characterization

The first step in modeling is the identification of the type of model. While this would need analysis of a large volume of sensor data, it is also dependent on the type of sensor and the application, and can be performed offline. Here we have examined the issues that need to be considered in making the choice of the model, and the trade offs involved. The estimation of model parameters may be performed offline or online, and is described in the next subsection.

Input for modeling: One issue that needs to be addressed while choosing a data model for any given sensing application is whether to model the raw data or some processed form of the data. Many sensing applications perform some initial processing on the raw data reported by the sensors *e.g.* depending on the application, the samples from an accelerometer may be integrated once or twice before performing any computation on them. In such cases, it may also be more efficient to model the final integrated values for the purpose of correction.

Nature of model: Since we are trying to make the predictions on the basis of the temporal correlation in the data, the applicable models will have the general form of a function of previous samples. One such model is the linear combination of a few immediately preceding samples, which we have used in our examples during evaluation. The coefficients of such a model, *e.g.* the *autoregressive* model of order M : $AR(M)$ can be estimated by a linear regression analysis of previous samples.

Modeling accuracy: While choosing a model, the primary requirement is a high prediction accuracy. One way to ensure this is by maximizing the accuracy of the model itself. This is possible only in an ideal case, when the temporal correlation in the source is captured completely in the model. For such a model, the modeling error will reflect the inherent randomness in the data generation process. In a real implementation, the accuracy and adequacy of the model will be limited by the amount of data available for modeling, the correlation present in the data, as well as computational resources. In addition to the properties of the physical process, the correlation in the sensor data also depends on the sampling rate. Here, we assume that the sensor is sampled at a rate adequate to capture the highest frequency of interest in the data source.

Overall accuracy: Apart from a high prediction accuracy, another important property of the model is how the accuracy is affected by errors. In a real application, it is expected that due to errors in the processing or communication stages, some of the samples may not be available or contain arbitrarily high levels of errors. In order to ensure that the prediction accuracy is minimally affected, it is better if the model is dependent on a large number of previous values or makes use of trends computed over large periods.

Energy and performance bounds: A conflicting requirement

may be imposed on the models by the computation and storage costs in the μ -gateway nodes which use the models for prediction and correction. Even though a μ -gateway will have available higher amount of computational and energy resources than the sensor nodes, it will still be responsible for the on line correction of data from all sensor nodes in its cluster in real time. For each sensor data sample received from each source, it will need to perform the prediction and decision making steps apart from other modeling related tasks within the fraction sample period allocated per sensor node. It is thus important that the prediction process be computationally simple, and the choice of model involves a trade-off to be made of accuracy of prediction against the complexity of the prediction process. In our experiments, we have chosen to use linear autoregressive models to minimize the complexity of prediction.

Storage bounds: Another factor that may be influenced by resources available at the μ -gateways is the amount of history that the model uses. A typical example might be a temperature sensor sampled every second, which shows a periodicity of 24 hours. Now, while it might have been attractive to choose a model that captures this periodicity, in practice it might not be possible because of limited storage available at the μ -gateway.

B. Model parameter estimation

For certain types of sensing applications like environmental sensing, the nature of the physical process and the limitations on the size of history that can be locally accessed may not favor the approach of computing a model offline and using it forever. This is because there may be 'seasonality' in the data, which means that the same model may not satisfy the modeling accuracy goals over the lifetime of the application. In such cases, it becomes necessary to recompute or update the existing model to reflect the current characteristics of the sensor data.

To handle such types of processes, we assume here that for purposes of updating the model on line, the sensor node can transmit the data for short periods with extra protection against errors in the communication channel. We do not make any assumptions regarding the kind of protection available, except that the extra cost per bit of protected data transmitted and the error bound on the protected data are known at the receiver. We thus allow the μ -gateway to move any particular sensor node from the normal reporting state to a protected reporting state, where any extra temporary measures for protecting the samples can be used. We henceforth identify a system operating in this mode as to be in an *Estimation mode* as opposed to the normal *Correction mode*.

If we follow the same approach for on-line remodeling as with the initial offline modeling phase, it will affect the performance as well as resource budget of the sensor nodes adversely since we would need to obtain a large set of data that has been protected during transmission at the cost of computation and transmission overheads at the sensor node.

We propose to avoid this problem by updating an existing model. For instance, for an AR model, we do not change the order of the model, but re-estimate the coefficients at the μ -gateway. This would lead to a lower cost of update if it can be accomplished using a less number of points. Another possible approach may be used to address the 'seasonality' if it can be identified during offline modeling that the process moves around between a set of known states. The online processing will then consist of identifying the current state of the source and choosing the appropriate model or parameters for that state.

One of the practical questions that need to be addressed as a part of the model selection and estimation process is how to determine when to stop estimating. For run-time estimation, as we increase the number of samples used for estimation, it may increase the accuracy of the model. At the same time sending the sample with extra protection incurs an overhead at the sensor node transmitting it, and it is also one less sample the μ -gateway corrects. In our implementation, we have used the RMS prediction error over the window of protected samples as a measure of the adequacy of the model. Thus, the system moves from estimation mode to correction mode when the average modeling error over the current estimation mode window falls below a preset threshold.

In order to effectively handle such processes whose properties change at run-time, the μ -gateway also needs to identify significant changes in the sensor data as they occur. In order to track model accuracy, we observe the error between predicted sample and received sample only when a observed sample is thought to be correct. The prediction error is used to compute a running windowed average of the estimation error. A comparison of the average estimation error with a fixed threshold gives a simple way to trigger model changes. However, such an approach requires a careful choice of the two parameters: averaging window size and threshold, which has been discussed with an example in section VI. The choice of the above two parameters determine the frequency of updates under the given conditions, which in turn affect the performance and costs of the overall system. Thus the system design can be tuned through the two parameter to optimize for overall performance and cost.

V. PREDICTIVE ERROR CORRECTION

As described in section III, given a predictive model characterizing the sensor data, the goal of the error correction method is to report corrected output samples using erroneous data samples as input.

A functional diagram of the data correction block is shown in Fig. 3. The data correction method has two main control components: *prediction* and *decision*. It also maintains state information required for modeling and prediction, such as observation history and prediction history. correction block shown in Fig. 3 which consists of control elements for *prediction* and *decision making*, as well as maintains state

information as histories of observations as well as predictions. It is assumed that a model of the data is already available, which can be used by the prediction block along with some of the data history to predict the next sample. The task of the decision block is to choose between the observed and predicted values. One way to make this decision is by using the difference between the predicted and the observed value, *i.e.* the prediction error which is expected to be high if a random bit of the observed data is flipped during processing or transport. However, for any model for a random process estimated with a finite input set, there will be a certain amount of prediction error due to the inherent randomness in the process. The main challenge for the correction algorithm is thus to accurately decide whether to attribute a prediction error to random variations in the source or errors in the processing or communication channel. In order to improve the efficiency of the correction, we use multiple samples to guide the process of correction. This is done by examining how each decision in a sequence would impact the trends of future prediction errors and making the decision after a delay of few sampling intervals.

The prediction history (shown in Fig. 3) is stored in a tree structure, called *Prediction History Tree (PHT)*, which holds all possible sequences of observed and predicted values along with their error metrics for past few sensor samples. The decision-making algorithm uses the PHT to guide the correction process. The depth of the PHT is decided by a key parameter in the decision algorithm, called *decision delay*, and its choice is bounded by the delay tolerance of the application. In the rest of this section we describe the PHT and the decision algorithm in further detail.

A. Prediction History Tree

The prediction history tree is a complete binary tree with $N + 2$ levels, where N is the *decision delay* parameter for the algorithm. The observed and corrected values for the k^{th} sample are represented by $X(k)$ and $X_c(k)$ respectively, while $X_p(k)$ represents the value predicted by the model for k^{th} sample. We assume that $n - 1$ samples have been observed at this point. The nodes in each level l , starting with the root on level 0, then contain the possible sample values for $X(n - N + l - 1)$. with $X(n)$ being placed on the last level $N + 1$

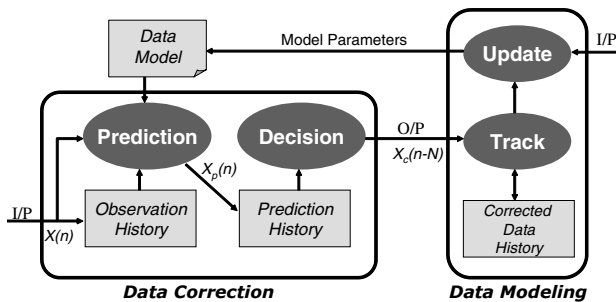


Fig. 3. Functional diagram of predictive error correction

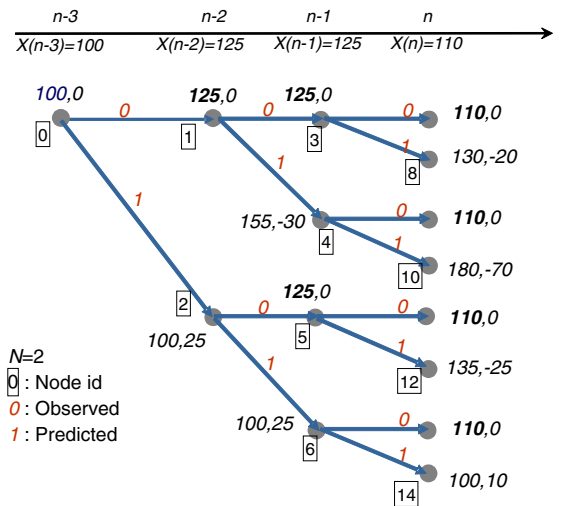


Fig. 4. Prediction History Tree with example data ($N = 2$)

of the tree. We call the subtrees rooted in node 1 and 2 as the *observation subtree* and *prediction subtree* respectively.

An example of a PHT for $N = 2$ is illustrated in Fig. 4. For the purpose of explaining the algorithm below, we have numbered the nodes of the PHT starting with the root as node 0. Then, for each node i , the child with the observed value for the following time period is numbered $2i + 1$, and the one with the predicted value is $2i + 2$. The root contains $X_c(n - N - 1)$, the last corrected value, and the even numbered leaf nodes contain the different predicted values of $X_p(n)$ that would be computed for different choices of previous values. After receiving the new observed sample $X(n)$, the prediction errors for all the values of $X_p(n)$ are computed, and the decision algorithm tries to decide the correct value for $X_c(n - N)$ from the observed and predicted values in nodes 1 and 2 respectively. This is followed by the PHT update procedure, where the chosen node becomes the new root node, and all the nodes in its subtree move up by one level. A new level of leaf nodes is added with the even numbered ones containing the next set of predicted values $X_p(n + 1)$, and the observed $X(n)$ is inserted in the odd numbered nodes for level N .

B. Decision algorithm

The goal of the decision algorithm is to choose the appropriate value for the $(n - N)$ th sample out of the observed and the predicted values by looking at the possible outcomes of each of this choice. The sequence of nodes on each root-leaf path of the PHT contain either an observed or a predicted value at each node, representing a sequence of such choices. The decision algorithm uses the prediction errors for the nodes on each path as a measure of its likelihood of being chosen as the final path.

In [1], we had introduced a simple decision algorithm which computes the average prediction error over the path from the root node to each leaf in the PHT, and chooses the path with

the minimum root mean square RMS error. Whichever level 1 node is contained in the minimum error path is chosen as the new root. The algorithm, which we hence refer to as *MinErr*, assumed that an external oracle provides it with a accurate model of the data properties.

However, there are two major problems in applying the above-mentioned approach to realistic data models. First, it assigns equal weights to all the paths. It is important to note that it is the nodes with predicted values that can provide an estimate of the error, which is used to decide whether $X(n-N)$ is erroneous or not. The impact of random modeling error, if present, is highest on the error estimates provided by path which have only a small number of predicted samples. For example, the average error over the path which contains observed values except for the last one [e.g.path root:node 8 in Fig. 4] will be the same as the modeling error for the latest sample. As a result, if the modeling error in the latest value ($X(n)$) is very small or zero, it will lead to the choice of the observed value irrespective of the errors on any other path.

This problem was corrected in the *MinMax* algorithm [Fig. 5(a)], where the two subtrees of nodes 1 and 2 are considered separately and the path with the maximum average error in each subtree are found. The subtree which has a smaller maximum error is then used to choose the corrected value.

A second problem is that none of the above algorithms are aware of the properties of the data model. The predictive model computes the next sample as a function of some previous observations. So if the model is such that the prediction for $X(n)$ is not dependent on the value of $X(n-N)$, then including that node in the decision process just makes the result more vulnerable to variations in the modeling error. For example, when an *AR(2)* model is used with a tree with $N = 3$, the prediction in the leaf node on a path with the levels $n-1$ and $n-2$ containing observed values will not be dependent on the choice in level $n-3$.

To address these issues we propose a new algorithm called *Peer* [Fig. 5(b)]. Here, in order to compute the path errors, we use available knowledge of the predictive model to exclude all those nodes which cannot be influenced by the value chosen for $X(n-N)$. Also, instead of looking at all paths, the comparisons are now made between peer nodes in the observation and prediction subtrees in order to minimize the effect of the perturbations in the modeling error.

It should be noted that the all of the algorithms perform well when the model manages to predict the observations very closely. The difference among them lies in the amount of modeling error they can tolerate while continuing to deliver acceptable correction performance.

The run-time cost of all three are similar. The time complexity is of the order of $\Theta(N \cdot 2^N)$. The space complexity is the size of the PHT, which is $\Theta(2^{N+2})$.

Algorithm : MAX_MIN(*PHT*, *N*)

```

comment: Find Max. Error in Obs. Subtree
obsErr ← 0
for  $n \leftarrow 2^{N+2} - 1$  to  $2^{N+2} + 2^{N+1} - 1$ 
  comment: Find Avg. Path Err. from 1 to n
  err ← PATHERROR(PHT, 1, n)
  if  $err > obsErr$ 
    obsErr ← err
comment: Find Max. Error in Pred. Subtree
predErr ← 0
for  $n \leftarrow 2^{N+2} + 2^{N+1}$  to  $2^{N+3} - 2$ 
  comment: Find Avg. Path Err. from 1 to n
  err ← PATHERROR(PHT, 2, n)
  if  $err > predErr$ 
    predErr ← err
if  $obsErr < predErr$ 
  return (0) comment: Use observed value
else
  return (1) comment: Use predicted value

```

(a) Min_Max Algorithm

Algorithm : PEER(*PHT*, *N*, *M*, *ETH*)

```

for each  $n \in$  Prediction Subtree
  comment: Check if n has the root node in its history
  comp ← CANCOMPARE(PHT, n, M)
  if  $comp == 1$ 
    comment: Find pair sibling of n
     $s =$  PAIRSIBLING(n)
    if  $abs(PHT[n].err) > abs(PHT[s].err) + ETH$ 
      count ← count + 1
    if  $abs(PHT[n].err) < abs(PHT[s].err) - ETH$ 
      count ← count - 1
  if  $count > 0$ 
    return (1) comment: Use predicted value
  else return (0) comment: Use observed value

```

(b) Peer Algorithm

Fig. 5. Pseudo-code for Min_Max and Peer Algorithms

VI. EXPERIMENTAL EVALUATION

In this section, we present the result of modeling the correlation in the available real sensor data, and illustrate the need for re-estimation of the model's parameters at run time. We then compare the performance improvements shown by the two proposed algorithms for a few example data sets. Finally we examine the effect of updating models at run-time on the overall correction performance. Before presenting the results of our comparison, we start with a brief description of our method of evaluation, the types and sources of sensor data, and the metrics used.

Evaluation Setup

In order to evaluate our model-based correction technique, we implemented our algorithms in C and Matlab ([4]), and evaluated their correction performance on real sensor data from different sources, under different levels of simulated error conditions. The specific data sets considered in the evaluation are listed in Table I along with their sampling period and the number of samples available. The sources of the data in Table I include an indoor light-level sensor from a testbed network [1], environmental temperature and humidity sensors from the California Data Exchange Center (CDEC) [2], and server rack temperature variations collected from a commercial data center. Each of the data sets report information about the sensed physical process as values quantized to 8 bit samples, *i.e.* the data values range from 0 to 255. It is important to note that the data sets considered differ in terms of autocorrelation properties and degrees of stationarity, which provided an opportunity for evaluating the correction performance under real-world limitations of the model.

For the lack of better models available representing transient error statistics in a sensor network, we considered errors which would be generated by a Bernoulli process for each transmitted bit. Assuming that the bit will be received in error if the outcome of the Bernoulli trial is successful, the probability of success of this error-generating process will be the bit error rate p_e . We use this model, with p_e varying from 10^{-4} to 10^{-2} , to represent both the communication errors as well as transient errors in the node.

The metric that we use to evaluate the correction performance is the ratio of the correction error to the magnitude of the sensor values. Extending the notation described in section V.A, let $X_s(k)$ be the k -th sample originally generated by the sensor, and NN be the total number of sample values in a

TABLE I
LIST OF DATA SETS

Data Set	Sensor Type	$1/f_s$	# samples
1	Data Center: Temperature	1 min	425
2	CDEC Alpine: Temperature	1 hr	398
3	CDEC Alpine: Humidity	1 hr	398
4	Testbed: Light	0.2 min	30000

TABLE II
MODELING PERFORMANCE FOR DATA SETS

Data Set	Model order	Modeling error (%)
1	4	0.021
2	22	0.430
3	4	1.150
4	2	0.007

data set, we define the metric as the per-cent error in corrected output X_c with respect to input as

$$E_{out} = 100 \cdot \sqrt{\frac{1}{NN} \cdot \sum_{i=1}^{NN} \left| \frac{X_s(i) - X_c(i)}{X_c(i)} \right|^2}$$

A corresponding metric E_{in} is computed for input errors by replacing X_c with X .

Data Modeling

We have used AR models for predicting the sensor data for the different data sets mentioned in Table I, however the order of the model is different for different data sets. In our initial approach to model computation, we were using a initial set of sensor values to compute the model which would then be used for the rest of the data. The order of the model is chosen by selecting the one which minimizes the modeling error over the given data set. The models for each of the data sets computed this way are shown in Table II. The table illustrates that the best fit data model depends on the data source as well as sampling rates.

Table III shows the effect of updating the model parameters at run-time on the modeling error for the example data set 4. The first row shows an approach where all the available data are used to create the model, with the data correction applied on the the same range of data. While this is impractical to use, it provides an estimate of the modeling error that can be attained by using the largest possible amount of data to be used for model updates. The second model $A2$ is estimated using only the early portion of the data (samples 550-700), and shows an increase in modeling error when used for later samples (1550-1700 or 1700-2000). Estimating the model again at 1550-1700 ($A3$) shows an improvement in prediction for subsequent samples. The above results illustrate the need for a way of modeling different parts of the data differently. As mentioned in section IV, we do this by run-time re-estimation of the model parameter. The effect of this on the overall correction performance is shown later.

Data Correction Evaluation

Fig. 6 shows temporal variation of original sensed data, data with errors and the corrected data for data set 4. The bit error rate (BER) is 10^{-2} . The plots demonstrate the nature of the effect the error process would have on a sensor data set, and provide a qualitative idea of the error correction performance of the algorithm. Some of the errors that were not corrected

TABLE III
NEED FOR MODEL UPDATES: DATA SET 4, ORDER 4

Estimation range (indices)	Data Prediction Model	Prediction range	% Prediction error
1-3251	$A1 = 1 - 1.06q^{-1} - 0.027q^{-2} - 0.07797q^{-3} + 0.1649q^{-4}$	550-700	0.0082
		1550-1700	0.0432
		1700-2000	0.0967
550-700	$A2 = 1 - 0.4935q^{-1} - 0.0974q^{-2} - 0.1883q^{-3} - 0.2208q^{-4}$	1550-1700	0.0761
		1700-2000	0.1571
1550-1700	$A3 = 1 - 0.9714q^{-1} - 0.3835q^{-2} - 0.02821q^{-3} + 0.3829q^{-4}$	1700-2000	0.1038

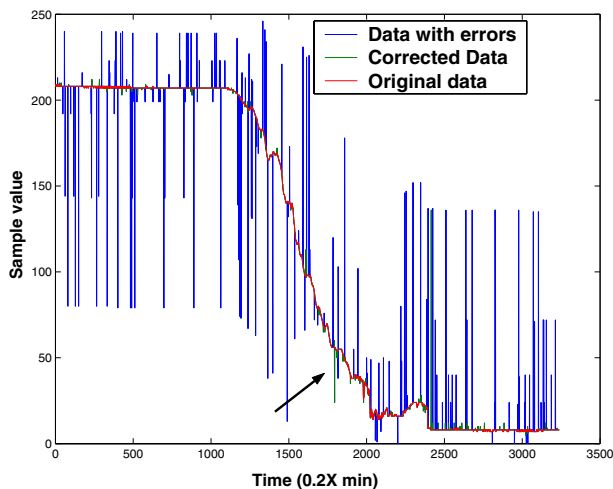


Fig. 6. Error correction detail

happen to occur near a point where the sensor data is also changing fast, in which case the modeling error is more likely to be high. One such case is shown with an arrow in the figure.

The plots in Fig. 7(a) and 7(b) show the relative performance of the three algorithms when the same model is used without run-time re-estimation. The final correction error as defined before is plotted against the error level in the observed data. It demonstrates the improvement achieved over the MinErr algorithm by the MinMax and Peer algorithms when the level of the errors in the input are high. We observe the difference in comparative performance in two data sets which are different in the distribution of values and the existing temporal correlation. From Table II it can be seen that the modeling error for data set 2 is higher than set 4. This agrees with the observation that the final correction errors for data set 4 are smaller, too. Also, there is a knee in all the curves above which the output error starts to grow at a faster rate, signifying the point where the effects of the modeling error and observation errors have similar characteristics. The position at which it occurs for a given algorithm is an increasing function of the modeling error.

It can be observed from the plots that while both the algorithms can correct most errors when the error level on the input is high, there is a non-zero residual error in the output, even at very low or zero input error levels. The source of this residual error is in the inaccuracy of the model, or in other words in the uncorrelated/random component of the data. The portion of the modeling error actually passed through to the output depends on the probability of false positive outcomes of the decision engine (where a positive outcome chooses that the predicted value). Upon closer examination of the output vs the input data, examples of this was revealed to occur when there were sharp and narrow peaks in the data itself. The decision algorithms interpret them as erroneous since the line quickly falls back to the previous trend, very in appearance much like transient error. It may be possible to reduce the level of the residual error by increasing the sampling rate so that no correlated variations in the process are interpreted as errors. However, that will decrease the battery life of the sensor node.

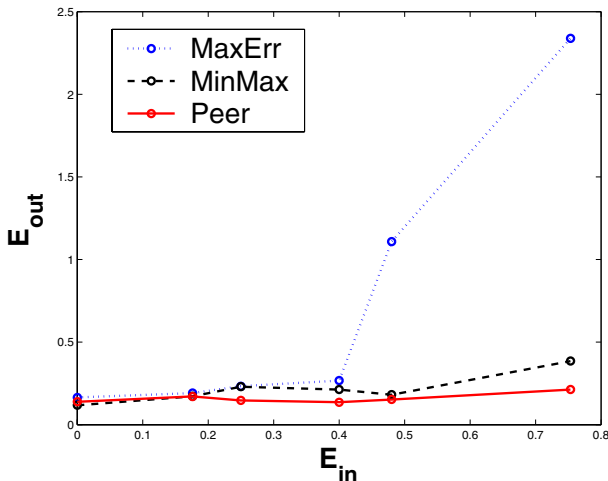
Effect of run-time model estimation

The following plot in Fig. 8 shows the effect of putting run-time model updates together with the Peer algorithm. The plot shows part of the same data with same error level as in Fig. 6, but with on line model parameter re-estimation added. The plot shows a better correction performance than the previous case. The frequency of estimation is indicated in the plot by the algorithm state parameter, where 0 and 128 denote model estimation phase and data correction phases respectively.

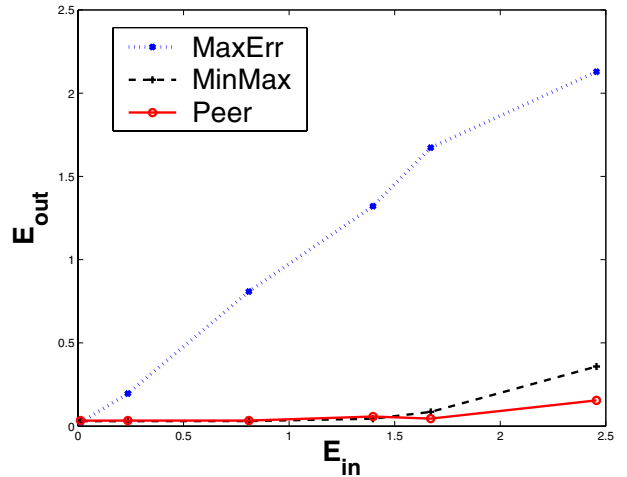
Table IV shows the performance improvements obtained for two data sets by using model updates. It also shows the overhead for extra estimation, defined as the ratio of number of samples used for model updates and those corrected using

TABLE IV
EFFECT OF DYNAMIC MODEL UPDATES

Data set ID	E_{in} (%)	E_{out} w/o updates (%)	E_{out} with updates (%)	Factor of improvement (E_{in}/E_{out})	Modeling overhead (%)
1	0.83	0.012	0.008	1.5	1.4
1	1.58	0.077	0.034	2.3	3.0
4	0.89	0.030	0.022	1.4	7.5
4	2.40	0.240	0.042	5.7	11.1



(a) Data Set 2 $DD=4$, $Order=9$



(b) Data Set 4 $DD=4$, $Order=2$

Fig. 7. Comparison of correction performance

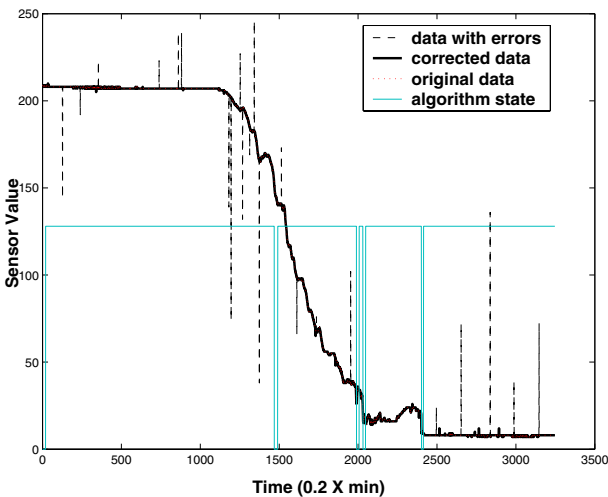


Fig. 8. Effect of model updates on correction

the model. It is observed that when the error at the input is higher, the improvement attained with model updates for a given data set is higher. This is expected since in presence of a high modeling error, larger errors in the input are likely to be passed through the correction block, thus lowering the performance more.

VII. RELATED WORK

In this section, we present past research attempts that address unreliabilities in sensor networks and compare our approach with them.

Fault tolerance has been an active field of research in many research communities. One of the ways to tolerate faults is to introduce redundancy through replication at node or circuit

level ([5],[6]). For instance, in [5], author suggests a design of fault-tolerant sensor by replication of physical sensors that can be configured based on failure models. However, replication at circuit level is not viable for large scale sensor networks as it leads to increase in the cost of deployment. Similarly, errors in communication channels can be addressed by general error correction techniques used in data communication networks, such as various types of Forward Error Correction(FEC), *i.e.* Reed-Solomon, Turbo codes, along with retransmission techniques [7]. However, unless it is accompanied by appropriate source coding, use of channel coding may lead to a significant overhead in terms of bits being transmitted, hence increases the energy consumption. The use of source coding schemes such as [8], is an alternate option to decrease the overhead cost by reducing the traffic volume generated from the node. However, we believe that the use of compression techniques will require higher computational resources at the sensor node, and will thus increase the cost of the nodes.

Apart from using some generalized techniques to address unreliabilities, a few techniques have been proposed recently that address the issue at different levels specifically for sensor networks. At the application level, techniques have been proposed to address the goal of the application in spite of erroneous samples collected from sensors, such as a distributed Bayesian algorithm for event-region detection application [9], query processing in presence of erroneous data [10]. Network level unreliabilities, such as node failures, have been addressed by error-resilient techniques for ad-hoc routing ([11]) and transport layer protocols([12],[13]). In this paper, we focus on providing a method to reduce errors in the data received from each sensor, hence can be complimentary to any of the upper layer techniques mentioned above.

In this work, we focus on correcting transient errors by exploiting sensor data properties. Several techniques have been

proposed which use the predictive nature of sensor data to develop efficient compression algorithms ([8]), reduce memory usage for routing algorithms ([14]) etc. Some of the non-transient errors such as mis-calibration and bias have been addressed by exploiting the relationship between co-located sensors ([15]). The goal and approach of the presented work is very similar to the techniques presented in [16] and [17]. However, there is a difference in the methodology. While in [16], the knowledge of the error process is actively used, we use the model of data property to clean erroneous data. Since our proposed approach does not assume any distribution for the noise process, it can address any type of transient errors including errors in sensor, communication errors and DSM error in hardware.

VIII. CONCLUSIONS

In this paper, we enhanced a model based error correction framework presented earlier to correct transient errors in the data received from the sensor nodes. Through simulation-based study on real sensor data, we demonstrate that with the proposed enhancements, the presented framework can be used efficiently to address transient errors in different types of sensor data across diverse applications. We plan to extend the approach to consider other methods of data modeling, such as probabilistic models, and to address other sources of unreliabilities, such as sensor bias.

IX. ACKNOWLEDGMENTS

This work was supported by UC Discovery Grant com02-10126 and the Center for Wireless Communications, UC San Diego. We also wish to thank Salil Pradhan and Malena Mesarina of Hewlett Packard Labs, Palo Alto for making available thermal sensor data from their data centers, as well as the California Data Exchange Center for making their environmental sensor data [2] publicly available.

REFERENCES

- [1] S. Mukhopadhyay, D. Panigrahi, and S. Dey, "Data aware, low cost error correction for wireless sensor networks," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2004, pp. 2492–7.
- [2] "CDEC: California data exchange center," California Department of Water Resources. [Online]. Available: <http://cdec.water.ca.gov/>
- [3] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, "Application driven systems research: Habitat monitoring with sensor networks," *Communications of ACM Special Issue on Sensor Networks*, pp. 34–40, June 2004.
- [4] "MATLAB: A high-level technical computing environment," <http://www.mathworks.com/products/matlab>.
- [5] K. Marzullo, "Tolerating failures of continuous valued sensors," *ACM Transactions on Computer Systems*, vol. 8, pp. 284–304, 1990.
- [6] Y. Zhao and S. Dey, "Separate dual transistor register: A circuit solution for on-line testing of transient errors in uds-m-ic," in *Proc. of Intl. On-line Testing Symposium*, 2003.
- [7] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*. Prentice Hall, 1995.
- [8] S. S. Pradhan and K. Ramachandran, "Distributed source coding: Symmetric rates and applications to sensor networks," in *Proc. IEEE Data Compression Conference (DCC)*, Mar. 2000.
- [9] B. Krishnamachari and S. Iyengar, "Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks," to appear in *IEEE Transactions on Computer* 2004.
- [10] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek, "Beyond average: Towards sophisticated sensing with queries," in *Proc. of Intl. Workshop on Information Processing in Sensor Networks (IPSN)*, Mar. 2003.
- [11] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly resilient, energy efficient multipath routing in wireless sensor networks," *Mobile Computing and Communications Review*, vol. 1, no. 2, 2002.
- [12] C. Y. Wan, "PSFQ: a reliable transport protocol for wireless sensor networks," in *Proc. of 1st ACM Intl. Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.
- [13] F. Stann and J. Heidemann, "RMST: Reliable data transport in sensor networks," in *Proc. of 1st Intl. Workshop on Sensor Network Protocols and Applications (SNPA)*, 2003.
- [14] M. Drinic, D. Kirovski, and M. Potkonjak, "Model based compression in wireless ad-hoc networks," 2003.
- [15] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak, "A collaborative approach in in-place sensor calibration," in *Proc. of Intl. Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.
- [16] E. Elnahrawy and B. Nath, "Cleaning and querying noisy sensors," in *Proc. of Workshop on Sensor Networks and Applications (WSNA)*, 2003.
- [17] —, "Context aware sensors," in *Proc. of European Workshop on Sensor Networks*, 2004.