

Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 05-018

dSENSE: Data-driven Stochastic Energy Management for Wireless
Sensor Platforms

Haiyang Liu, Abhishek Chandra, and Jaideep Srivastava

May 09, 2005

dSENSE: Data-Driven Stochastic Energy Management for Wireless Sensor Platforms

Haiyang Liu , Abhishek Chandra and Jaideep Srivastava

Department of Computer Science
University of Minnesota
{hliu, chandra, srivasta}@cs.umn.edu

Abstract

Wireless sensor networks are being widely deployed for providing physical measurements to diverse applications. Energy is a precious resource in such networks as nodes in wireless sensor platforms are typically powered by batteries with limited power and high replacement cost. This paper presents *dSENSE: a data-driven approach for energy management* in sensor platforms. dSENSE is a node-level power management approach that utilizes knowledge of the underlying data streams as well as application data quality requirements to conserve energy on a sensor node. dSENSE employs *sense-on-change*—a sampling strategy that aggressively conserves power by reducing sensing activity on the sensor node. Unlike most existing energy management techniques, this strategy enables explicit control of the sensor along with the CPU and the radio. Our approach uses an efficient statistical data stream model to predict future sensor readings. These predictions are coupled with a stochastic scheduling algorithm to dynamically control the operating modes of the sensor node components. Using experimental results obtained on PowerTOSSIM with a real-world data trace, we demonstrate that our approach reduces energy consumption by 29-36% while providing strong statistical guarantees on data quality.

1 Introduction

Wireless sensor networks are being widely deployed for providing physical measurements to diverse applications [11] such as structure monitoring [6], habitat monitoring [26], vehicle tracking [22], and building monitoring and control [17, 10]. Most wireless sensor platforms are powered by batteries that have limited life, and are subject to high replacement cost. Energy is thus a precious resource on sensor nodes, and its conservation is one of the main challenges in these wireless sensor network applications.

1.1 Sensor Energy Management

Recent advances in sensor platform hardware design have made sensor nodes more energy-aware. Many sensor platforms now allow their main components, such as the CPU, the

radio and the sensors, to have multiple operating modes with significantly different power levels. For example, the mica2 sensor platform [29] allows the CPU to be put into sleep mode where it consumes only about $\frac{1}{80}^{th}$ of the power consumed in active mode; the radio component on Telos consumes only 0.1% of its full power transmission energy [33, 28] when in idle mode. Similarly, the temperature/humidity sensor on the Telos platform [33, 34] draws only 1 μA of current in sleep mode, compared to 550 μA in active sampling mode. With the support of such energy-aware hardware design, it is now feasible to efficiently manage energy consumption on a sensor node by appropriately switching the operating mode of each component.

Most existing research efforts in sensor energy management have focused on optimizing the power consumption of the radio and the CPU [11, 3]. These efforts have been driven largely by the conventional wisdom that these components consume most of the power on a sensor node [11]. At the same time, energy conservation on the sensor component has received little attention. In reality, the power consumption of sensors critically depends on their modality [11]. For example, a high-power modality sensor, such as the heading sensor offered by xBow [37], can consume a power of about 375 mW, which is much higher than the 60 mW consumed by the mica2 radio transmitting at full power. Further, even low-power modality sensors, if not well-managed, could account for a significant percentage of the total energy consumption after aggressive CPU and radio energy management. Our experiments, presented in Section 5, reveal that the SHT series temperature/humidity sensor integrated on the Telos platform, that uses only 1.65 mW of power while sampling, could consume upto 38% of the total energy consumption. Thus, effective modulation of the sensor operating modes is crucial for better energy conservation. Moreover, reduced sensing activity enables the CPU and the radio to spend more time in sleep mode, thus resulting in even higher energy savings for these components. Therefore, we believe that *sensor power control is not only desirable, but essential for sensor platform energy management*.

1.2 Data Properties and Requirements

One major limitation of existing energy management approaches is that most of them do not consider the properties of the sensor data or application requirements. Sensors measure physical phenomena and produce data streams—sequences of data values observed over a period of time. Most physical phenomena are governed by physical laws, that result in high temporal correlations within these data streams. For instance, the temperature variation in a room is governed by heat transfer laws, which limit the amount of variation that can occur between two successive sensor readings of the room temperature. Such temporal correlations can be exploited for energy management by taking sensor measurements only when large variations are expected in the underlying data values.

The desired resolution of sensing is highly dependent on application requirements. Different applications have different data quality requirements from the sensor data streams [32]. For instance, a Heating, Ventilation and Air-conditioning Control (HVAC) application might require fine-grained temperature readings of a building within an accuracy of 1°C. On the other hand, a fire monitoring application may only be interested in determining whether the temperature is greater than a pre-defined threshold, and could afford to have coarse-grained accuracy in its temperature readings.

In addition, data quality requirements may change even for the same application over different time periods and for different value ranges [32]. For example, the HVAC application may require more precise readings during daytime when offices are occupied, while only coarse measurements might be sufficient at night when offices are empty. Similarly, the fire monitoring application might require higher data quality when readings are close to the threshold compared to when they are far away. Support for multiple data resolutions on sensor nodes also provides applications with an effective means to achieve graceful performance degradation [9] when the network is congested or the sensor nodes are constrained. In case of such constraints, the application can slow down the data sensing and transmission rates by reducing its data fidelity requirement. Thus, sensor node-level awareness of application requirements is essential for more efficient usage of the sensor resources.

The tradeoff between energy usage and data fidelity calls for a novel approach to energy management for sensor platforms. In this paper, we present *dSENSE: a data-driven approach for energy management* that utilizes knowledge of the underlying data streams to conserve energy on a sensor node, while satisfying the application data quality requirements. *dSENSE* is a node-level power management approach that employs *sense-on-change*—a sampling strategy that aggressively conserves power by reducing sensing activity on the sensor node. Our approach uses an efficient statistical model of the underlying physical phenomena to predict future sensor readings. These predictions are coupled with a stochastic scheduling algorithm to dynamically control the

operating modes of the sensor node components. Using experimental results obtained on an enhanced version of PowerTOSSIM [29], we demonstrate that our approach reduces energy consumption by 29-36% while providing statistical guarantees on data quality.

1.3 Research Contributions

This paper makes the following research contributions:

- *Data-driven sensor control*: Most existing approaches focus on minimizing data communication to reduce the energy consumption of the radio and the CPU. Our approach is novel in that it provides energy-efficient control of the sensing operations by incorporating both observation data characteristics as well as application data quality requirements.
- *Efficient data stream modeling*: We present a compact statistical representation for the underlying data streams that can be used to efficiently predict future sensor readings. Compared to many existing data models, our statistical models are ideal for sensor platforms that have severe storage and computational constraints.
- *Stochastic Sensor Scheduling*: We propose a novel stochastic scheduling algorithm that combines probabilistic data stream predictions with application data quality requirements to produce a sampling schedule for each sensor node. Our scheduling algorithm provides statistical guarantees on the application-specific data quality while substantially reducing the energy consumption of the sensor platform.

The rest of this paper is organized as follows. Section 2 describes the sensor and application data models, and introduces the terminology to be used in subsequent sections. The statistical data stream model is presented in Section 3, followed by a description of the stochastic scheduling algorithm in Section 4. Section 5 presents experimental results obtained on the PowerTOSSIM simulator using a real-world data trace. Related work is discussed in Section 6, and Section 7 concludes with a summary and future work discussion.

2 System Model and Terminology

2.1 Sensor Platform Architecture

The general operational cycle of a sensor node consists of sensing, processing, and transmitting data to other sensor nodes. Figure 1 illustrates the general architecture of a sensor platform. Table 1 shows the operating modes and power levels of the various components on Telos [33, 28, 34], a recently introduced low-power sensor platform. As can be seen from the table, there is a great deal of variation in the power consumption of these modes. The table shows that the components with the highest power consumption are the radio and

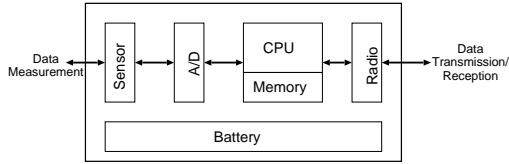


Figure 1: General architecture of a sensor platform: the main components are the sensor, the CPU, and the radio that together control the measurement and transmission of data. An A/D converter is used to convert sensor measurements to digital format, and a battery is used to power the various components.

| Component | Operating Mode | Current Drawn (μA) |
|----------------------|-----------------------|---------------------------------|
| CPU | Active | 1800 |
| | Idle | 55 |
| | Power save | 5 |
| Radio | TX (0 dBm) | 17,400 |
| | RX | 19,700 |
| | Power on | 365 |
| | Idle (Oscillator Off) | 20 |
| | Power down | < 1 |
| Temp/humidity Sensor | Measuring | 550 |
| | Sleep | 0.3 |

Table 1: The operating modes and power levels of different components on the Telos sensor platform.

the processor in active modes. This implies that substantial power saving can be achieved by avoiding wireless transmissions and receptions, as well as long computations. While the sensor power consumption is smaller in comparison, it can still add up to a large portion of the total energy usage if it is kept on for long durations of time. Moreover, sensing requires the processor to be in wakeup (active or idle) mode, and hence, sensing can indirectly result in higher CPU power usage. Thus, carefully controlling the operating modes of each component is essential for efficient energy consumption on a sensor node.

2.2 Data-Sensing Model

A sensor node typically measures the underlying process by sampling it periodically with a *base sampling frequency*. The base sampling frequency could depend on the physical limitations of the sensing device, the available communication bandwidth, or the highest temporal resolution required by the application. Thus the sampled data sequence at the base sampling frequency represents the closest approximation to the underlying (possibly continuous) process that could be achieved by a sensor node. We refer to this data sequence obtained by sensing at the base sampling frequency as the *baseline data sequence* (illustrated in Figure 2).

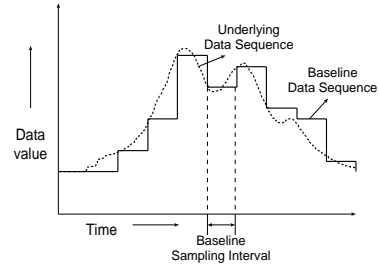


Figure 2: The baseline data sequence approximates the underlying process by sampling at the baseline frequency.

Send-on-Change: A common power-saving technique used in sensor networks is the *send-on-change* approach for data transmission [11, 30]. In this approach, new data or updates are sent out by a sensor node only when there is a “substantial” change detected in the underlying data values. Under this approach, if no new data is received by an application, it implicitly assumes the current data value to be the same as the most recently reported value. The goal of the send-on-change approach is to avoid redundant transmissions when there is little variation in data values.

The amount of variation in the data that drives the decision to send or not is application-dependent. We define a data value to have *changed* from its previous value if the difference between the two values exceeds an application-specified *resolution threshold* δ . The resolution threshold is the minimum change in the underlying data values that the application is interested in¹. We then define a *state change* to be a change in the data value exceeding the resolution threshold. Intuitively, a state change corresponds to an interesting sensor measurement that needs to be reported to the application.

Sense-on-change: Based on the above model, we propose a new data-sensing approach called *sense-on-change*. The goal of this approach is to make data measurements *only* when a state change is likely. This approach is an extension of the send-on-change approach in that it avoids redundant sensing operations in addition to avoiding redundant data transmissions. Thus, this approach could potentially save sensing and processor energy in addition to the transmission energy when there are no data changes to report.

However, with this new approach, it is possible to miss certain state changes if data was not sampled at those time instances. We refer to such missed state changes as *false negatives* or *misses*, as they correspond to a false expectation of not having a state change when actually there is one. Simi-

¹This definition of resolution threshold corresponds to a *relative* threshold, that is useful for tracking data variations (e.g., temperature variations of 1°C). On the other hand, we could also define an *absolute* threshold if we are interested in comparison to a specific value of interest (e.g., a temperature of 100°C).

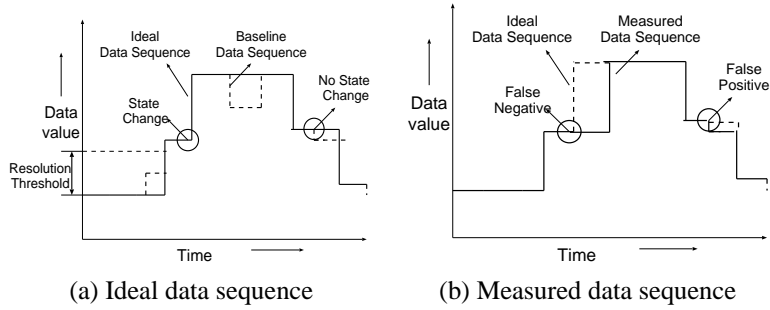


Figure 3: Data sequences produced by using different strategies under the sense-on-change sampling approach. (a) An ideal strategy samples data only when there is a state change based on the application’s resolution threshold δ . (b) A practical sense-on-change sampling strategy would try to emulate the ideal strategy, but would have some false negatives and some false positives, where it would deviate from the ideal sequence.

larly, it is possible for this sensing approach to make a measurement when there is no actual state change. We refer to such redundant sensing events as *false positives* or *false hits*, as they correspond to a false expectation of observing a state change when there is none. Figure 3 illustrates the notion of false negatives and false positives for a data sequence by comparing it to an ideal data sequence (that consists of samples taken only at state change points).

Note that a sensing scheme should strive to minimize *both* false negatives as well as false positives: while false negatives result in degraded data quality, false positives result in energy wastage. We define two quantities—the *miss ratio* μ and the *false hit ratio* ρ —to quantify the degradation in data quality and wasteful sampling respectively:

$$\mu = \frac{n_f}{n}, \text{ and } \rho = \frac{n_p}{n},$$

where, n_f and n_p denote the number of misses and false hits respectively, and n denotes the total number of sampling points (corresponding to the base sampling frequency).

Having defined the sensor architecture and data sensing models, we now present a statistical modeling and prediction technique for the underlying data streams.

3 Data Modeling and Prediction

The measurements made by a sensor node are discrete and form ordered data sequences, generally referred to as *observation data streams*. Most physical phenomena are governed by physical laws, resulting in temporal correlations within the observation data streams. These correlations constrain the amount of variation in the data value from one observation instant to the next. Our energy management technique exploits such temporal correlations to schedule sampling points as part of the sense-on-change data-sensing approach. Therefore, the first step in our sense-on-change approach is to build a data stream model that captures the intrinsic temporal characteristics of the physical phenomenon being observed.

Data stream models can be generally classified into two categories: *physics-based* models and *statistical* models. Physics-based models [1] are derived using detailed knowledge of the phenomenon being observed, such as the underlying physical laws and the configuration of the system under observation. These models can be very accurate when the system configuration is simple or the operational environment is well-controlled. However, it is difficult to get precise and accurate data models when the system under observation is complex and highly dynamic in nature, which is the case for many sensor network applications [10]. Statistical models are more suitable for such scenarios as they typically do not require any domain knowledge, and can be constructed easily using historical observations. While it is possible to build highly complex and detailed statistical models [5, 36], we present a simple and efficient model that is ideally suited for resource-constrained sensor platforms.

3.1 Statistical Data Stream Model

Any statistical model that we employ on a sensor node for predicting future data values must satisfy the following properties:

- The model must be *computationally efficient and compact* to represent. These properties are extremely important because of the limited computing power and storage available on common sensor platforms. Further, efficient prediction is also essential to reduce prediction overhead and energy consumption (which we are trying to minimize in the first place).
- Since any statistical model is based on historical observations, some of its predictions are bound to be inaccurate and deviate from actual measurements. Such uncertainty is inherent in any model that tries to predict a dynamic system. The model must, therefore, have the *ability to capture and quantify the uncertainty* inherent in its predictions.

- Under the sense-on-change approach for data measurement, it is possible to keep the sensor and the CPU in sleep mode for extended periods of time. During these off periods, the sensor node can neither update its data readings, nor can it predict subsequent data values until the sensor and the CPU are turned on again. To handle these periods of inactivity, the data stream model must be able to *predict data values upto k steps ahead* in the data sequence. The value of k would depend on the sensor node capability as well as application data requirements. Of course, as soon as the sensor node makes a new measurement, the new value could be employed by the data model to make fresh predictions.

Model Derivation

We now present a statistical data stream model that satisfies the above properties. Intuitively, in this model, we represent any data value by a probability distribution on its possible values². We then model the temporal variations in the data values to make predictions in the data stream.

Formally, we define the data model as:

$$X_{i+k} = X_i + D_k, \quad (1)$$

where, X_i denotes the data value at time instance i , X_{i+k} denotes the predicted data value at time $i + k$, i.e., k time steps forward from step i , and D_k represents the distribution of the difference between these values. Then, the model is completely defined once the distributions D_m , $m = 1, \dots, k$ are known.

One way to derive these distributions is by using a training data stream of length n : $X_i, i = 1, \dots, n$, and, for each value $m = 1, \dots, k$, computing the histogram of the m -step value differences

$$D_m(i) = X_{i+m} - X_i, i = 1, \dots, n - m.$$

Direct use of these histograms to represent the corresponding distributions would require large amount of storage on each sensor node. However, these distributions could be compactly approximated by normal distributions³, each of which needs only two parameters: the mean value μ and the standard deviation σ . The mean value of each normal distribution is estimated using the sample mean of the histogram and the standard deviation is estimated using the corresponding sample standard deviation. Then, given a (deterministic) data value X_i , Equation 1 can be rewritten as

$$\begin{aligned} X_{i+k} &= X_i + N(\mu_k, \sigma_k) \\ &= N(X_i + \mu_k, \sigma_k), \end{aligned}$$

²A deterministic value would be represented in this model by a distribution with a probability of 1 at the given value.

³We assume the values $D_k(i), i = 1, \dots, n - k$, to be i.i.d., and thus expect D_k to be normally distributed by the Central Limit Theorem [2].

where, $N(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ . Intuitively, the above model captures the data variation within k time steps as a random walk process.

Model Properties

This model satisfies the desirable properties we specified earlier. First of all, this model has a compact representation: it requires only k pairs of (μ, σ) values to be stored on each sensor node, along with a small table for the normalized unit normal distribution $N(0, 1)$. The model is also computationally efficient for prediction: any normal distribution $N(\mu, \sigma)$ can be transformed to the unit normal distribution $N(0, 1)$ using the simple relation $z = \frac{x-\mu}{\sigma}$. The prediction of subsequent data stream values then reduces to a table lookup. Secondly, the model implicitly captures the uncertainty of prediction by employing probability distributions in the estimation. These distributions enable us to estimate the likelihood of the occurrence of a specific data value. Finally, by using k different (μ, σ) pairs, the model allows us to predict values upto k steps ahead in the data sequence.

State Change Probability Evaluation

The above statistical data stream model allows the prediction of future data values. These predictions can then be used to determine the state change probabilities, based on the application's data quality requirement. Recall that an application's data quality requirement is specified in terms of a resolution threshold δ . Then, given the above prediction model, computing the state change probability is as simple as looking up the value of δ in the locally stored normal distribution table after appropriate transformations to the unit normal distribution.

3.2 Model Construction and Update

The statistical model described above must be constructed before it is stored on individual sensor nodes. Moreover, in order to capture changes in the dynamics of the observed data stream, the model may also have to be updated from time to time. We argue that such model construction and update should be carried out at base stations with sufficient computational and bandwidth resources. This is because sensor networks presents several challenges in carrying out these functions on sensor nodes:

- Each sensor node has only limited and incomplete view of the phenomenon being observed, especially when data at certain time instances are not sampled due to energy conservation concerns. Deriving and modifying the prediction model based on this incomplete view at individual sensor nodes is thus both inaccurate as well as undesirable.

- Estimating model updates on each sensor node may cause redundant work. For example, multiple sensors that are placed in close proximity would have similar models.
- Putting the burden of model update on each sensor node adds energy consumption overhead, which may prevent usage of complex models. This overhead is no problem for base stations with sufficient computing power and no energy constraints.
- Several physical phenomena exhibit long-range correlations such as time-of-day effects. The phenomena thus need to be modeled using long-term historical data, which is difficult to store on individual sensor nodes. However, this can be done much more effectively on base stations which have no storage constraints.

Updating a model is the same as its initial construction except for the use of a different training data set. New model parameters are sent from the base station to sensor nodes whenever the base station detects a significant deviation in the data streams. Since the prediction models take small number of parameters and are updated infrequently, the amortized communication cost of model updates is expected to be negligible.

4 Stochastic Scheduling Algorithm

4.1 Overview

The data stream model derived for an underlying process can be used to predict subsequent data values that are likely to occur based on the past observed values. These predicted values can then be used to determine the sampling points at which state changes are likely to occur and where the sensor node should be scheduled to sense new data values. For the remaining time periods, the sensor node can be put into sleep mode: the CPU in power-save mode, radio in power-down mode, and the sensors in sleep mode. In this section, we present a stochastic scheduling algorithm that employs the underlying data stream model to determine subsequent sensing instants. The goal of this scheduling algorithm is to minimize the energy consumption while meeting the desired data quality requirements of the application. Note that our stochastic scheduling algorithm does not depend on the specific prediction model being used, and can be used in conjunction with any kind of data stream model.

The sensor scheduling algorithm must satisfy several important requirements while determining the sensing points:

- Since energy is a precious resource in sensor platforms, the overall energy consumption of the sensing process should be minimized as much as possible. The overall energy consumption of a sensing process is the sum of energy spent in the sensors (for making measurements),

the radio (for sending required value updates when necessary), and keeping the CPU in power-on state, either active or idle, (for sensing, radio transmission and scheduling operations). The scheduling algorithm must try to minimize the energy consumption of all these components.

- Since data at some time points will not be sampled, it is possible to miss some of the state change events. In this case, the scheduling algorithm must ensure that the overall sampled data meets an application-specified data quality requirement. The data quality requirement of an application can be expressed in terms of the miss ratio μ , defined in Section 2.2, as this metric captures the accuracy of the measured data compared to the baseline data sequence. Thus, for instance, an application may require that the sensing scheme have a miss ratio of at most 5%, thus capturing 95% of the samples accurately.
- As sensing decisions must be made before a sample is taken, they are based on predicted information provided by the underlying data model. Since uncertainty is inevitable in predictions, state changes can only be predicted probabilistically. For such probabilistic events, deterministic scheduling would result in poor data quality or energy wastage. Therefore, scheduling decisions must also be stochastic and the probability of sensing should depend on the likelihood of state change as well as the data quality required by the application.

Overall, the scheduling algorithm must make a tradeoff between energy consumption and sampling quality in a stochastic manner that depends on the prediction model of the underlying process. Next, we formalize the scheduling problem and the various requirements discussed above.

4.2 Problem formulation

We formulate the stochastic scheduling problem as an optimization problem that minimizes the total energy consumption while providing statistical guarantees on data sampling quality.

Let us assume that the baseline data sequence consists of N data samples, and the probability of state change at a sampling instant i (determined using the underlying data stream model and the application's resolution threshold δ) is q_i . Further assume that the average energy⁴ spent for each measurement is e_{avg} . Finally, let the application's data quality requirement be expressed as a *tolerance level*⁵ $F_N \in [0, 1]$,

⁴The actual energy spent at each sampling instant consists of sensor, CPU, and radio power consumptions. This energy may vary between sampling points depending on which components are active (e.g., whether there is a transmission event when data is sensed). For simplicity of exposition, we consider an average value for this energy here, even though we accurately account for the actual total energy spent in our experiments.

⁵We define a corresponding metric *confidence level* as $(1 - F_N)$.

such that its miss ratio $\mu \leq F_N$. Then, the goal of the stochastic scheduling algorithm is to determine a probability of sensing $p_i \in [0, 1]$ for each sampling instant such that it minimizes the total energy

$$E = \sum_{i=1}^N p_i \cdot e_{avg} \quad (2)$$

under the constraint

$$\frac{\sum_{i=1}^N (1 - p_i) \cdot q_i}{N} \leq F_n. \quad (3)$$

The constraint given by Inequality 3 satisfies the statistical data quality requirement of the application as we require the *expected* miss ratio to be less than the application-specified tolerance level. Recall from Section 2.2 that the expected miss ratio, μ , is evaluated as the expected number of false negatives divided by the total number of data samples. Thus, given the false negative probabilities f_{ni} over all sampling instances, we have,

$$\mu = \frac{\sum_{i=1}^N f_{ni}}{N}. \quad (4)$$

Now, since the probability of sensing p_i is positively correlated to the probability of state change q_i at each sampling instant⁶, the false negative probability f_{ni} at each scheduling instant is less than what would be obtained by assuming independence between p_i and q_i . In other words,

$$f_{ni} \leq (1 - p_i) \cdot q_i.$$

Thus, the miss ratio (Equation 4) reduces to

$$\begin{aligned} \mu &= \frac{\sum_{i=1}^N f_{ni}}{N} \\ &\leq \frac{\sum_{i=1}^N (1 - p_i) \cdot q_i}{N} \end{aligned}$$

Thus, the constraint (Inequality 3) satisfies the data quality requirement $\mu \leq F_n$. In fact, the constraint is a conservative bound on the data quality requirement, such that if a schedule satisfies the constraint, it must also satisfy the data quality requirement.

4.3 Scheduling Algorithm

Having presented the formal problem formulation, we now present a stochastic scheduling algorithm that closely approximates the optimization problem. The goal of the scheduling algorithm is to determine the sensing probability p_i for each sampling instance given the state change probability q_i for that scheduling point. Given q_i , solving for the precise

⁶This can be seen from the fact that the higher the value of q_i , i.e., the probability of state change is, the higher the value of p_i , i.e., the chance of taking a sample would be.

value of p_i would require the joint distribution of the random processes of sampling and state changing. This distribution is neither available nor desirable due to its high storage and computational overhead. Instead, we simplify the computation of p_i as follows: we first determine the upper and lower bounds for p_i , and the scheduling algorithm then chooses a value from this range based on a heuristic we describe later. Intuitively, the upper bound of p_i specifies a limit such that selecting values higher than it would only waste energy for providing unnecessary data quality improvement. On the other hand, the lower bound of p_i corresponds to a limit, such that going below it would always result in violation of the application's tolerance level.

4.3.1 Determining the Upper Bound of Sensing Probability

To determine the upper bound on the value of p_i for a given q_i value, our scheduling algorithm performs local optimization instead of global optimization⁷. In other words, the optimization problem is reduced to minimizing p_i at each scheduling instant under the constraint

$$(1 - p_i) \cdot q_i \leq F_n,$$

which yields the solution

$$p_i^{ub} = \begin{cases} 0, & \text{if } 0 \leq q_i \leq F_n \\ 1 - \frac{F_n}{q_i}, & \text{if } F_n < q_i \leq 1 \end{cases}$$

This value of p_i is an upper bound on the value of the sensing probability, because, any sensing probability value higher than p_i^{ub} , while always satisfying the local optimization constraint, would be more wasteful of energy. In other words, p_i^{ub} is the *minimum* value of p_i that guarantees the satisfaction of the data quality requirement for *each* sensing instance.

4.3.2 Determining the Lower Bound of Sensing Probability

To determine the lower bound on the value of p_i , we consider the most optimistic scenario where every sample catches a real state change, i.e., there are no false positives. In this scenario, the data quality requirement can be satisfied only if $q_i - p_i \leq F_n$, or

$$p_i \geq q_i - F_n,$$

which provides us with the following lower bound:

$$p_i^{lb} = \begin{cases} 0, & \text{if } 0 \leq q_i \leq F_n \\ q_i - F_n, & \text{if } F_n < q_i \leq 1 \end{cases}$$

This value of p_i is the lower bound because any sensing probability value smaller than p_i^{lb} would always result in violating

⁷Note that local optimization meets a stricter requirement since satisfying the constraint at each sampling instance automatically satisfies the constraint over all sampling instances.

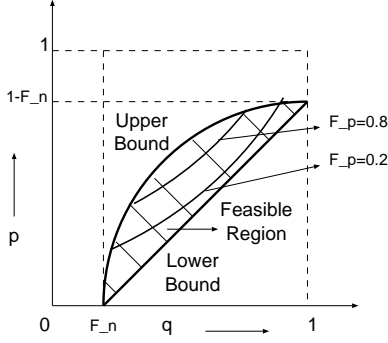


Figure 4: The relation between sensing probability p_i and the state change probability q_i . F_n is the tolerance level that determines the bound on data quality, while F_p is the false hit ratio threshold that is used as a tuning parameter to select a value from the feasible region bounded by the upper and lower bounds of p_i .

the data quality requirement. Thus, the value p_i^{lb} corresponds to the *smallest* value of p_i given q_i , such that the data quality constraint could be met.

4.3.3 Selecting the Sensing Probability Value

Given the upper and lower bounds on the value of p_i given q_i , we present a heuristic to select the actual value of p_i . Note that the application uses a miss ratio bound F_n to limit the data quality degradation. Analogously, we can bound the energy wastage by using a false hit ratio limit F_p . Our heuristic uses this limit F_p as the tuning parameter to determine the p_i value from the region bounded by p_i^{lb} and p_i^{ub} . Lower values of F_p correspond to more aggressive energy saving, while higher values of F_p provide better data quality at the expense of higher power consumption. F_p can be approximated as $F_p = p_i \cdot (1 - q_i)$, which yields

$$p_i = \frac{F_p}{(1 - q_i)},$$

subject to the two bounds derived above. The stochastic scheduling algorithm then uses this p_i value to probabilistically schedule a sensing event at a sampling point.

Figure 4 shows the relation between the sensing probability p_i and the state change probability q_i for a given value of the data quality threshold F_n . The figure also shows the intermediate values that p_i would take based on the value of the tuning parameter F_p .

4.4 Dynamic Adaptation

One limitation with any down-sampling scheme is that it provides no information about the unsampled data values, and it is possible to potentially miss unforeseen state changes. Since the estimation of the state change probability q_i in our

scheduling algorithm depends on the underlying data model as well as the recently observed data values, missing state changes could result in highly inaccurate q_i values. Such inaccuracy could further lead to poor scheduling decisions affecting the data quality of the application. Thus, it is important to ensure that our scheduling scheme adapts to sudden or unforeseen data variations.

While it is not possible to directly observe false negatives (corresponding to missed state changes), we can measure false positive rates to estimate the dynamism in the underlying data. Intuitively, a low rate of false positives implies that most of the sensing events result in state changes, suggesting the possibility of missing other significant changes. Thus, a low false positive rate could be taken as an indication of more dynamic data values, and the number of sampling events should be increased in this case to catch possibly significant state changes. On the other hand, if we observe a high rate of false positives, it means that we are taking large number of redundant samples, many of which are non-informative. Such a high rate indicates a relatively stable data process, and the sampling probability should be decreased in this case to save energy.

We use the tuning parameter F_p to achieve this dynamic adaptation of the sampling probability p_i . Recall that a higher value of F_p corresponds to higher tolerance to observing false positives, while lower values correspond to tighter bounds on the number of redundant samples. Thus, based on the intuition described above, whenever the observed false positive rate (the false hit ratio ρ) becomes low, we increment the value of F_p to allow more stochastic sampling events. On the other hand, when ρ increases to high values, we decrement the value of F_p to reduce the sampling probability and save energy. The selection of ρ is a tradeoff between energy efficiency and responsiveness of dynamic adaptation.

4.5 Practical Considerations

While we have described our stochastic scheduling algorithm in terms of sensing probabilities p_i , these probabilities have to be translated into the actual sampling schedule. We perform this translation as follows. Given p_i , a pseudo random number generator is used to generate a number $x \in [0, 1]$, and a sensing event is scheduled at the scheduling instance i if $x < p_i$. However, performing this translation at each scheduling instance is not efficient as the CPU would have to be turned on to perform the required computation, defeating the purpose of stochastic sampling in the first place. To amortize the energy cost of schedule generation, we generate a batch sequence $\{x_i\}$ of pseudo random numbers a priori. Then, at each scheduled sensing instant (when the CPU is turned on anyway for controlling the sensor), the stochastic scheduler determines the next sampling instant using the pre-generated random number sequence. The scheduler could cycle through the sequence again or generate a new sequence should the numbers be exhausted. The CPU (and other de-

vices) are then put into sleep mode until the next sampling instant.

5 Experimental Evaluation

5.1 Experimental Setup

We have implemented a dSENSE prototype on TinyOS and evaluated its performance for the Telos platform [28] in PowerTOSSIM [29], an extension of the TinyOS simulator TOSSIM [20] with energy estimation capabilities. The Telos platform was selected for its support for sensor power management: the temperature/humidity sensor on Telos is automatically put into sleep mode when no data is being measured.

We made several enhancements to PowerTOSSIM to enable accurate sensor control and power estimation. The current version of PowerTOSSIM does not support detailed sensor power estimation. Sensors are assumed to be always on, as is the case for the mica2 platform [29]. We extended PowerTOSSIM by adding new debug messages in the ADC reading interface used for sensor readings. These debug messages carry the channel number and the current time-stamp whenever an ADC reading request is issued. The energy model for the Telos platform is given in Table 1. An ADCDataLoading plugin was also developed for the TinyViz visualization tool to allow dynamic loading of data traces for sensor readings. This ADCDataLoading plugin also enabled catching all sensor-related debug messages to compute the corresponding sensor energy consumption.

We use real-world temperature readings to test the effectiveness of our prototype. These data were sampled in an air-conditioned storage room using a SHT11 temperature/humidity sensor[] at sampling frequency of 0.1Hz for two days. In order to capture greater temperature variations, the sensor was placed close to a ventilation exit. The collected data trace is shown in Figure 5. We used this data trace in our simulations, each of which was run for a simulation period of 10,000 seconds (corresponding to 1000 sample points). The simulation time period starts at the data point corresponding to about 6am on the second day in the trace, when air conditioning is configured to turn on in the room. This choice of data set results in richer variations in the test data. Each simulation was repeated multiple times with different random seeds and the arithmetic mean is reported.

5.2 Model Construction

Building environmental data such as temperature readings usually exhibits time-of-day patterns as can be seen from Figure 5. Therefore, we use the same time period of 10,000 seconds on the first day as the training set for constructing the data stream model (described in Section 3). Figure 6 shows the data histogram and the corresponding Gaussian distribution approximation for single-step prediction ($k=1$). This fig-

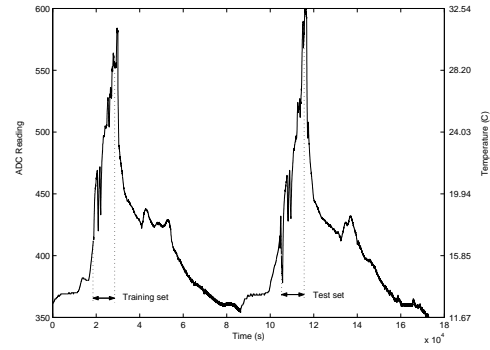


Figure 5: Data trace consisting of two days of temperature variation in a room.

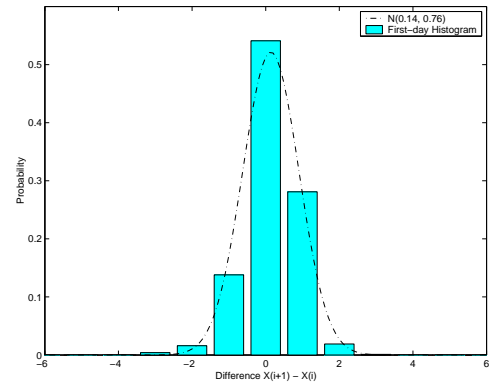


Figure 6: Gaussian approximation to the histogram of a single-step predictor ($k=1$)

ure illustrates that the Gaussian distribution is a close approximation of the histogram distribution. Table 2 lists the Gaussian distribution parameters corresponding to k -step prediction models, with k ranging from 1 to 20.

5.3 Performance Results

We now present the results obtained from our performance evaluation. We begin by measuring the overhead of dSENSE. We then present experimental results obtained by varying the different experimental parameters such as the application’s resolution threshold and the miss ratio confidence level. We present two cases: one with relative resolution thresholds and the other with absolute threshold specification. For each experiment, we examine the energy savings and the level of data quality provided. In our experiments, we compare the following energy management strategies:

- *Baseline sampling:* Baseline sampling corresponds to taking sensor readings at the baseline frequency of the sensor (as defined in Section 2.2). This sampling strategy uses a send-on-change approach for energy management, where it transmits data only when there is a state change. Rest of the time, the CPU and the sensor

| Time Step k | μ | σ | Time Step k | μ | σ |
|-------------|-------|----------|-------------|-------|----------|
| 1 | 0.14 | 0.76 | 11 | 1.54 | 5.79 |
| 2 | 0.28 | 1.29 | 12 | 1.68 | 6.25 |
| 3 | 0.42 | 1.80 | 13 | 1.82 | 6.70 |
| 4 | 0.56 | 2.23 | 14 | 1.97 | 7.14 |
| 5 | 0.7 | 2.86 | 15 | 2.11 | 7.58 |
| 6 | 0.84 | 3.37 | 16 | 2.25 | 8.00 |
| 7 | 0.98 | 3.87 | 17 | 2.39 | 8.42 |
| 8 | 1.12 | 4.37 | 18 | 2.53 | 8.83 |
| 9 | 1.26 | 4.85 | 19 | 2.68 | 9.23 |
| 10 | 1.40 | 5.32 | 20 | 2.82 | 9.62 |

Table 2: Parameters for the 20-step prediction model.

are in sleep mode and the radio is turned off. We use baseline sampling for comparison as it reflects the current best practice in sensor energy management. Note that baseline sampling represents the worst-case sampling strategy in terms of the number of samples taken, while achieving zero miss ratio for the application’s data requirement.

- *Ideal sampling:* Ideal sampling is a clairvoyant sampling strategy based on an application’s data resolution threshold that takes samples only when there is a state change. It further transmits data only at these sampling instances, and leaves the CPU and the sensor in sleep mode, and the radio turned off rest of the time. Thus ideal sampling consumes the least possible energy while achieving zero miss ratio.
- *dSENSE with upper bound sampling probability:* This is our stochastic scheduling algorithm that employs the upper bound of the sampling probability (described in Section 4.3) to determine the sampling points. Recall that this choice of sampling probability corresponds to the most conservative choice for meeting the application’s miss ratio guarantee.
- *dSENSE with lower bound sampling probability:* This is similar to the previous strategy except that it employs the lower bound of the sampling probability. This strategy is the most energy-efficient scheme while providing the most relaxed guarantees on data quality.
- *Dynamic dSENSE:* This strategy employs the stochastic scheduling algorithm with dynamic adaptation. It adapts to the current data stream conditions and thus provides a tradeoff between energy consumption and data quality guarantees.

dSENSE Overhead

The energy overhead in dSENSE consists of two main components. First, at each sampling instant, dSENSE incurs a fixed energy overhead due to the extra CPU time spent computing the state change probability, sampling probability, and

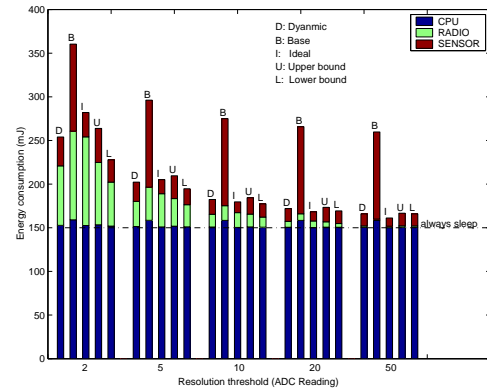


Figure 7: Energy consumption for a fixed confidence level = 90%.

generating a pseudo-random number for making a sampling decision. To compute this overhead, we used the CPU profiling tools in PowerTOSSIM to measure the number of CPU cycles used for these computations. It turned out to be negligible as, on average, only 660 cycles needed for each time point, corresponding to a total energy overhead of 0.89 mJ for the simulation period (less than 0.5% of the total energy consumption). The second energy overhead component is the energy needed for model update. However, as we discussed in Section 3.2, the model update would be typically done on a base station, and the model would then be loaded on the sensor node. Thus the only overhead incurred on the sensor node by this component is the energy spent in downloading the model, which is extremely small as, in practice, these models would either be pre-loaded or infrequently updated (weekly or monthly). We also measured the dSENSE memory overhead to be 2310 bytes in ROM and 242 bytes in RAM. These results show that dSENSE implementation is extremely compact and efficient.

Relative state change threshold

This section shows the energy performance of various sampling strategies described above in response to multiple data resolution requirements, specified in form of relative state change threshold. In this requirement, it is considered a state change when the difference between the new data value and the most recently sent value is greater than the pre-defined threshold. Performance is evaluated in terms of the energy consumption and miss ratio. For each experiment, the application specifies a *confidence level* that corresponds to the minimum miss ratio it is ready to suffer.

Figure 7 shows the energy consumption of the various scheduling strategies. The horizontal line in the figure denotes the minimum energy needed when there is no sensing or radio transmission. This energy corresponds to the absolute minimum energy required just to keep the sensor node powered on (with the CPU in sleep mode, and the radio and sensor

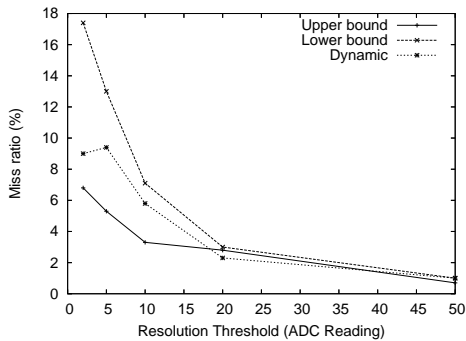


Figure 8: Miss ratio for a fixed confidence level = 90%.

powered off) without doing any useful work. This energy reflects the hardware limit imposed on any power management scheme.

We make several interesting observations from Figure 7. First, after employing the commonly used send-on-change strategy, the radio component is no longer the most energy-consuming component (consuming only about 0.7-28% of the total energy). Second, the CPU now consumes the largest amount of energy (about 44-61% of the total), most of which is spent in sleep mode. This result occurs due to two reasons: one is that the CPU spends most of its time in sleep mode, and the other is due to the new advances in Telos platform which have significantly reduced the idle CPU current drain to $55 \mu\text{A}$ ⁸. Third, sensing now consumes a significant percentage of the total power. For instance, with baseline sampling, sensing accounts for about 28% and 38% of the total energy for state change thresholds of 2 and 50 respectively. These results verify our contention that even low-power modality sensors can account for a large percentage of energy consumption.

We also see from Figure 7 that the energy consumption of the dynamic scheduling algorithm (166-254 mJ) lies between those of the upper bound (167-264 mJ) and the lower bound (166-228 mJ) strategies. Dynamic scheduling can reduce the baseline energy consumption by about 29% to 36%. In addition, the dynamic scheduling algorithm consumes almost the same amount of energy as the ideal sampling strategy. In some cases (threshold = 2 and 5), the dynamic strategy even consumes *less* energy than ideal sampling. Note that this is possible because dynamic sampling is allowed a 10% miss ratio, while ideal sampling has zero miss ratio.

Figure 8 shows the miss ratios associated with the three dSENSE sampling strategies. As can be seen from Figure 8, the dynamic (1-9.4%) and upper bound (0.7-6.8%) sampling strategies always stay within the tolerance level (10%) for all values of resolution threshold δ . The lower bound strategy yields lower miss ratios for high δ values (corresponding to coarser accuracy requirement), but go above the tolerance level to 17.4% and 13% respectively for low δ values (fine-

⁸The CPU, in idle mode, now draws *less* power than low-power modality sensors such as the temperature sensor.

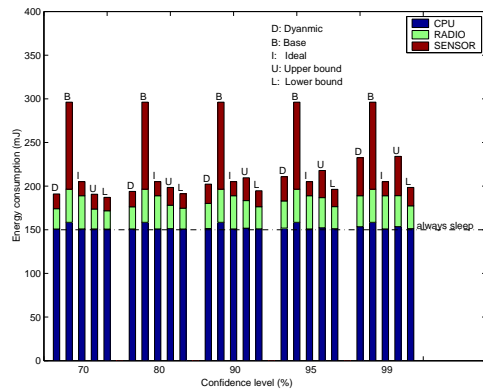


Figure 9: Energy consumption for a fixed resolution threshold = 5.

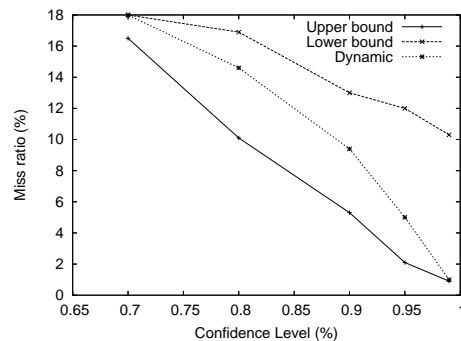


Figure 10: Miss ratio for a fixed threshold = 5.

grained accuracy) of $\delta = 2$ and 5.

Figures 9 and 10 show the performance of the various strategies for a fixed resolution threshold of 5, with varying miss ratio tolerance levels. Figure 9 shows upto an additional 18% energy saving when varying the tolerance level from 1% (confidence = 99%) to 30% (confidence = 70%). Figure 10 again verifies that the dynamic and upper bound strategies always guarantee the required tolerance level.

As described in Section 4.3.3, in the dynamic dSENSE scheduling strategy, the false positive threshold F_p is used as a dynamic tuning parameter to adjust sampling probabilities p_i following variations in the underlying processes. The values of the sampling probability p_i obtained in this manner are an indicator of how energy consumption is distributed temporally. Figure 11 shows the variation of the tuning parameter F_p and the sampling probability p_i over the time period of the test data trace. As can be seen from the figure, higher values of F_p , and thus higher sampling probabilities (corresponding to higher expected energy consumption) occur at time periods with larger data variation. This result indicates that the dynamic algorithm is able to adapt to changes in the underlying phenomenon and spends more energy during stages of flux. This figure also shows the ability of the dynamic scheduling algorithm to achieve adaptive sampling without any model

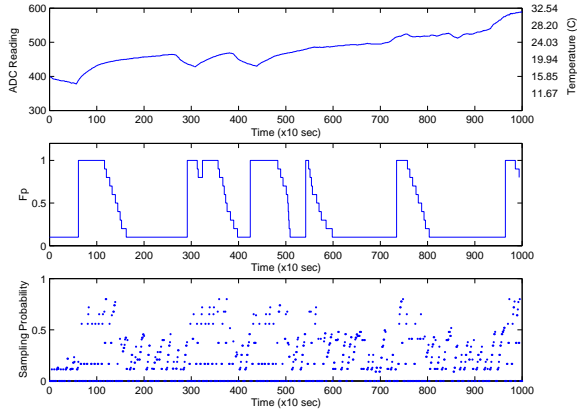


Figure 11: The variation of the tuning parameter F_p and the sampling probability p_i , relative threshold = 5 and confidence level = 90%.

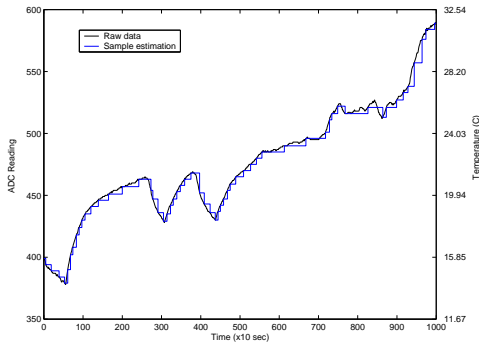


Figure 12: Sampling approximation of the original data series, relative resolution threshold = 5, confidence level = 90%.

updates.

Figure 12 shows how well the dynamic scheduling algorithm approximates the original data trace. This figure is generated for a resolution threshold of 5 with confidence level of 90%. As can be seen from the figure, the algorithm matches the actual data values closely, thus providing a good approximation of the data stream.

Overall, these figures demonstrate that the dynamic dSENSE scheduling algorithm can achieve significant energy savings in response to multiple resolution requirements, while providing statistical data quality guarantees. Dynamic scheduling balances the power consumption and quality guarantees between the upper and lower bounds as expected. The dynamic dSENSE scheduling algorithm also showed the capability to adapt to data variations without any external model updates, thus making judicious use of the available energy.

Absolute state change threshold

We now show the performance of dSENSE for absolute resolution threshold specification. With an absolute threshold, a state change is defined to occur only when the data value

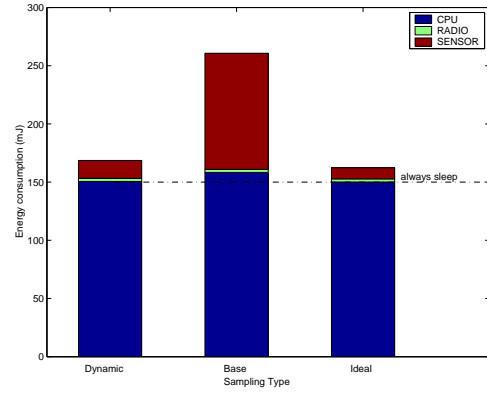


Figure 13: Energy consumption for absolute resolution threshold = 450 and confidence level = 99%.

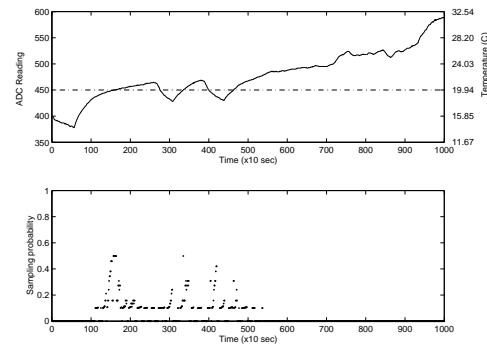


Figure 14: The variation in the sampling probability p_i , absolute threshold = 450 and confidence level = 99%.

crosses the threshold. The same data source and prediction models are used for these experiments. This kind of threshold specification is motivated by alarm-type applications, that typically generate alarms at threshold crossings, and have little interest in tracking the actual data. This experiment also demonstrates how dSENSE handles value-based data accuracy requirement as higher accuracy is required for values around the threshold line.

Figures 13, 14 and 15 present results generated from an experiment using an absolute threshold of 450 and confidence level of 99%. Figure 13 shows that dSENSE can achieve a 35% energy saving compared to baseline sampling scheme and consumes only about 4% more than the ideal sampling strategy. The miss ratio for the dSENSE scheduling is measured at 0.8%, satisfying the confidence level requirement. Figure 14 shows the sampling probability distribution over the test time period. This figure shows that higher sampling probabilities (more energy) are generated in the period when data values are close to the threshold line. This is primarily because the data model predicts higher state change probabilities during these time periods. Figure 15 shows that the dynamic dSENSE strategy closely tracks the threshold crossings of the original data trace.

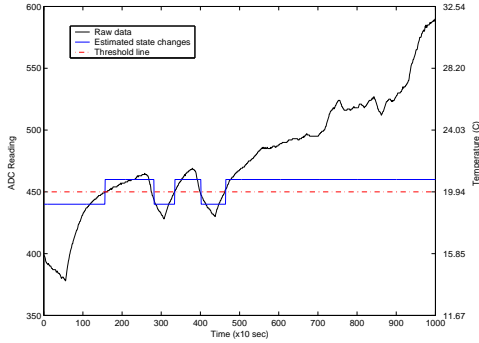


Figure 15: Sampling approximation of the original data series, absolute resolution threshold = 450 and confidence level = 99%.

6 Related Work

Dynamic Power Control: Dynamic voltage scaling and operating-mode switching techniques have been widely used for energy management in workstations [13], mobile systems [23, 12, 24, 27, 38], embedded systems [21, 8, 31], and server disks [16]. While most of these existing approaches have used the general idea of prediction-based dynamic power control, they do not consider application semantics and data quality requirements. On the other hand, one of the main contributions of our work is the explicit incorporation of application requirements in dynamic energy management.

Sensor Energy Management: Recently, there has been a great deal of research interest in energy management on sensor platforms. However, most research efforts have focused at the network level, trying to minimize the data communications between multiple sensor nodes [4, 25]. Node-level power management for sensor platforms has received little attention. Dynamic Power Management (DPM) [30] is one of the few efforts directly related to our work. DPM is similar to our approach in that it also attempts to control the operating modes of sensor node components in response to different workloads. However, unlike our approach, DPM is a purely OS-directed power management technique that is completely unaware of application semantics. It provides energy-accuracy tradeoff only at a generic level without any multi-level accuracy guarantees. On the other hand, our approach provides statistical data quality guarantees to applications. Finally, our technique is able to handle bursty data patterns [32] and time-of-day effects [36] that are not handled by DPM. The multiple sensing unit scheduling (MSUS) [7] takes a similar approach to DPM, and it also fails to incorporate application semantics and temporal locality inherent in measurement data streams. Another approach to node-level energy management employs admission control for multiple applications [4]. Our work is complementary to this approach as dSENSE is a lower-level building block suitable for man-

aging individual applications.

Stochastic Scheduling: Stochastic sensor scheduling has received some attention in the control community for target-tracking applications [15]. However, the focus of this work is on maximizing estimation accuracy through the scheduling of a set of sensors, while our main goal is to minimize energy consumption given an accuracy requirement. Moreover, the algorithm proposed in this work is essentially exhaustive search, which makes it infeasible for implementation on a low-power sensor platform.

Multi-resolution data quality: Access to data quality at multiple resolutions has been employed in several contexts. For instance, multimedia servers allow transmission of video content at multiple resolutions based on network conditions and client capabilities [19, 18]. Another approach [14] allows applications to degrade their data fidelity to conserve energy in server or workstation environments. These approaches are fundamentally different from our approach: while they enable application-driven degradation of data quality, the goal of our approach is to minimize energy consumption while *meeting* the application’s data quality guarantees. Application data quality requirement has also been employed for maintaining consistency of Web caches [35]. However, this effort is focused on conserving network bandwidth over the Internet, while our main goal is the conservation of energy in a sensor network.

7 Concluding Remarks

In this paper, we presented dSENSE: a data-driven approach for energy management in sensor platforms. dSENSE is a node-level power management approach that utilizes knowledge of the underlying data streams as well as application data quality requirements to conserve energy on a sensor node. dSENSE employs sense-on-change—a sampling strategy that aggressively conserves power by reducing sensing activity on the sensor node. A novel aspect of this strategy is that, unlike most existing energy management techniques, this strategy enables explicit control of the sensor along with the CPU and the radio. We presented an efficient statistical data stream model that is used by dSENSE to predict future sensor readings. These predictions are coupled with a stochastic scheduling algorithm to dynamically control the operating modes of the sensor node components. Using experimental results obtained on PowerTOSSIM with a real-world data trace, we showed that our approach reduces energy consumption by 29-36% while providing strong statistical guarantees on data quality.

As part of future work, we intend to extend our work in several directions. We would like to model the radio component as a pseudo-sensor, and extend our stochastic scheduling algorithm to efficiently schedule data transmissions on the network. We intend to explore correlations between multiple sensors on the same node or neighboring nodes to strengthen

our stochastic predictions and achieve higher energy savings. We would also like to investigate how our technique could be coupled with application-driven data quality degradation for network-wide energy conservation.

References

- [1] Y. Bar-Shalom and X. Li. *Estimation and Tracking: Principles, Techniques and Software*. YBS Publishing, 1998.
- [2] D. A. Berry and B. W. Lindgren. *Statistics: Theory and Methods (2nd Edition)*. Duxbury Press, 1996.
- [3] A. Boulis, S. Ganeriwal, , and M. B. Srivastava. Aggregation in sensor networks: a energy-accuracy trade-off. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*, May 2003.
- [4] A. Boulis and M. B. Srivastava. Node-level Energy Management for Sensor Networks in the Presence of Multiple Applications. In *Proceedings of the First IEEE Annual Conference on Pervasive Computing and Communications (PerCom'03)*, March 2003.
- [5] G. Box, G. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting and Control (3rd Edition)*. Prentice Hall, 1994.
- [6] J. M. Caicedo, J. Marulanda, P. Thomson, and S. J. Dyke. Monitoring of Bridges to Detect Changes in Structural Health. In *Proceedings of the 2001 American Control Conference*, June 2001.
- [7] H. Cam, R. Poornachandran, and H. Ahmad. Energy-efficient Task Scheduling for Wireless Sensor Nodes with Multiple Sensing Units. In *Proceedings of the International Performance Computing and Communications Conference (IPCCC'05)*, April 2005.
- [8] P. H. Chou, J. Liu, D. Li, and N. Bagherzadeh. IMPACCT: Methodology and Tools for Power Aware Embedded Systems. *Design Automation for Embedded Systems, Special Issue on Design Methodologies and Tools for Real-Time Embedded Systems*, 7(3):205–232, October 2002.
- [9] A. Deshpande, C. Guestrin, and S. Madden. Resource-aware Wireless Sensor-Actuator Networks. *Data Engineering Bulletin*, 28(1):40–47, March 2005.
- [10] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model Driven Data Acquisition in Sensor Networks. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, August 2004.
- [11] D. Estrin, A. Sayeed, and M. Srivastava. Wireless Sensor Networks, September 2002. Tutorial at MobiCom'02.
- [12] K. Flautner, S. Reinhardt, and T. Mudge. Automatic Performance-setting for Dynamic Voltage Scaling. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MobiCom'01)*, pages 260–271, July 2001.
- [13] M. Fleischmann. LongRun Power Management - Dynamic Power Management for Crusoe Processors (White Paper). Technical report, Transmeta Corporation, January 2001.
- [14] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 48–63, December 1999.
- [15] V. Gupta, T. Chung, B. Hassibi, and R. M. Murray. Sensor Scheduling Algorithms Requiring Limited Computation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2004.
- [16] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: dynamic speed control for power management in server class disks. In *Proceedings of the 30th annual international symposium on Computer architecture*, pages 169–181, June 2003.
- [17] N. Kurata, B.F. Spencer Jr., M. Ruiz-Sandoval, Y. Miyamoto, and Y. Sako. A study on building risk monitoring using wireless sensor network MICA mote. In *Proceedings of the First International Conference on Structural Health Monitoring and Intelligent Infrastructure*, November 2003.
- [18] W. Lee, J. Srivastava, and H. Cha. QoS-adaptive bandwidth scheduling in continuous media streaming. *Information and Software Technology*, 44(9):551–563, 2002.
- [19] W. Lee, J. Srivastava, and B. Sabata. QoS-Aware Admission Control and Dynamic Resource Provisioning Framework in Ubiquitous Multimedia Computing Environments. *Journal of Supercomputing*, 32(1):25–50, 2005.
- [20] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, pages 126–137, November 2003.
- [21] D. Li, P. H. Chou, and N. Bagherzadeh. Mode Selection and Mode-Dependency Modeling for Power-Aware Embedded Systems. In *Proceedings of the 7th Asia and South Pacific Design Automation Conference*, pages 697–704, January 2002.
- [22] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao. Distributed Group Management in Sensor Networks: Algorithms and Applications to Localization and Tracking. *Telecommunication Systems*, 26(2):235–251, 2004.
- [23] X. Liu, P. Shenoy, and W. Gong. A Time Series-based Approach for Power Management in Mobile Processors and Disks. In *Proceedings of the 14th ACM Workshop on Network and Operating System Support for Audio and Video (NOSSDAV'04)*, pages 74–81, June 2004.
- [24] J. Lorch and A. Smith. Operating System Modifications for task-based speed and voltage scheduling. In *Proceedings of the First ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys'03)*, pages 215–229, May 2003.
- [25] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.
- [26] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, September 2002.
- [27] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling on the lpARM microprocessor system. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED'00)*, July 2000.
- [28] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN/SPOTS'05)*, April 2005.
- [29] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, November 2004.
- [30] A. Sinha and A. Chandrakasan. Dynamic Power management in Wireless Sensor Networks. *IEEE Design and Test*, 18(2):62–74, March 2001.
- [31] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive System shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on VLSI Systems (TVLSI)*, 4(1):42–55, March 1996.
- [32] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load Shedding in a Data Stream Manager. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, September 2003.

- [33] Telos/Tmote Datasheet, December 2004. <http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf>.
- [34] SHT1x/SHT7x Humidity and Temperature Sensor Datasheet, July 2004. http://www.sensirion.com/en/pdf/Datasheet_SHT1x_SHT7x.pdf.
- [35] B. Urgaonkar, A. Ninan, M. Raunak, P. Shenoy, and K. Ramamritham. Maintaining Mutual Consistency for Cached Web Objects. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pages 371–380, April 2001.
- [36] R. Vilalta, C. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 41(3):461–474, 2002.
- [37] xBow Technology Inc. Heading Sensor Datasheet, July 2004. http://www.xbow.com/Products/Product_pdf_files/Mag_pdf/6020-0030-01_A_C%HS110.pdf.
- [38] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, October 2003.