Stochastically Consistent Caching and Dynamic Duty Cycling for Erratic Sensor Sources *

Shanzhong Zhu, Wei Wang and Chinya V. Ravishankar

Department of Computer Science and Engineering
University of California, Riverside, CA 92521
{szhu, wangw, ravi}@cs.ucr.edu

Abstract. We present a novel dynamic duty cycling scheme to maintain stochastic consistency for caches in sensor networks. To reduce transmissions, base stations often maintain caches for erratically changing sensor sources. Stochastic consistency guarantees the cache-source deviation is within a pre-specified bound with a certain confidence level. We model the erratic sources as Brownian motions, and adaptively *predict* the next cache update time based on the model. By piggybacking the next update time in each regular data packet, we can dynamically adjust the relaying nodes' duty cycles so that they are awake before the next update message arrives, and are sleeping otherwise. Through simulations, we show that our approach can achieve very high source-cache fidelity with low power consumption on many real-life sensor data. On average, our approach consumes 4-5 times less power than GAF [1], and achieves 50% longer network lifetime.

1 Introduction

Power-efficient sensor data acquisition has become important as large-scale sensor networks become increasingly practical. A framework for data acquisition in sensor networks was introduced in [2], and various power-efficient techniques have been proposed in [3–6] for sensor data collection in multi-hop wireless environments. Typically, users present their queries to a base station (BS), which collects data appropriately and generates responses.

In this paper, we show how to combine two strategies for reducing sensors' power consumption: *base station caching* and *dynamic duty cycling*. These ideas have been applied independently, but little work exists on strategies for combining them effectively in sensor networks.

1.1 Caching to Reduce Data Transmissions

Caching is commonly used to reduce data transmissions, which dominate power consumption in sensor networks. Several models for caching have been explored in the literature. In the first such model, exemplified by [2, 3], queries explicitly

* This work was supported by a grant from Tata Consultancy Services, Inc.

specify the sampling rate for sensor data. Queries arriving at intermediate times are handled using cached data at the BS. Source and relaying transmissions are scheduled to occur as required by the known sampling times. The sensors can also be put to sleep in between, saving even more power. Although simple, this model cannot offer any guarantees on the precision of the cached data, especially when the underlying source data change rapidly and unpredictably.

Another approach is represented by [7], in which sources continuously stream updates to a central server which handles a large number of aggregate queries registered by users. The server caches a copy of each source object. Sampling times are not presepecified, but each aggregate query is associated with a *precision requirement*, indicating the maximum error the user will tolerate. A *filter*, or error bound, is installed on each source, and only values exceeding the filter bounds will be sent to the server. Filter bounds are adaptively set to minimize transmission costs, while ensuring that the precision requirements are met.

Unfortunately, for all the reasons discussed in [8], this approach wastes power if directly used in sensor networks. In addition, since sampling intervals are not fixed, updates arrive unpredictably, so all relaying nodes must always have their radios on. It is well-known that sensors consume significant amount of power in the listening mode [9, 10]. For example, in MICA2, the power consumed in listensing/receiving mode (7mW) is very close to the power consumed in transmitting mode (10mW); while in MICAz, the power consumed in listening/receiving mode (19.7mW) is even higher than in transmitting mode (17mW) [11]. Thus, to conserve power in sensor networks, we must put sensors into sleep as often as possible, while still guaranteeing cache consistency requirements.

Erratic Data Sources and Stochastic Consistency Power optimization is particularly challenging in sensor networks that monitor *erratic* data sources [12]. Erratic data are numerical data that change frequently and unpredictably, such as temperature, pressure, and humidity. It is hard to predict erratic data behavior, making it hard to ensure cache-source consistency.

Strict cache-source consistency is unrealistic in sensor networks, since power is limited and wireless channels are volatile. However, users are often willing to tolerate some error, as long as it remains within pre-specified bounds. *Stochastic consistency*, first introduced in [12], captures this idea, and guarantees the cache-source deviation is within an error bound with a certain confidence level.

For example, in a sensor network to monitor temperatures, a user may be satisfied with a value within $2°F$ of the true value, with confidence 90%. Therefore, the source sensor needs to update the cached copy at the BS only when the cache-source deviation is no longer within $2°F$ with confidence 90%. We address the issue of maintaining stochastic consistency with minimum power consumption in sensor networks.

1.2 Dynamic Duty Cycling to Reduce Power Consumption

Our goal is to let each sensor node dynamically adjust its duty cycle so that it is in sleep most of the time, but has its radio on whenever an update must

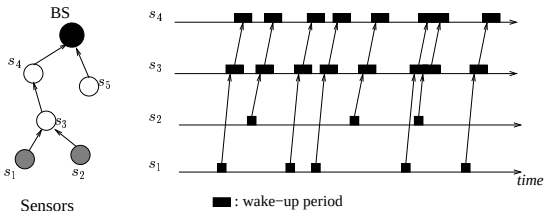


Fig. 1. Duty-cycling sensor nodes

be relayed to the BS. Lowering duty cycles is known to be an effective way to extend the lifetime of the network [1, 13–15]. Our approach is illustrated in Figure 1, in which s_1 and s_2 are two sources, whose updates are delivered to the BS through s_3 and s_4. Both s_3 and s_4 turn on their radios only when an update packet is expected to arrive. The key challenge is how to let each relaying node estimate the arrival time of the next update, so that it can adjust its duty cycles accordingly.

Approaches such as GAF [1] and SPAN [15] try to maintain a routing backbone to ensure connectivity of the wireless ad hoc network, while allowing as many nodes as possible to sleep. At least one routing node is guaranteed to be within the transmission range of any node. GAF uses geographic location information (from GPS) to determine node equivalance for routing. SPAN uses a distributed randomized algorithm to maintain the backbone. Unfortunately, neither scheme exploits source data characteristics which may not require a connected backbone at all times.

In a sample network with 100 nodes uniformly distributed in a $1500m \times 300m$ region mentioned in [1], the resulting GAF routing backbone consists of 45 nodes. Hence, GAF always has 45 nodes listening, whether or not a message is active. In contrast, our approach captures source data characteristics, so that each source can *predict* update times, letting relaying nodes safely sleep till that time. Our approach will save more power especially under light source rates, since the relaying nodes are allowed to sleep more often than in GAF or SPAN (see Fig. 6).

We predict update times using the *Brownian motion* model [16], a stochastic model widely used to characterize randomly fluctuating data. Based on the user-provided consistency requirement and current data characteristics, the model adaptively determines the due time of the next update so that errors are bounded. The next update time is piggybacked on the current update message and delivered to the relaying nodes en route to the BS, which can safely turn off their radios and sleep before the arrival of the next update.

In our approach, each source delivers updates only at the times *predicted* by the Brownian motion model. In contrast, in approaches such as [7], updates are delivered at the times the source *detects* that the actual value has exceeded the error bound. The correctness of our approach is determined solely by how well our model matches future data behaviour under the stochastic consistency model. As shown by our extensive experiments (see Section 6), our method achieves high consistency (or *fidelity*) on various real-life sensor data, while saving a significant amount of power.

1.3 Our Contributions

We make several contributions in this paper. First, we experimentally verify that sensor data, such as temperature, humidity and ocean salinity, can be modeled as Brownian motions. This model has been successfully used in earlier work to model many other real-world erratic data sources [12, 17]. We confirm that model parameters, such as the *drift* and *diffusion* parameters, can capture the short-term linear trend and variance, respectively, with high confidence.

Next, we propose a dynamic duty cycling scheme based on the Brownian motion model, to allow nodes to turn off their radios frequently, while guaranteeing consistency requirements. A node will turn on its radio only when an update message is expected to arrive. In general, duty cycles are driven by the consistency requirements and source data characteristics.

Finally, we verify the correctness and efficiency of our approach with extensive simulations, which show that we can achieve high fidelity using far less power than GAF.

The rest of this paper is organized as follows: We review some related work in Section 2. Our system architecture and routing scheme are described in Section 3. In Section 4, we briefly introduce the Brownian motion model and perform experiments to verify its applicability on many sensor generated data. Our dynamic duty cycling scheme is presented in Section 5. The experimental results are presented in Section 6. Section 7 concludes our work.

2 Related Work

Various consistency models have been proposed to accommodate different requirements for cache freshness. For example, *quasi-caching* [18] allows the cached value to deviate from the source value in a controlled way (say, delay-bounded or error-bounded). *Probabilistic consistency* [19] guarantees that cached values are temporally consistent with the true value with a probability p. The concept of *stochastic consistency* was introduced in [12], and aims to provide an error-bounded cached copy with a given confidence. This model has been successfully used in pull-based replicated systems for erratic data streams [12]. We use this model in sensor environment.

2.1 Duty Cycling

Dynamic duty cycling is another technique widely used to achieve power efficiency in sensor networks. In GAF [1] and SPAN [15], nodes adaptively switch between sleeping and listening, while guaranteeing the existence of a capacity-preserving backbone routing network at any time. In GAF, each node used geographic location information (provided by GPS) to associate itself with a *virtual grid*. All the nodes in a virtual grid are equivalent for routing. *SPAN* is a distributed randomized algorithm, in which nodes can locally determine whether to sleep or stay awake in the backbone routing network, without knowledge of their

geographic locations. Periodically, the set of routing nodes is changed to ensure even power dissipation. LEACH [20] aims to provide a cluster-based routing hierarchy where all sensor nodes are divided into clusters. A cluster head is elected to route data on behalf of the other nodes in each cluster. In our approach, duty cycles are driven by the source update rates, which are in turn governed by consistency requirements. In Section 6, we show that our approach lets sensors sleep more often, thus saving more power than GAF.

A periodic duty cycling scheme was introduced in *S-MAC* [21], in which nodes periodically switch between the listening and sleeping modes to conserve power. Neighbouring nodes exchange their listen/sleep schedules to synchronize their duty cycles. To deliver a packet, a sending node waits till the next hop node wakes up. However, significant latency will still be introduced since delays are accumulated along multiple-hop paths to the BS. A similar scheme was proposed in *STEM* [14], which assumes that two separate radios, a *wakeup* radio and a *data* radio, are available to each sensor node. To send a packet, the wakeup radio of the sending node polls the receiving node until it wakes up, and turns on its data radio. Again, data packets will experience significant delays because such delays at each hop will accumulate over the route. Both schemes are clearly not suitable in our situation, where updates must reach the BS as soon as possible, to ensure cache freshness. In TAG [3], the nodes along the aggregation tree are periodically synchronized with each other to relay and aggregate new sensor data. Since the source sampling rate is specified in the query, their synchronization scheme is much simpler than ours.

The success of our scheme relies on modeling the underlying data as Brownian motions. Applying probabilistic models to sensor data has been shown to be effective in conserving power while providing quality results [17,22,23]. Section 4 confirms earlier work that has shown that Brownian motions can model erratic data streams with high confidence [12, 17].

2.2 Stochastic Consistency

Stochastic consistency [12] guarantees that the deviation between a cached value and the true value is within a pre-specified error bound ϵ with a confidence at least p. Let $v_i(t)$ and $c_i(t)$ be the source and the cached values, respectively, of object o_i at time t. The cache is stochastically consistent with source at t if

$$\Pr[\,|v_i(t) - c_i(t)| \leq \epsilon\,] \geq p. \tag{1}$$

We must update the cached copies frequently enough to maintain stochastic consistency. On the other hand, to save cache/source communications, we must send the updates right before the confidence that cache-source deviation is within ϵ starts to drop below p. In Section 4, we discuss how to determine update times under this model.

3 Our System

Sensor networks typically consist of a BS with ample resources and a set of resource-limited sensor nodes communicating with the BS over multi-hop wire-

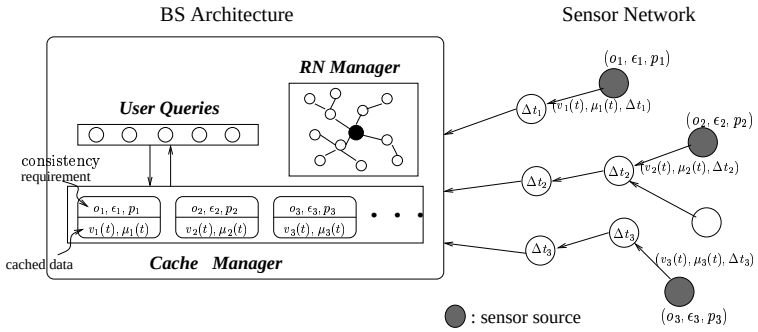


Fig. 2. The BS Architecture

less channels (see Figure 2). The BS serves as the destination for sensor data, and as the interface to user queries. It maintains caches to reduce communications and provide prompt responses. Our focus is on how to maintain cache-source consistency in a power-efficient way. Our caching system can support a broad spectrum of queries, ranging from monitoring single sensor's readings to aggregate queries as in [3, 7].

Fig. 2 also shows an architectural schematic for the BS. The *cache manager* manages all cached objects. The object o_i represents a data source sensed by sensor s_i. Each object o_i is associated with a *consistency requirement* (ϵ_i, p_i) determined by user requirements. (Converting user requirements to object consistency requirements is an orthogonal concern we do not address. An example can be found in [7].)

An object's consistency requirement is also available at the corresponding source sensor, which determines when a cache update must be sampled and delivered. A cache update takes the form $(v_i(t), \mu_i(t))$, where $v_i(t)$ is the sampled value at time t, and $\mu_i(t)$ is the current *drift* parameter estimated at the source. The drift parameter is a Brownian motion parameter and represents the current linear trend of o_i. It helps to provide a more accurate cache value at the BS (see Section 5.1). The next update time Δt_u is adaptively evaluated under the stochastic consistency model, and included with each update so that each relaying node en route can sleep for time Δt_u. The BS responds to queries by retrieving the current values from the cache, calculating query results, and returning them to users.

The *RN manager* maintains a view of the *routing network (RN)*, which is a collection of routes through which sensors may reach the BS. Based on this information, the BS can determine a power-efficient route for each newly cached object source (see Section 3.1).

3.1 Routing

In principle, our dynamic duty cycling scheme is independent of the routing protocol, as long as routes are *persistent*, that is, the route for each source remains unchanged for a certain time. This property allows nodes on each route to obtain the wake-up time for the next update from each update message (see

time interval	**temp** traces (*Depth*)		**salinity** traces (*Long./Lat.*)		**humd** traces (*Long./Lat.*)		**slp** traces (*Long./Lat.*)	
	36M	*47M*	*5N/180W*	*2N/180W*	*0N/155W*	*2N/140W*	*2N/110W*	*2S/95W*
10 min	75.21%	72.96%	77.44%	76.65%	80.14%	81.13%	76.54%	80.12%
15 min	72.38%	75.90%	76.26%	76.60%	79.47%	79.79%	75.88%	79.17%
20 min	73.45%	73.55%	76.10%	75.15%	76.45%	77.31%	75.45%	76.92%
30 min	71.47%	66.13%	75.00%	74.84%	72.60%	75.14%	74.38%	73.04%

Table 1. Average p-values from the *W-S* test for various sensor traces and time intervals, confidence interval: 95%, all traces obtained from *TAO* Project.

Section 5.3). Many ad hoc routing protocols for sensor networks generate persistent routes [3, 4, 24, 25]. In our work, we use a *energy-aware* routing scheme similar to [24] and [26], to avoid bottleneck nodes that would otherwise dissipate their power much faster than the others. First, we build a *routing network (RN)* through which sensor nodes can communicate with the BS. Typically, the RN includes good-quality wireless links to ensure reliable transmissions. Besides, as power is our major concern, it is desirable that each route in the RN be the shortest path from the source to the BS.

A common approach to building a RN is to assign a *level* number to each sensor node depending on its distance to the BS [3, 6, 26]. The BS is at level 0; Those nodes 1-hop away from the BS are at level 1, and so on. Initially, the BS broadcasts a query message containing its ID and level number. Upon receiving this message from its neighbours, each node determines its level and parents, and rebroadcasts the query message with its own ID and level number. After the query messages have flooded the entire network, a RN is formed where each node has one or more parents through which it can send packets towards the BS. Any path is the shortest one in the resulting RN. A more detailed description of constructing the RN can be found in [26]. The above algorithm must be run periodically to accommodate topology changes. To allow the BS maintain a view of the RN, each node must send a message containing its level number and parents to the BS.

After the RN is set up, the *RN manager* is responsible for determining a route for each newly cached object source. To balance power consumption, we choose routes with the maximum remaining power. A route's remaining power is defined as the minimum remaining power on its en-route nodes. Each node periodically determines its remaining power level and piggybacks the value on regular update messages destined for the BS. The RN manager periodically re-evaluates the remaining power on each route and chooses the one with the maximum power left.

4 Modeling Sensor Data

Sensor data are often numerical values, and change continuously. Modeling their behavior is central to our dynamic duty cycling scheme.

4.1 Standard and Drifting Brownian Motion Models

The Brownian motion model [27] is a continuous-time stochastic process widely used to characterize highly fluctuating data, and has been successfully used to model stock prices [28] and other erratic data sources [12] such as temperature and computer system loads. A *Standard Brownian motion (SBM)* $W(t)$ satisfies: 1) $W(0) = 0$; 2) $W(t) - W(s)$ is normally distributed with mean 0 and variance $t - s$ $(t \geq s)$; 3) $W(t) - W(s)$ is independent of $W(v) - W(u)$ if (s, t) and (u, v) are non-overlapping time intervals.

A variant of the *SBM* is the *drifting Brownian motion (DBM)* $S(t)$, which includes a secular drift in the expectation of the process. The increment of $S(t)$ is modeled as:

$$\Delta S(t) \; = \; \mu(t)\Delta t + \sigma(t)\Delta W(t), \tag{2}$$

where $\mu(t)$ and $\sigma(t)$ are the *drift* and *diffusion* parameters for $S(t)$, respectively. $W(t)$ is a SBM process. The drift parameter models a secular upward or downward trend in the random data, while the diffusion parameter models the randomness of the data. Hence, the DBM is a combination of a predictable linear trend and a Brownian motion process. It is easy to see that the increment $\Delta S(t)$ also follows a Normal distribution: $\Delta S(t) \sim N(\mu(t)\Delta t, \sigma^2(t)\Delta t)$. In Section 4.2, we show that the DBM model is applicable to many real-life sensor data.

4.2 Verifying DBM on Sensor Data

In [12], we have already shown that real-life data sources such as stock traces, ocean temperatures, and system load data can be successfully modeled as DBM. In this work, we further verify that this model is appropriate for a wider variety of sensor genearted data, using the same methodology as in [12]. Since data increments are normally distributed in the DBM model, we will perform *normality tests* [29] on increments of sensed data. Various methods for normality testing have been proposed, and, as explained in [12], the *Wilk-Shapiro (W-S)* test [29] is the most appropriate in our case.

Our sensor data traces were taken from the *TAO* project [30] at the Pacific Marine Environmental Laboratory (PMEL). We tested four categories of data generated by ocean sensors: ocean temperature (*temp*), relative humidity (*humd*), salinity (*salt*), and sea level pressure (*slp*). Each category included data traces generated at different geographical locations (*Longitude/Latitude*) or ocean depths (*Depth*). Each data trace comprised about a year's data sampled at every 1 minute. Table 1 shows the results of the *W-S* test on these traces.

Each value in Table 1 represents the average p-value [16] evaluated on increment samples over a certain time interval. The *W-S* test calculates a test statistic for each data series. The p-value measures the probability that the test statistic will take on a value that is at least as extreme as the calculated value when the samples are normal. The p-value measures the probability that the tested data are drawn from a normal distribution. The significance level (α) of our test is set to 0.05. The larger the p-value, the stronger the confidence with which we may accept the samples as normal [29]. As shown in Table 1, The p-values for

our data are far higher than α, indicating that we can believe the increments are normal with high confidence. For longer intervals, the p-value drops somewhat, suggesting the model may evolve in the long run.

The *W-S* test strongly supports our hypothesis that the sensor data are DBMs. On the other hand, since the model may evolve along with time, it is important to periodically estimate the model parameters $\mu(t)$ and $\sigma(t)$ to accurately characterize the underlying data. We will discuss how to estimate the two parameters in Section 5.2.

5 Dynamic Duty Cycling

We achieve power efficiency by operating sensors in low duty cycles, while guaranteeing that cache updates will arrive on time at the BS. The update intervals change dynamically due to the erratic nature of data sources. An approach to maintaining cache consistency similar to that of [7] would be for each source to constantly sample the underlying data. If it finds the sampled value deviates from the last update by more than ϵ, it forwards the value as a cache update. This approach forces relaying nodes to be awake all the time since update times are unpredictable. We need a more intelligent approach that can *predict* the due time of the next update, and let the relaying nodes sleep safely until that time.

Our approach models erratic data sources as Brownian motions, estimates the times when the cache-source deviation is expected to exceed ϵ, and schedules the next cache update at that time. When a source is ready to deliver an update, it also determines the time interval Δt_u until the next update, based on the DBM model. The source then sends the update along with Δt_u, so that each relaying node can obtain Δt_u. Since it knows that the next update from the source will arrive after time Δt_u, it can safely turn off its radio and sleep for time Δt_u. Our approach allows the relaying nodes to dynamically synchronize with each other and form a connected path whenever an update is ready to be sent.

We discuss how to adaptively derive Δt_u in Section 5.1. The drift and diffusion parameters must be estimated regularly from the underlying data. The issue of parameter estimation is discussed in Section 5.2. Our scheme to perform dynamic duty cycling is presented in Section 5.3. In Section 5.4, we analyze the power cost for our scheme.

5.1 Determining Cache Update Times

The drift and diffusion parameters characterize the current linear trend and randomness, respectively, of the sensor data. Each sensor source can use these parameters to adaptively determine Δt_u, the time till the next update. Let t_0 be the last time an update was delivered for object o_i, and $v_i(t)$ and $c_i(t)$ be the true and cached values of o_i at time t, respectively. Stochastic consistency requires the next update to be delivered before our confidence that the cache-source deviation is within ϵ drops below p. We must solve Δt_u from the following equation:

$$\Pr[\,|v_i(t_0 + \Delta t_u) - c_i(t_0 + \Delta t_u)| \leq \epsilon\,] = p. \tag{3}$$

Based on the DBM, we have $v_i(t_0 + \Delta t) = v_i(t_0) + \mu_i(t_0)\Delta t + \sigma_i(t_0)\Delta W(t)$, if $\mu_i(t_0)$ and $\sigma_i(t_0)$ are the drift and diffusion parameters estimated at time t_0, and $W(t)$ is the SBM (see Equation 2). Clearly, the expected value of $v_i(t_0 + \Delta t)$ is $v_i(t_0) + \mu_i(t_0)\Delta t$, which is also the best estimate the cache can make at time $t_0+\Delta t$, given that the last cache update is $(v_i(t_0), \mu_i(t_0))$. Therefore, $c_i(t_0+\Delta t) = v_i(t_0)+\mu_i(t_0)\Delta t$. We can easily derive that the cache-source deviation is normally distributed:

$$v_i(t_0 + \Delta t) - c_i(t_0 + \Delta t) \sim N(0, \sigma_i^2(t_0)\Delta t). \tag{4}$$

From Equations 3 and 4, we can obtain:

$$\Delta t_u = \frac{1}{2}\left(\frac{\epsilon}{\sigma_i(t_0)\,\mathrm{erf}^{-1}(p)}\right)^2, \tag{5}$$

where $\mathrm{erf}^{-1}(p)$ is the well-known *inverse error function* [31]. The detailed derivation of Equation 5 can be found in the Appendix of [32].

Δt_u must be recomputed on-line at sensor sources. Since $\mathrm{erf}^{-1}(p)$ can be precomputed and stored for the required p, computing Δt_u requires only some simple arithmetic operations, and is easily affordable for sensors.

5.2 Estimating Model Parameters

Obtaining accurate estimates for $\mu_i(t)$ and $\sigma_i(t)$ is critical to the success of our approach. According to the DBM model, increments follow the normal distribution $N(\mu_i(t)\Delta t, \sigma_i^2(t)\Delta t)$. Assuming both $\mu_i(t)$ and σ_i remain relatively constant over small time intervals, we may estimate $\mu_i(t)$ and $\sigma_i(t)$ by estimating the *mean* and *variance* of increment samples over a small time interval. The simplest unbiased estimators [16] of the mean and variance of a sample $\{x_1, \ldots, x_n\}$ are $\hat{x} = (\sum x_i)/n$ and $\hat{\sigma}^2 = \sum(x_i - \hat{x})^2/(n-1)$.

Let $\hat{\mu_i}(t)$ and $\hat{\sigma_i}(t)$ be the estimated values of $\mu_i(t)$ and $\sigma_i(t)$, respectively. Our estimation scheme works as follows: Let t_1 be the time of the next update. Starting at time $t_1 - \delta$, we sample the underlying data every h time units, where $h < \delta$. Thus, at time t_1, we collect n data samples: $v_i[1], v_i[2], \cdots, v_i[n]$, where $n = \delta/h + 1$. The obtained $n-1$ increments $v_i[j+1] - v_i[j]$ $(1 \le j < n)$ are independent normal samples, and since δ is small, these samples are identically distributed. Thus, we calculate $\hat{\mu_i}(t_1)$ as follows:

$$\hat{\mu_i}(t_1) = \frac{(v_i[n] - v_i[1])}{\delta}. \tag{6}$$

We can also estimate obtain $\hat{\sigma_i}(t_1)$ from:

$$\hat{\sigma_i}^2(t_1) = \frac{1}{(n-2)h}\sum_{j=1}^{n-1}(v_i[j+1] - v_i[j] - \hat{\mu_i}(t_1))^2. \tag{7}$$

In typical sensors, such as those for light, temperature, or magnetic fields, the sampling time is on the order of $0.1ms$ [2].

A relatively small δ ensures accurate estimation of $\mu_i(t)$ and $\sigma_i(t)$, since these parameters remain constant during small intervals with high probability. On the other hand, a smaller h leads to more samples but may increase power consumption. A larger h saves power but may result in inaccurate estimates

due to too few samples. Thus, we must choose both δ and h carefully to balance estimation accuracy and power consumption. In our experiments, we set $\delta = 10h$. Our results show that the obtained sample size is appropriate for our purpose.

5.3 Our Scheme

Each node can be in the *active* or *idle* state, depending on whether or not it is actively delivering/relaying update packets. Initially, all nodes are idle. During the RN setup phase, each node is assigned a *wakeup interval* t_w. It wakes up every t_w time units to check for pending caching requests from the BS. Upon receiving a request $R(s_i)$, an idle node switches to the active state, since it knows it will participate in relaying updates for source s_i. The choice of t_w must balance power consumption against response time (how long the BS must wait until receiving the first cache update). Larger t_w values let nodes sleep longer, but increase response times. We chose a moderate value for t_w in our experiments.

When the BS must query sensor s_i, it first consults the RN manager to find a route to s_i (see Section 3.1). It then sends the request $R(s_i)$ and the consistency requirement to s_i. If a node s_{j1} along the route finds the next node s_{j2} to be still asleep, s_{j1} will poll s_{j2} until it wakes up. Node s_{j2} records the sending node s_{j1}, so that it knows where to deliver s_i's updates. Each en-route node remains in listening mode until it receives the *first* update from s_i, and lets the Δt_u supplied by s_i drive its duty cycles after that point.

At the source s_i, Δt_u is evaluated on a regular basis according to Equation 5. A series of samples must be collected for parameter estimation before delivering the update message. Each update message contains the most recent sample $v(t_{update})$, the drift parameter $\mu(t_{update})$, and the next Δt_u.

Each node on the return route to the BS obtains Δt_u from the message containing the sensor update, and schedules to wake up at $t_{next} = t_{curr} + \Delta t_u - t_e$, where t_{curr} is the current system time, and t_e is a small time offset to accommodate variations in wireless transmission delays. In our simulations, we set $t_e = 10ms$ with moderate traffic in the network. Since a node may relay messages for several sources, it maintains a list to hold the future wakeup times. After relaying an update message, the node can safely turn off its radio and sleep until the next time entry in the list is due. Since the BS may make data requests while the node is asleep, the node must check for such requests to avoid poor response times. If the time interval until the next wakeup time entry is larger than t_w, the node must wake up at t_w to perform this check. More details on our scheme can be found in [32].

Every time a user requests o_i's value from the cache at the BS, the cache manager returns $v_i(t_l) + \mu_i(t_l)(t_{curr} - t_l)$, where t_{curr} is the current time, and t_l is the last time an update message was received.

5.4 Analysis of Power Consumption

Power is charged for communication, computation, and data sampling on sensor nodes. We ignore the power consumed by computation in our analysis since it

is orders of magnitude lower than that by communication [2]. A sensor's radio may be in one of the following modes: *transmitting (T)*, *receiving (R)*, *idle(I)*, or *sleeping (S)*. In the idle mode, it listens to the wireless channel, waiting for incoming packets. In the sleeping mode, it turns off its radio, so the consumed power is negligible compared with other modes. Let the power consumed in transmitting, receiving, and idle mode be P_T, P_R, and P_I, respectively, and let P_S be the power consumed by sampling the underlying data. The total consumed power is simply $P = P_T + P_R + P_I + P_S$.

Our approach enables each node to remain in the sleeping mode most of the time and wake up only when an update is expected to arrive. Ideally, the idle time on each node is zero. Thus, $P = P_T + P_R + P_S$. Let P_{t_0} and P_{r_0} be the power consumed for transmitting and receiving one bit of data, respectively, and let P_{s_0} be the power consumed by sampling one piece of data from the environment (we assume each sensor has only one sensing module). Node s_i's power consumption rate at time t is

$$P_i(t) = \left(\sum_{s_j \in R_i} f_j(t) \right) M(P_{t_0} + P_{r_0}) + I(s_i) f_i(t) \left(M P_{t_0} + (\frac{\delta}{h} + 1) P_{s_0} \right), \quad (8)$$

where M is the size of the update message, R_i is the set of sources whose updates are relayed by s_i, $f_j(t)$ is the rate of updates generated by source s_j at time t, and $I(s_i)$ is an *indicator* function which is 1 if s_i is also a source, and 0 otherwise. Equation 8 suggests that a relaying node's power consumption is proportional to the aggregate amount of update traffic it relays, and a source's power consumption is also governed its updating frequency.

Combining Equations 5 and 8, we can further obtain:

$$P_i(t) = \left(\sum_{s_j \in R_i} (\beta_j \sigma_j^2(t)) \right) M(P_{t_0} + P_{r_0}) + I(s_i) \beta_i \sigma_i^2(t) \left(M P_{t_0} + (\frac{\delta}{h} + 1) P_{s_0} \right), \quad (9)$$

where $\beta_j = 2(\frac{\text{erf}^{-1}(p_j)}{\epsilon_j})^2$. In Equation 9, a relaying node's power consumption $P_i(t)$ is a function of its upstream sources' consistency requirements (ϵ_j and p_j) and their current data variance ($\sigma_j^2(t)$). Not surprisingly, the higher the variance, the greater the number of updates to be delivered to maintain a certain level of consistency, and the greater the power consumption. On the other hand, a more stringent consistency requirement (small ϵ_j and large p_j) also results in higher power consumption, which is illustrated in Fig. 6.

If s_i is a source, its power consumption is also governed by its own consistency requirement (ϵ_i and p_i) and data variance ($\sigma_i^2(t)$). Every time an update is required, additional δ/h samples must be collected for parameter estimation (see Equation 9). Since the power required for sampling (P_{s_0}) is very low for many typical sensors such as light, temperature, and accelerometer [2] (on the order of 0.1 μJ), such sampling overhead is affordable.

6 Experiments

We conducted extensive experiments to demonstrate the correctness and efficiency of our dynamic duty cycling scheme using the ns-2 simulation package [33].

6.1 Simulation Setup

We uniformly deployed 100 sensor nodes in a $1500 \times 1500\, m^2$ region with the BS at the center. We chose UDP as the transport layer protocol and 802.11 [34] as the MAC layer protocol. The ns-2 simulator currently supports three propagation models, among which the *shadowing* model [35] is the most realistic and widely-used. This model has two parts: a path loss model, and a statistical model for the variation of reception at certain distances. We set the value of the path loss exponent as 2.0, and the value of the shadowing deviation as 4.0, representing a typical outdoor environment. We set the radio communication range to $250m$ and chose 0.95 as the rate of correct reception.

A subset of sensor nodes were chosen as sources. We used various categories of real-life sensor data as the source traces such as the ocean temperature traces (*temp*), the relative humidity traces (*humd*), and the ocean salinity traces (*salt*), all obtained from the *TAO* project [30] (see Section 4.2). Each source was associated with one data trace every time. In each experiment, we used 5 different traces from each category and demonstrated the average results. We purposely selected sources from the most distant nodes from the BS, since it is more challenging to maintain cache consistency for the distant sources. We chose a fixed payload size of 16 bytes for each update message including the data value, the drift parameter, and the Δt_u value.

To measure power, we adopted the power parameters from the *Chipcon* CC1000 RF transceiver [10], which is used as the radio module in MICA2 and MICA2DOT sensor models. When operated at $433MHz$, its receiving power is $22.2mW$, and the transmitting power is $31.2mW$, with the output power of $0dBm$. Each node was set to the same power level initially, and we measured the remaining power after the simulation ran for some time.

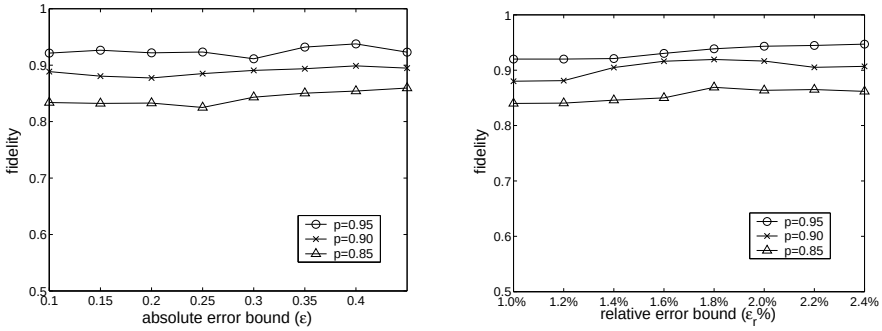
6.2 Measuring the Fidelity

We define the *fidelity* $f(o_i)$ as the percentage of time that the object o_i's source-cache deviation is within the error bound, that is,

$$f(o_i) = \frac{\text{the time cache-source error} \leq \epsilon}{\text{the total simulation time}}. \tag{10}$$

Fidelity measures how well our scheme meets the consistency requirements. The higher the $f(o_i)$ value, the more confidence we have to achieve stochastic consistency. Ideally, the fidelity value must match the user-provided confidence probability p, indicating the drifting Brownian motion model is accurate in characterizing source data.

Fig. 3 shows the fidelity values for the *salt* and *humd* traces under absolute error bounds (ϵ) and relative error bounds ($\epsilon_r\%$), respectively. We randomly generated a topology of 100 nodes and picked a source s^* at the highest level. We associated different data traces to s^* one at a time, and measured its fidelity. Each data point in Fig. 3 represents an average fidelity value over five traces from the same category. To generate a certain amount of traffic in the network, we



(a) the *salt* trace (b) the *humd* trace

Fig. 3. The fidelity

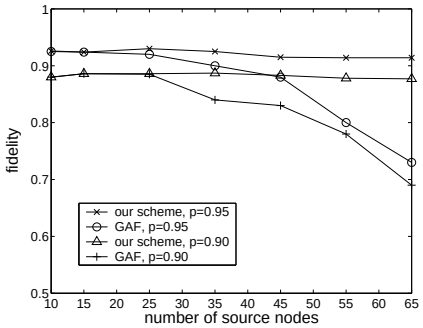


Fig. 4. Fidelity under various traffic loads (*temp* trace, $\epsilon = 0.1$)

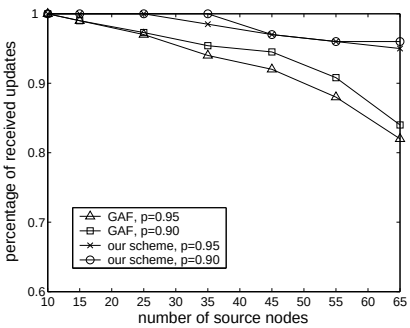


Fig. 5. The percentage of received updates at BS (*temp* trace, $\epsilon = 0.1$)

chose ten other sources in the topology, each associated with the same data trace, with a randomly chosen error bound.

Our scheme clearly achieves high fidelity for s^* under both categories of traces. The obtained fidelity value is very close to the corresponding confidence level p. For example, under the confidence level 90%, the average fidelity for the *salt* trace is 89.2%, while it is 89.5% for the *humd* trace. This is strong evidence for the accuracy of our DBM-based approach.

In Fig. 4, we show the impact of different network traffic loads on the fidelity. We used the same 100-node topology and chose a source s^*, associated with the *temp* trace with the error bound 0.1, at the highest level. We also chose a certain number of other nodes as sources, each associated with the same *temp* trace and the same error bound. We varied the number of sources from 10–65 and observed s^*'s fidelity. The amount of traffic in the network increases as the number of sources increases. Our scheme achieves a high and stable fidelity at confidence levels of 90% and 95%.

We also compared our scheme with GAF [1], an adaptive scheme that maintains a routing backbone in the wireless network, and puts other nodes to sleep as much as possible. We simulated GAF on the top of AODV [36]. With the

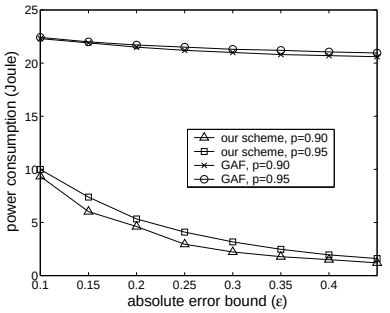


Fig. 6. The average power consumption per node (*temp* trace, 1000 *secs*)

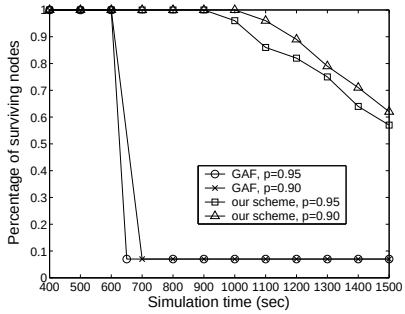


Fig. 7. The fraction of surviving nodes (*temp* trace, $\epsilon = 0.1$, 100 nodes)

same topology and input traces, GAF achieves the same fidelity as our scheme under light traffic loads, but much lower fidelity under heavy traffic loads.

This behavior is explained by Fig. 5, which shows the percentage of s^*'s update packets received by the BS, under various traffic loads. Starting from 45 sources, the percentage of received packets begins to drop rapidly under GAF, while it remains stable under our scheme. Since GAF ensures that a connected routing backbone is always available, heavy traffic loads will lead to severe contention in the wireless channel. Our scheme, however, is more flexible in adjusting each node's duty cycle, causing less channel contention and increasing throughput under heavy traffic loads.

6.3 Power Consumption

In Fig. 6, we show the average power consumption per node in the 100-node sensor network, under our scheme and under GAF. Let E_{i_0} and E_{i_r} be node s_i's intial energy level and remaining energy after simulation, respectively. The average power consumption per node is $\frac{\sum_i (E_{i_0} - E_{i_r})}{100}$. To ensure that no node runs out of power during our simulation, we set a high initial energy level (100 J). We chose 15 sources located as far from the BS as possible, each associated with the same *temp* trace and the same consistency requirement. We increased the error bound from 0.1 to 0.45 and measured the power consumption for a simulation time of 1000 *secs*.

In general, our approach consumes far less power than GAF. As the error bound increases, the difference is more significant, since fewer updates are generated, and our approach allows the nodes to sleep more often. More power is consumed for a higher confidence level ($p = 0.95$) since more updates must be generated and delivered to the BS.

To compare sensor network lifetimes under our scheme and under GAF, we show the fraction of nodes surviving after a given simulation time in Fig. 7. The initial energy level was set to 15 J. For simulation time less than 600 *secs*, all the nodes survive under our scheme as well as under GAF. However, beyond 600 *secs*, the survivor fraction drops rapidly under GAF, while it still remains 100%

under our scheme. With $p = 0.9$, we can achieve 67% longer network lifetime than GAF, while with $p = 0.95$, our lifetime is 50% longer.

7 Conclusions

We have proposed a novel approach to maintain stochastic consistency for erratic sensor sources. We achieve power efficiency by dynamically adjusting sensors' duty cycles. A node is guaranteed to be awake when an update message needs to be delivered/relayed, and asleep at other times. We model erratic sensor sources as drifting Brownian motions, and adpatively evaluate the model parameters at the sources. We have verified on various categories of real-life sensor traces that the DBM model faithfully captures the erratic data characteristics in the short term, and helps the source to adaptively evaluate when the next cache update is due, and notify the relaying nodes to wake up before this time.

Our scheme achieves high fidelity under the stochastic consistency model. Our fidelity is higher than that of GAF, which maintains a connected routing backbone and puts the other nodes to sleep, under heavy traffic loads and stringent consistency requirements, suggesting that we can attain higher throughput than GAF. Our approach also consumes significantly less power than GAF, since it is more flexible in adjusting each node's duty cycles, thus saving more power.

References

1. Y.Xu, J.Heidemann, D.Estrin: Geography-informed energy conservation for ad hoc networks. In: Proc. of the MobiCom Conf, Italy (2001)
2. S.Madden, M.J.Franklin, J.M.Hellerstein, W.Hong: The design of an aquisitional query processor for sensor networks. In: Proc. of the 2003 ACM SIGMOD Conf, San Diego (2003)
3. S.Madden, M.J.Franklin, J.M.Hellerstein, W.Hong: Tag: a tiny aggregation service for ad-hoc sensor networks. In: The 5th Symposium on OSDI. (2002)
4. C.Intanagonwiwat, R.Govindan, D.Estrin: Directed diffusion: A scalable and robust communication paradigm for sensor networks. In: Proc. of the ACM/IEEE MobiCom Conf. (2000)
5. Q.Han, S.Mehrotra, N.Venkatasubramanian: Energy efficient data collection in distributed sensor environments. In: Proc. of the 24th ICDCS Conf. (2004)
6. M.A.Sharaf, J.Beaver, A.Labrinidis, P.K.Chrysanthis: Tina: A scheme for temporal coherency-aware in-network aggregation. In: Proc. of the 3rd ACM MobiDE Workshop. (2003)
7. C.Olston, J.Jiang, J.Widom: Adaptive filters for continuous queries over distributed data streams. In: Proc. of the 2003 ACM SIGMOD, San Diego (2003)
8. A.Deligiannakis, Y.Kotidis, N.Roussopoulos: Hierarchical in-network data aggregation with quality guarantees. In: Proc. of the 9th EDBT, Greece (2004)
9. ASH Transceiver Designer's Guide, http://www.rfm.com, May, 2002.
10. Chipcon CC1000 RF Transceiver Datasheet, http://www.chipcon.com.
11. MPR/MIB Mote Sensor Hardware Users Manual, http://www.xbow.com.
12. S.Zhu, C.V.Ravishankar: Stochastic consistency, and scalable pull-based caching for erratic data sources. In: Proc. of the 2004 VLDB Conf, Toronto, Canada (2004)

13. F.Bennett, D.Clarke, J.B.Evans, A.Hopper, A.Jones, D.Leask: Piconet: Embedded mobile networking. In: IEEE Personal Communications Magazine. (1997)
14. C.Schurgers, V.Tsiatsis, S.Ganeriwal, M.Srivastava: Optimizing sensor networks in the energy-latency-density design space. In: IEEE Transactions on Mobile Computing. (2002) 1(1)
15. B.Chen, K.Jamieson, H.Balakrishnan, R.Morris: Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In: Proc. of the IEEE/ACM MobiCom Conf, Rome, Italy (2001)
16. S.Karlin, H.M.Taylor: A First Course in Stochastic Processes, 2nd Edition. Academic Press (1975)
17. S.Zhu, C.V.Ravishankar: A scalable approach to approximating aggregate queries over intermittent streams. In: Proc. of the 2004 SSDBM Conf, Santorini Island, Greece (2004)
18. R.Alonso, D.Barbara, H.Molina: Data caching issues in an information retrieval system. In: ACM Trans. Database Systems. (1990) 15(3)
19. H.Zou, N.Soparkar, F.Jahanian: Probabilistic data consistency for wide-area applications. In: Proc. of the 16th ICDE Conf. (2000)
20. W.Heinzelman, A.Chandrakasan, H.Balakrishnan: Energy-efficient communication protocols for wireless microsensor networks. In: Proc. of the Hawaii Intl. Conf on Systems Sciences. (2000)
21. W.Ye, J.Heidemann, D.Estrin: An energy-efficient mac protocol for wireless sensor networks. In: Proc. of the 21st InfoCom Conf, New York, NY (2002)
22. A.Deshpande, C.Guestrin, S.Madden, J.M.Hellerstein, W.Hong: Model-driven data acquisition in sensor networks. In: Proc. of the 30th VLDB. (2004)
23. G.Hartl, B.Li: infer: A baysian inference approach towards energy efficient data collection in dense sensor networks. In: Proc. of the 25th ICDCS Conf. (2005)
24. S.C.Huang, R.H.Jan: Energy-aware load balanced routing schemes for sensor networks. In: Proc. of the 10th Intl Conference on Parallel and Distributed Systems, Newport Beach, California (2004)
25. A.Woo, D.E.Culler: A transmission control scheme for media access in sensor networks. In: Proc. of the MobiCom Conf. (2001)
26. X.Hong, M.Gerla, W.Hanbiao, L.Clare: Load balanced, energy-aware communications for mars sensor networks. In: Proc. of the Aerospace Conf, vol 3. (2002)
27. A.M.Mood, F.A.Graybill, D.C.Boes: Introduction to the Theory of Statistics, 3rd Edition. McGraw-Hill (1974)
28. S.N.Neftci: An Introduction to the Mathematics of Financial Derivatives, 2nd Edition. Academic Press (2000)
29. H.C.Thode: Testing for Normality. Marcel Dekker, Inc. (2002)
30. The TAO Project, http://www.pmel.noaa.gov/tao/index.shtml.
31. http://mathworld.wolfram.com/InverseErf.html.
32. S.Zhu, W.Wang, C.V.Ravishankar: Stochastically Consistent Caching and Dynamic Duty Cycling for Erratic Sensor Sources, Technical Report, Univ. of California, Riverside, 2005, http://www.cs.ucr.edu/~szhu/sensorcache.pdf.
33. The Network Simulator ns-2, http://www.isi.edu/nsnam/ns/.
34. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE 802.11 Standard, 1997 Edition.
35. T.S.Rappaport: Wireless Communications, Principles and Practice. Prentice Hall (1996)
36. C.E.Perkins, E.M.Royer: Ad hoc on-demand distance vector routing. In: Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA (1999)