Estimating Clock Uncertainty for Efficient Duty-Cycling in Sensor Networks

Saurabh Ganeriwal, Deepak Ganesan[†], Hohyun Shim, Vlasios Tsiatsis, Mani B. Srivastava

Networked and Embedded Systems lab, 56-125B, EE-IV, University of California Los Angeles, CA 90095

† Department of Computer Science, University of Massachusetts, MA 01003

{saurabh, tsiatsis, shimho, mbs}@ee.ucla.edu, {dganesan}@cs.umass.edu

ABSTRACT

Radio duty cycling has received significant attention in sensor networking literature, particularly in the form of protocols for medium access control and topology management. While many protocols have claimed to achieve significant duty-cycling benefits in theory and simulation, these benefits have often not translated to practice. The dominant factor that prevents the optimal usage of the radio in real deployment settings is time uncertainty between sensor nodes. This paper proposes an uncertainty-driven approach to duty-cycling where a model of long-term clock drift is used to minimize the duty-cycling overhead. First, we use long-term empirical measurements to evaluate and analyze in-depth the interplay between three key parameters that influence long-term synchronization - synchronization rate, history of past synchronization beacons and the estimation scheme. Second, we use this measurement-based study to design a rate-adaptive, energy-efficient long-term time synchronization algorithm that can adapt to changing clock drift and environmental conditions while achieving application-specific precision with very high probability. Finally, we integrate our uncertainty-driven time synchronization scheme with a MAC layer protocol, BMAC, and empirically demonstrate one to two orders of magnitude reduction in the transmit energy consumption at a node with negligible impact on the packet loss rate.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer Communication Networks – *Network Protocols.*

General Terms

Algorithms, Experimentation, Performance, Verification.

Keywords

Sensor Networks, Time Synchronization, Sampling Period, Clock Drift, Polynomial Model Estimation, Rate Adaptation.

1. INTRODUCTION

Many important applications of sensor networks involve the detection of rare, random and ephemeral events [1]. Examples of such *event-response* applications are diverse and include intrusion detection, chemical spill monitoring, warning of imminent natural disasters, condition-based monitoring of complex equipment and structural integrity monitoring. In theory, sensor nodes deployed for these applications need to use the radio only when the rare events are observed, hence, radio energy consumption should be minimal. However, this is far from true in practice since much energy is expended in addressing time uncertainty between communicating nodes. This can be illustrated by a simple example. Consider a generic event-response network where, to save energy, the nodes are duty-cycled. By default the radio would be off but would wake up periodically to participate in potential network communication. Consider two nodes *A* and *B* that simultaneously decide to wakeup after time *t* to check if the other node has observed an interesting event. During this period *t*, clocks on both *A* and *B* can vary in several uncorrelated ways due to ambient conditions and clock crystal characteristics. Therefore, the local time at *A* and *B* can be quite different at time *t*, hence, *A* and *B* would wakeup at different times.

Existing duty-cycling techniques use a variety of approaches to deal with this uncertainty. A popular MAC layer protocol, BMAC [2], uses an asynchronous technique that involves no time synchronization or clock estimation whatsoever. Instead, each packet is transmitted with a long preamble which is chosen such that the receiver would wakeup some time during the preamble (refer to Figure 1). This incurs significant transmission overhead. For example with 11.5% duty-cycle a preamble of 250 bytes is used to transmit a 29 byte payload! Other techniques such as SMAC [3] and TMAC [4] use synchronized techniques where explicit time synchronization beacons are transmitted periodically between neighboring nodes. This enables the transmitter to turn on the radio at the right moment (refer to Figure 1), but the inability to deal effectively with time varying changes in clock drift force these techniques to re-synchronize frequently. For instance, one of the most efficient time synchronization protocols available in literature, FTSP [5], synchronizes once every minute to achieve 90µs synchronization error. To put it into perspective, if the event rate is once every hour, 60 extra synchronization beacons will be transmitted for every event notification packet. Thus, existing radio duty-cycling approaches expend a lot of energy in handling time uncertainty between sensor nodes.

The lack of techniques to accurately estimate time uncertainty also impacts the ability to deploy long-lived sensor network applications. Although available time synchronization implementations such as FTSP [5], RBS [6] and TPSN [7] can synchronize a pair of nodes within a few microseconds, their focus has been on achieving accurate instantaneous synchronization. These approaches do not enable us to understand either how error accumulates over long periods into the future or how to choose the synchronization frequency for bounding the time uncertainty. While these schemes are important for several applications that require short-term synchronization such as measuring the time-of-flight, acoustic beamforming and tracking, they are insufficient for efficient duty-cycling as well as for applications that require continuous time synchronization such as coordinated actuation and synchronized sampling. In real application scenarios where these approaches have been employed, the choice of synchronization frequency has been handpicked to correspond to the precision requirements of the application. For example, the CENS deployment at James Reserve [8] uses RBS to synchronize after every 5 minutes and the Shooter localization system [9] uses FTSP to synchronize once every 45 seconds. This has two drawbacks. First, the synchronization frequency needs to be manually configured for each new application synchronization requirement and deployment environment. Second, the chosen synchronization frequency is often highly suboptimal energy-wise as a result of the manual configuration.

Thus, there is a critical need for better datasets, models and techniques for estimating and bounding the long-term uncertainty in clocks. This will have significant benefits across a broad spectrum of sensor network applications as well as for energy-efficient MAC layer design. The re-synchronization period in SMAC can be of the order of tens of minutes (from the current periods of a few seconds), and BMAC can use significantly shorter sender-side preambles (from hundreds of bytes to a few bytes). Furthermore, the sleeping periods of nodes can be increased significantly (from 100ms to many tens of minutes) while not sacrificing synchronization error. Applications with different long-term synchronization requirements such as triggered actuation (100 ms bound), synchronized sampling (e.g.: structural monitoring with 10ms bound) can use our scheme without pre-configured parameters to obtain the minimum energy synchronization solution for the specified error bound. Data processing techniques can use the uncertainty in time between clocks to determine how to align and compress time series data that was sensed at different nodes.

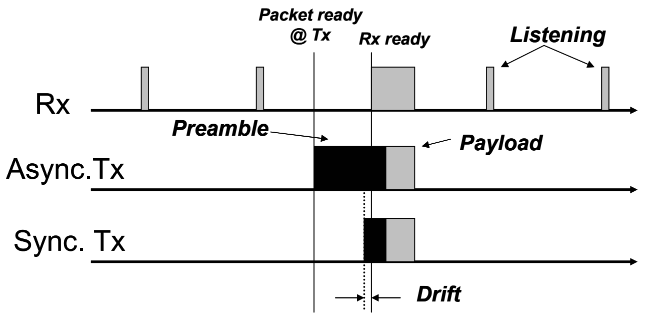


Figure 1. **Transmission Mechanisms**

1.1 Contributions

Our core contributions are three-fold. First, we experimentally obtain long time-scale data sets both in indoor and outdoor settings for Berkeley Mica2 motes. We then perform a detailed characterization of size of history and time synchronization beaconing period (or rate) on accuracy and energy requirements. A very important revelation of this study is that the optimal history of samples to choose for prediction is roughly constant across different sampling rates for polynomial estimators.

Second, we use the results of our empirical analysis to design an adaptive time synchronization protocol that can provide bounded error under varying ambient conditions, simultaneously optimizing the energy overhead. We will show that in comparison to the best possible hypothetical periodic re-synchronization scheme, the minimum and maximum energy gains are 1.1x and 12.5x respectively for achieving the same error bound. Our scheme outperforms the point solutions proposed by existing time synchronization schemes by one to two orders of magnitude.

Lastly, we develop a prototype implementation on Mica2 motes for sense-response applications. This system integrates our long-term time synchronization protocol with BMAC. We use a much shorter preamble for transmitting the packets at a nominal overhead of keeping the nodes in sync within a desired error bound. We will demonstrate that this reduces the transmit energy consumption by one to two orders of magnitude.

2. RELATED WORK

Time synchronization algorithms aim at synchronizing the local clocks of the nodes in the network. The most widely adopted protocol is the Network Time Protocol (NTP) [10]. NTP has been widely deployed and proven to be effective, secure and robust in Internet. However, the time synchronization requirements in sensor networks are more stringent, often requiring an accuracy of a few microseconds. NTP is also computationally intensive and is not focused on energy efficiency.

Several time synchronization protocols have been developed to deal with the precision and energy requirements of sensor network applications [11]. Some of the notable ones are Reference Broadcast Synchronization (RBS) algorithm [6], Timing-sync Protocol for Sensor Networks (TPSN) [7] and Flooding Time Synchronization Protocol (FTSP) [5]. There exist prototype implementations for each of these approaches that can achieve within a few microseconds synchronization accuracy over a minute period. However, these results are based on short-term synchronization. To the best of our knowledge, there exists no work in sensor networks that enables us to understand the accumulation of error over a long period of time.

While long-term synchronization has not been the focus of sensor network research, it has been studied in the context of NTP [10] for the Internet. Mills [12] reports computer clock drifts over several days and uses this data to design interesting extensions to NTP. However, since energy is not a constraint, this work focuses on achieving tighter synchronization rather than minimizing the energy overhead. Moreover, this solution is static and does not adapt to the varying user-level precision requirements.

3. PROBLEM FORMULATION

We consider the problem of adaptive long-term synchronization of a pair of sensor nodes *A* and *B*, where node *A* sends beacons to *B*. Each beacon results in generating a new sample (T_A, T_B). T_A is the time at which the beacon packet is transmitted by node *A* and T_B is the receipt time of this packet at node *B*. Note that T_A and T_B are measured by the local clocks of *A* and *B* respectively. Node *B* has a history of these samples and wishes to estimate the minimum frequency of beaconing such that it can synchronize with node *A* within a user-defined synchronization error bound, E_{max}. The problem can be represented by the feedback control loop shown in Figure 2. A window of past samples (T_A, T_B) from the samples repository (Block A) is input to an estimator (Block B). The estimator uses these observations to estimate the relative clock model between A and B. This model can then be used to predict the future time at node B. Besides predicting the future time, a confidence estimate of this prediction is also calculated (Block C). This prediction error is then compared with the given error bound, based on which a minimum beaconing period is calculated. A new sample is taken (Block D) based on this new sampling period, which then gets added to the repository of samples.

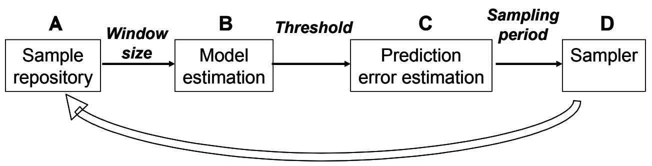


Figure 2. **Time Synchronization Control Loop**

3.1 Model Estimation

Throughout this paper, we will use the following polynomial model for representing the relative clock model between A and B:

$$T_B = \sum_{k=0}^{K} (\beta_k * T_A{}^k) + \varepsilon \ldots (1)$$

In (1), T_B is represented as a simple K^{th} degree polynomial in T_A. The error, ε, is meant to capture both measurement errors as well as environmental factors that are known to influence clock behaviors but are not explicitly modeled by the simple polynomial model in (1). We return to the incorporation of these factors later in the paper. Given a window of n observations ($T_{A,i}$, $T_{B,i}$), $i=1,..,n$, we estimate the parameters, $\beta_0,...,\beta_K$, using ordinary least squares (OLS) as:

$$RSS = \min_{\beta_k \forall k=0,\ldots,K} \sum_{i=1}^{n} \left(T_{B,i} - \left[\sum_{k=0}^{K} \beta_k T_{A,i}{}^k \right] \right)^2 \ldots (2)$$

Refer to [13] for computational and theoretical details. Here, RSS stands for the residual sum of squares, the accumulated squared errors between the fitted model and the original data, $T_{B,i}$. The method of OLS assumes that the best-fit curve of a given type is the curve that has the minimal sum of the deviations squared (*least square error*) from a given set of data. The values of the parameters that minimize (2) are denoted by $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_K$.

3.2 Sample Repository

The sample repository consists of tuple of the form (T_A, T_B). A new sample is formed when node A sends a beacon packet to B. The sampling period, S, corresponds to the periodicity with which A sends these beacon packets to B (inverse of the sampling rate). Taking a new sample imposes an energy cost on the system. However, new samples need to be taken to re-estimate the relative clock model so that the error remains bounded. Thus, the sampling period has a direct impact on both the energy and error performance of the system. Intuitively a higher sampling period will correspond to less energy consumption and a higher synchronization error and vice-versa. Our objective is twofold – (1) S should change adaptively in accordance with the system and environmental dynamics, and (2) The system should be operating at maximum S possible to meet the desired user level precision.

Instead of using all the samples from the repository, we only use a few most recent samples, equal to the *window size*, for estimating the clock model. The impact of the window size on both the energy and error performance is less intuitive.

3.3 Prediction Error Estimation

Equation (1) can be used for predicting the future time at node B based on the time at node A. However, going even one step further, we want to estimate the uncertainty in this prediction. In this paper, we describe an approach of estimating this uncertainty based on a combination of analytical and empirical techniques. We concentrate only on the one-step look-ahead prediction i.e. the prediction uncertainty in the next immediate sample. From now onwards, we will use prediction error (E_p) to refer to the error at just this specific point.

3.4 Objective

The optimization problem can be formulated as: Given a repository of samples, find the combination of window size (W) and the degree of fit (K) that can maximize the sampling period (S) (and correspondingly minimize energy) while bounding the prediction error (E_p) within a user-defined error bound (E_{max}).

Given: Past repository of samples
Find: W, K and S
Optimize: Maximize S
Constraint: $E_p < E_{max}$

Figure 3. **Optimization Problem**

4. EMPIRICAL STUDY

The experimental set-up consists of two motes, A and B, with A periodically sending beacon packets to B. Node A notes the send time (T_A) and node B the receipt time (T_B) with their local clocks respectively. Motes send the timestamp information to IPAQs over the serial port, where it gets written in a file. The motes as well as the IPAQs are powered through a 5V DC supply. We also kept a log of the temperature throughout the duration of the experiment using two Mica2 sensor boards, one attached on each A and B. We have used MTS420 sensor boards that come with calibrated temperature sensors (accuracy within 1^0C). We conducted different experiments across widely varying environments as well as on different pairs of devices. The objective was two-fold – (1) study the behavior across different environments to validate our algorithms in different potential sensor network deployment environments, and (2) isolate hardware/clock crystal effects on individual devices. We collected four data sets; the salient features are described in Table 1.

Table 1. Salient features of the data sets

Identification tag	**Duration (hours)**	**Temperature range (^{0}C)**	**Mote**	
			A	**B**
Indoor-I	30	25 – 26	1	2
Indoor-II	12	25 – 26	3	4
Outdoor-I	26	17 – 21	5	6
Outdoor-II	12	22 – 27	3	4

Every data set was collected by keeping the sampling period to 5 seconds. For the empirical study, we extract higher sampling period from this basic data set. For example, if we want to analyze a data set with a sampling period of 1 minute, we consider every 12th (60 / 5) sample. A simple sliding window scheme is used for this empirical study, whereby after choosing *W*, *S* and *K* we estimate the clock model and E_p. After this, we slide the window by one, thereby dropping the oldest sample in the window and taking in the new sample. This is repeated throughout the duration of the data set; the following sections show the average statistics.

To understand the interaction between sampling period, sample history and estimation scheme, we first study these parameters in isolation and then in conjunction. Over short timescales, there is general agreement that a linear relative clock model (K=1) is sufficient [5, 6, 7]. We first study the interplay between the sampling period and the window size making this assumption. We will compare and contrast the performance of higher degree polynomial estimators with linear regression in later sections.

4.1 Impact of History

In this section, we study the impact of sample history on error. We fix the sampling period and the degree of fit (linear) and vary the window of samples used by the estimator to estimate the clock model. Figure 4 plots the prediction error for a sampling period of 1 minute over varying window sizes for Indoor-I.

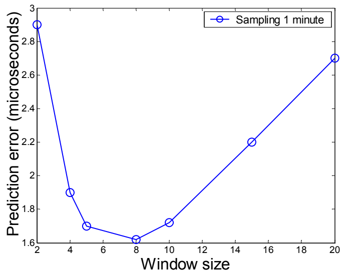


Figure 4. **Optimal window size**

The key revelation is the existence of an optimal window size, represented as W^*, at which the error is minimized. In Figure 4, W^* is equal to 8. The existence of an optimal window size and rapidly increasing error for both less ($W<W^*$) and greater history ($W>W^*$) suggests that the choice of history window should be carefully performed. At lower window sizes ($W<W^*$), there are not enough samples available for obtaining the accurate clock model, whereas for higher window sizes ($W>W^*$) the drift no longer remains a constant and hence, the available degrees of freedom (2 as it is a linear estimator) are not enough to accurately capture the dynamics of the clock model.

4.2 Impact of Sampling Period

We now study the dependency of this optimal window size on the sampling period. Figure 5 plots the optimal window size for different sampling periods for all the data sets. Two interesting observations can be made from Figure 5. First, the value of the optimal window size changes with the sampling period. For example, in the case of Indoor-I, the optimal window size is approximately 4 for a sampling period of 2 minutes, whereas it changes to 25 for a sampling period of 15 seconds. For sampling periods higher than 2 minutes, the optimal window size is the minimum (=2), thus the use of time history is not useful for high sampling periods. Second, although the trends and knee points are very similar for different datasets, the choice of optimal window size corresponding to each sampling period varies slightly with the choice of the environment.

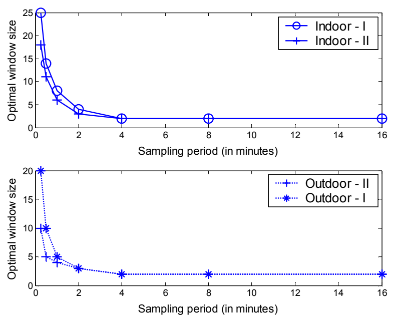


Figure 5. **Sampling period v/s optimal window size**

Figure 6 plots the prediction error at these optimal points of operation. The prediction error increases monotonically with the sampling period. Although intuitive, this is the tradeoff that we seek to exploit while developing an adaptive scheme for resynchronization. Operating at a higher sampling period will lead to less energy consumption but more error and vice-versa.

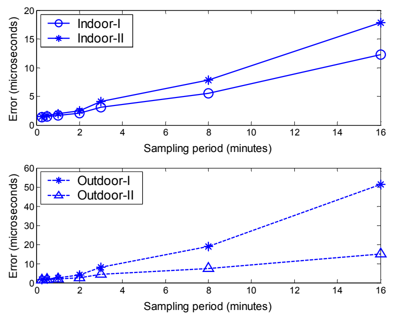


Figure 6. **Error at the optimal point of operation**

4.3 Optimal Time Window

In the previous section, we were focused on the optimal window size. However, an interesting metric is the optimal time window calculated as the product of optimal window size and sampling period. Figure 7 plots this time window for varying sampling periods for different data sets.

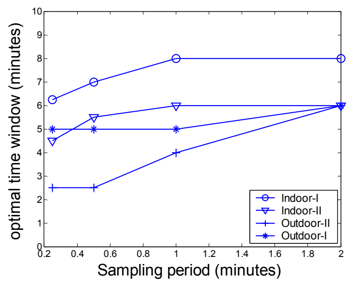


Figure 7. **Existence of an optimal time window**

Two very interesting observations emerge. First, the optimal time window is quite small, within 8 minutes for all the datasets.

Second, the time window is roughly constant for a given dataset, although Outdoor-II shows some discrepancy. We represent this optimal time window by T and it can be approximated to be 8, 6 and 5 minutes for Indoor-I, Indoor-II and Outdoor-I respectively. The existence of the optimal time window can also be predicted and explained by a non-classical statistic, Allan Variance [14], which is used to estimate the stability of clocks. For lack of space, we will not describe this in greater detail in this paper.

The existence of the optimal time window implies that if model estimation is done over this window of time, the prediction error is minimized. For example, any time history beyond 8 minutes of data does not improve prediction error for future timestamps in Indoor-I. However, for higher sampling periods the time history will automatically exceed T; in which case the minimum window size ($=K+1$) will be optimal. Note that we need a minimum of $K+1$ samples for doing a *K-polynomial* model estimation. This explains why W^* is always equal to 2 for all sampling periods greater than 8 minutes for every data set in Figure 5. The existence of the optimal time window also makes it easy to discover the optimal point of operation. If T is known W^* can be calculated for any given S as follows:

$$W^* = \max\left\{K+1, \left\lceil \frac{T}{S} \right\rceil\right\} \ldots(3)$$

5. ERROR PREDICTION

Revisiting Figure 2, we know how to choose the window size for a given sampling period and do relative clock model estimation. The next building block in Figure 2 is the ability to accurately *predict* the error (E_p) given past history of beacons.

5.1 Estimating Prediction Error

Given a time window of n observations, ($T_{A,i}$, $T_{B,i}$), we can predict the time at node B given a new time at node A, T_A, using our OLS estimates derived in (2) as:

$$\hat{T}_B = \hat{\beta}_0 + \hat{\beta}_1 T_A \ldots(4)$$

Following standard regression theory [13], we can construct a ($1-\alpha$) confidence interval for this prediction as:

$$\hat{T}_B \pm [t_{(1-\alpha)/2,n-2} * SE(\hat{T}_B)] \ldots(5)$$

The first term in the product in equation (5) refers to an upper quantile of the t distribution with $n-2$ degrees of freedom, and the second term is the standard error of the predicted value [13]. We represent this product, the estimate of the prediction error (E_p), by δ. In this paper, we use the 95% confidence interval to estimate δ. Note that 95% confidence interval is the typical choice made in existing literature due the Gaussian assumption on the error distribution, ε (see equation (1)).

5.2 Accuracy of Estimation

If the error term, ε, in (1) has a normal distribution, then these confidence intervals behave as given by (5). However, there are many variables left out of (1), including temperature and humidity, hardware and other system errors that we cannot take into account explicitly to avoid the estimator from becoming too complex for simple sensor nodes. As a result the error term is likely to have more structure in it. Instead, a model of the form in equation (6) might be more appropriate.

$$T_B = f(T_A, x) + \varepsilon^* \ldots(6)$$

Here $f(.)$ is some function that depends on T_A as well as the environmental conditions (described here by the vector of variables x), and ε^* is a zero mean, normal error term. When we fit a linear relative clock model, we are leaving out something systematic, calling into question our original assumptions about the error terms in (1). When a more elaborate model is actually true, our calculations leading to the confidence set (5) are now incorrect. Instead, one can decompose the prediction error into:

$$\begin{aligned} T_B - \hat{T}_B &= f(T_A, x) + \varepsilon^* - (\hat{\beta}_0 + \hat{\beta}_1 T_A) \\ &= \varepsilon^* + [(\beta_0^* + \beta^*{}_1 T_A) - (\hat{\beta}_0 + \hat{\beta}_1 T_A)] + [f(T_A, x) - (\beta_0^* + \beta^*{}_1 T_A)] \\ &= \varepsilon^* + [E\hat{T}_B - \hat{T}_B] + [f(T_A, x) - (\beta_0^* + \beta^*{}_1 T_A)] \ldots(7) \end{aligned}$$

Here $\beta_0^* = E[\hat{\beta}_0]$ and $\beta_1^* = E[\hat{\beta}_1]$, the expected values of our estimates. Recall that β_0^* and β_1^* are given explicitly as the OLS solutions to equation (2), where we replace the polynomial fitting model with $f(T_A, x)$, see [13] for details. The first two terms on the right in (7) are used in calculating the standard error that appears in our confidence interval (5); the first reflects an irreducible error, and the second describes the variance in our estimate (the difference between a random variable, $\hat{T}_B$, and its expectation). The last expression in brackets is known as a bias term and is what we leave out by not knowing the proper form of $f(.)$.

By ignoring the bias, our confidence intervals do not have the correct coverage properties. There are many techniques in the statistics literature that attempt to estimate or correct for the bias, many of which are computationally expensive. Given our need for efficient calculations, we have chosen a simple approach in which we widen the confidence bands, introducing a scaling factor, Δ:

$$\hat{T}_B \pm [\Delta * t_{(1-\alpha)/2,n-2} * SE(\hat{T}_B)] \ldots(8)$$

5.3 Scaling Factor

More formally, scaling factor, Δ, is defined as the ratio of actual prediction error, E_p, and the estimation of this error obtained from equation (5), δ:

$$\Delta = \frac{E_p}{\delta} \ldots(9)$$

Figure 8 plots the cumulative distribution function of the scaling factor, for Indoor-I for different sampling periods. For every sampling period, we fix the window size corresponding to the optimal time window. Figure 8 can be used for deciding the value of scaling factor that probabilistically bounds $x\%$ of the prediction error. For example choosing $\Delta=6$ will probabilistically bound 90% the prediction error whereas $\Delta=2$ will only bound 60% of the prediction errors. We refer to $x\%$ as the *cut-off threshold* and represent it by λ. We now evaluate the impact of varying environmental conditions on the scaling factor. For this, we calculate the cumulative distribution function of the scaling factor for various data sets and sampling periods. From these plots, we calculate the value of the scaling factor corresponding to the cut-off threshold of 60, 80 and 95 respectively. The results are summarized in Table 2.

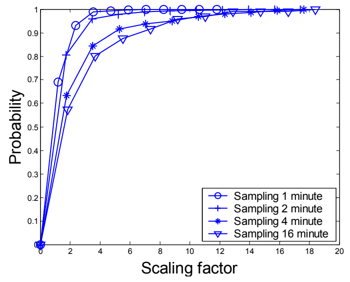


Figure 8. **Cumulative distribution of scaling factor**

For any particular dataset, the value of Δ increases with increasing sampling period (for a chosen threshold). Finding out the exact relationship between the scaling factor and the sampling period is rather complex and outside the scope of this paper. In this paper, we find out the scaling factor for a given cut-off threshold, λ, at a medium sampling period, say 4 minutes, and then assume it to be same across every sampling period. We will show that even such simplistic scheme is able to give us considerable gains. A more surprising and pertinent observation is that the scaling factor is roughly constant across different environments (for a chosen threshold and a set sampling rate). This has an important implication for protocol design: the scaling factor can be expected to be consistent across different deployments and times.

Table 2. Scaling factor

Data Set	**S = 1 minute**			**S = 16 minute**		
	λ =60	λ =80	λ =95	λ =60	λ =80	λ= 95
Indoor-I	1.1	1.7	2.8	2.1	3.6	6.1
Indoor-II	1.0	1.6	2.8	2.0	2.9	6.3
Outdoor-I	1.3	2.0	3.7	2.2	2.7	6.9
Outdoor-II	1.0	1.8	3.2	1.6	2.9	6.3

6. RATE ADAPTIVE SYNCHRONIZATION

Our empirical study provides an in-depth understanding of the impact of history and sampling period on the synchronization error. In this section, we propose the Rate Adaptive Time Synchronization (RATS) protocol based on the empirical study.

6.1 Learning Parameters

The time synchronization control loop shown in Figure 2 requires two parameters for its operation: *optimal time window (T)* and *scaling factor (Δ)*. A simple strategy of learning these parameters in a deployed system can be to use an initialization phase. Since the scaling factor varies little with the actual environment, this initialization can be done together with in-factory calibration before the nodes are even deployed. This will reduce the power requirements of this initialization. A post-deployment initialization phase for calculating the optimal time window will always be needed, as the optimal time window changes with the environment in which the system is deployed. In the initialization phase, a data set can be collected in a similar manner as in Section 4, and then the empirical analysis of section 4 and 5 can be used to calculate these parameters. This approaches hinges on the assumption that the optimal time window and scaling factors have a *temporally stationary distribution* so that the values learned over an initial window of time are valid for long durations.

We evaluated the efficacy of this assumption and the training-based approach. Figure 9 plots the value of the optimal time window v/s duration of the learning phase for Indoor-I and Outdoor-I. For a given duration of the learning phase, we took 50 different measurements by varying the starting point randomly over the duration of the data set. The plot shows the average statistics; the standard deviation is shown by the vertical bars. As expected, the accuracy is higher for a longer duration of the learning phase. A 2-4 hour learning phase seems to hit the sweet spot across all the datasets that we studied. We also calculated the scaling factor for varying durations of the learning phase and a 2-4 hour learning phase seems appropriate here as well. This clearly highlights the feasibility of training the system to estimate the two parameters within reasonable levels of accuracy, by using a small window of training information.

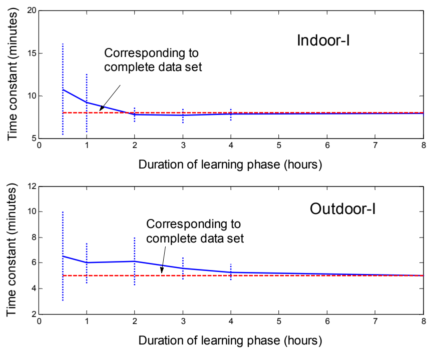


Figure 9. **Impact of the duration of learning phase**

6.2 Protocol Description

The Rate Adaptive Time Synchronization (RATS) protocol tries to imitate the time synchronization control loop of Figure 2 at runtime. The objective of RATS is to repeatedly calculate the new sampling period so that the synchronization error remains bounded within the user specifications. The pseudo-code for the protocol is shown in Figure 10.

```
1. Compute $W^* = max(K+1, T/S)$
2. Find ($\hat{\beta}_o, \hat{\beta}_1$)using a window of $W^*$ samples in equation (2)
3. Compute $\delta$ using equation (5)
4. Find $E_p = \Delta * \delta$
5. if ($E_p < [E_{max} * \eta_{low}]$)  $S = S * MIMD_{inc}$
   elseif ($E_p > [E_{max} * \eta_{high}]$) $S = S / MIMD_{dec}$
6. if ($S < S_{min}$) $S = S_{min}$
   elseif ($S > S_{max}$) $S = S_{max}$
```

Figure 10. **Psuedo code for RATS**

It starts with calculating the optimal window size, W^* (using the optimal time window), for the given sampling period, *S*. The relative clock model is estimated using a linear estimator on the sample history equal to the optimal window size. The prediction error, δ, is then computed and scaled using the scaling factor, Δ. If the error is below the lower threshold, we multiplicatively increase sampling period, and if it is above the higher threshold, the sampling period is decreased multiplicatively. The sampling period remains unchanged if the error is between the two thresholds. At the end, we make sure that the new sampling period is within [S_{min}, S_{max}] to avoid the unbounded increase/decrease of the sampling period.

Our reasons for choosing a multiplicative increase, multiplicative decrease (MIMD) strategy are two-fold. First, an MIMD scheme enables fast convergence to a sampling rate that minimizes energy as well as provides quick response to changing environmental or system conditions that result in loss of time synchronization precision. Second, MIMD is not as complex as an exponential scheme and becomes really simple if the multiplicative factors are powers of two. Although additive increase/decrease is equally simple, it doesn't adapt fast. Table 3 summarizes the parameter settings of the RATS protocol that were kept fixed in our analysis. For lack of space, we do not provide an in-depth study of the impact of all these parameters settings. Instead, we provide a set of representative instances in this paper.

Table 3. Parameter settings of MIMD

Parameter	Value
Upper threshold fraction	$\eta_{high} = 0.9$
Lower threshold fraction	$\eta_{low} = 0.75$
Mode of operation	Optimistic (λ=60) Balanced (λ=75) Pessimistic (λ=90)
Multiplicative increase factor	$MIMD_{inc} = 2$
Multiplicative decrease factor	$MIMD_{dec} = 2$
Minimum sampling period	S_{min} = 30 seconds
Maximum sampling period	S_{max} = 64 minutes

6.3 Performance Evaluation

We evaluated the performance of RATS on all the four datasets. Note that no prior knowledge of *any* parameter in the system including the deployment conditions is provided to the protocol at startup. Instead, a 2 hr long learning phase is used to learn the optimal time window and the scaling factor, as mentioned in Section 6.1. Table 4 summarizes the values of these learned parameters, which we use in the following subsections.

Table 4. Parameter settings for the data sets

Parameter		Value			
		Indoor-I	Indoor-II	Outdoor-I	Outdoor-II
T (in minutes)		8	6	6	3
Δ	Optimistic	1.86	1.71	1.47	1.96
	Balanced	2.61	2.14	1.95	2.62
	Pessimistic	3.62	3.9	3.4	4.12

6.3.1 *Metric of Evaluation*

The dual goal of our algorithm is to minimize the energy consumption in terms of synchronization beacons while achieving bounded synchronization error as requested by the application. We use the following two metrics to capture these goals:

Faulty ratio (E_F): This is defined as the percentage number of times the synchronization error crosses the desired user-defined synchronization error bound, E_{max}. E_F is normalized to the total number of measurements in order to express it as a percentage. To calculate the faulty ratio, we calculate E_p for every sample in the data set i.e. at the minimum granularity of 5 seconds.

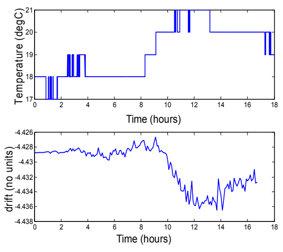
$$E_F = \begin{cases} E_F + 1 \forall |E_p| \geq E_{max} \\ E_F \forall |E_p| < E_{max} \end{cases} \ldots(11)$$

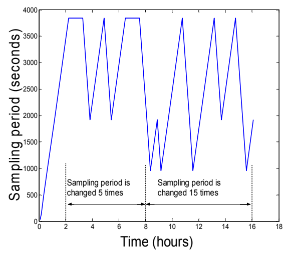
Average sampling period (S_{avg}): This is expressed as average of the sampling period at which the system operates over the complete duration of the data set. If the algorithm samples for time T_i at sampling rate S_i and there are a total of M different sampling rates, then:

$$S_{avg} = \frac{\sum_{i=1}^{M} S_i T_i}{\sum_{i=1}^{M} T_i} \ldots(12)$$

6.3.2 *Gauging the Adaptive Nature of RATS*

Figure 11(a) plots the variation of temperature and drift between the nodes over the complete duration of Outdoor-I. As anticipated, the rate of change of drift is highly correlated with the rate of change of temperature. Figure 11(b) shows the performance of RATS (balanced version); it plots the sampling period at which the system operates over the course of the experiment. The user-defined error bound is 60µs.





(a) Drift and temperature variation (b) Performance of RATS

Figure 11. **Runtime adaptation to system dynamics**

RATS makes just 5 transitions in the first half compared to 15 transitions in the second half. The average sampling period during the first and second half of the experiment is 59 and 42 minutes respectively. Thereby, RATS adapts itself to the more dynamic nature of the system during the second half when temperature variations result in rapid changes in drift.

6.3.3 *Energy v/s Error Tradeoff*

Figure 12 plots the average sampling period (S_{avg}) and the faulty ratio (E_F) for different user-defined error bounds for Outdoor-I. For any version of RATS, S_{avg} increases and E_F decreases with E_{max}, implying that RATS adapts itself automatically to meet the desired precision.

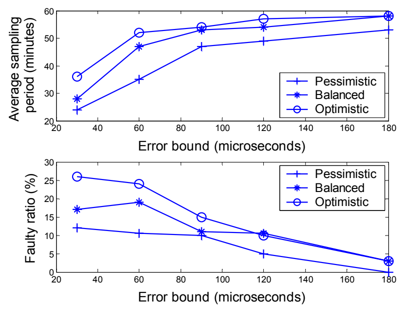


Figure 12. **Energy & Error performance of RATS**

6.3.4 *Comparing with Existing Approaches*

Existing long-term time synchronization protocols are based on periodically re-synchronizing the network. Figure 13 plots the faulty ratio and the average sampling period for the periodic scheme for Indoor-I. These protocols are non-adaptive and hence the average sampling period will be equal to the fixed sampling period at which the system operates. The desired error bound is fixed to 90µs. Unlike RATS, the error bound does not have any effect on the operation of the periodic scheme; its only significance is for calculating E_F. The window size is fixed to the corresponding W^* for every sampling period. Note that all proposed implementations in literature (*S=30s & W=8*, *S=300s & W=8* in FTSP, *S=10min & W=5* in Great Duck Island, *S=5min & W=5* in James Reserve, *S=1min & W=8* in shooter localization) will be just different points of operation on the curve in Figure 14. The different points of operation for the three versions of RATS are also shown in Figure 13 for comparison.

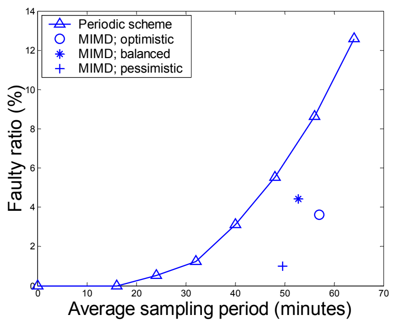


Figure 13. **Comparison with the non-adaptive sceheme**

As can be seen from Figure 13, all the versions of RATS lie on the right side of the curve corresponding to the periodic scheme. This implies that RATS either achieves same error operating at a higher average sampling period (viewed horizontally) or achieves less error operating at the same sampling period (viewed vertically) as compared to the periodic scheme. Thereby, RATS gives a superior performance *both in terms of energy and error* as compared to the existing approaches or any possible implementation (choice of *S* and *W*) of the periodic scheme.

6.3.5 *Comparison with Ideal Periodic Scheme*

We now evaluate the benefit of using adaptive MIMD synchronization in RATS by comparing it to a hypothetical ideal periodic synchronization scheme. We take a version of the RATS scheme and calculate the tuple $(S_{avg}, E_F)_{RATS}$ for a given error bound. We then obtain a similar plot as in Figure 14 for the periodic scheme. From this plot, we find the sampling period, $(S_{avg})_{FIXED}$, at which $(E_F)_{RATS} = (E_F)_{FIXED}$. Note that $(S_{avg})_{FIXED}$ represents the upper bound, from an energy perspective, that any periodic scheme can hope to achieve for the same error performance as RATS. The ratio $(S_{avg})_{RATS} / (S_{avg})_{FIXED}$ represents the relative energy gains of RATS vis-à-vis the best possible periodic scheme. Similarly, we gauge the relative error gains of RATS with the best possible periodic scheme by calculating $(E_F)_{FIXED}$ for which $(S_{avg})_{RATS} = (S_{avg})_{FIXED}$. Table 5 presents the results for a fixed error bound of 90 microseconds over different data sets. The minimum energy gains we got are approximately 1.1x over the periodic scheme whereas the maximum energy improvement is an order of magnitude, 12.5x. Similarly the minimum and maximum error gains are of the order of 1.25x and 13.5x respectively. We repeated this experiment for an error bound of 60 and 120 microseconds and we did not observe even a single scenario in which the performance of RATS was worse than the periodic scheme either in terms of energy or error.

Table 5. Comparison of RATS with ideal periodic scheme

Data set	**Version**	**$(S_{avg})_{RATS}$ (minutes)**	$\frac{(S_{avg})_{RATS}}{(S_{avg})_{FIXED}}$	**$(E_F)_{RATS}$ (%)**	$\frac{(E_F)_{FIXED}}{(E_F)_{RATS}}$
Indoor-I	Optimistic	56.99	1.36	3.63	2.51
	Balanced	52.82	1.19	4.42	1.67
	Pessimistic	49.53	1.69	1.00	6.72
Indoor-II	Optimistic	56.1	3.57	0.93	13.47
	Balanced	54.26	5.16	2.03	5.76
	Pessimistic	49.48	12.52	0.87	10.93
Outdoor-I	Optimistic	54.33	1.1	14.81	1.31
	Balanced	53.62	1.26	10.55	1.78
	Pessimistic	47.4	1.16	9.65	1.38
Outdoor-II	Optimistic	22.02	1.15	18.43	1.45
	Balanced	21.72	1.11	19.71	1.31
	Pessimistic	19.55	1.08	15.9	1.24

These results demonstrate two key benefits of our adaptive scheme. First, an adaptive scheme *always* out-performs the *best* periodic synchronization scheme both in terms of energy and error. This demonstrates the effectiveness of our protocol and the potential benefits that can be obtained. Secondly, and very importantly, our algorithm requires *no initial parameters*, *no application specific customization,* and it *can adapt automatically to system dynamics*. The periodic sampling scheme can perform poorly in terms of error under constantly varying outdoor environments, whereas our adaptive scheme achieves low error even under these conditions.

6.3.6 *Impact of Higher Order Polynomial Estimation*

Although RATS is always able to outperform a periodic strategy, its relative performance in an outdoor setting is worse than indoors (refer Table 5). This hints towards looking for better estimators and as a first step in this direction, we wanted to investigate the performance benefits of using higher degree polynomial estimators such as quadratic and cubic. For this, we repeated the performance evaluation of the previous section with a quadratic estimation strategy. Table 6 summarizes the results.

The observations to be made from Table 6 are twofold. First, in an indoor setting linear estimation always performs better. Second, in an outdoor setting, where the performance of RATS was worse, the quadratic estimation strategy does not seem to give consistent improvement. In fact in most of the cases, the performance is worse than using linear regression. These results indicate that using higher degree polynomial estimators results in diminishing gains for the additional complexity. The right approach to obtain more accurate results is to explicitly include ambient conditions such as temperature and humidity, hardware and other systems error into the estimator. We are currently exploring both empirical and statistical techniques such as Kalman filtering in this regard.

Table 6. Comparison of linear v/s quadratic estimation

Data set	Version	Linear		Quadratic	
		S_{avg}	E_F	S_{avg}	E_F
Indoor-I	Optimistic	56.99	3.63	55.13	4.4
	Balanced	52.82	4.42	53.14	3.28
	Pessimistic	49.53	1.00	47.00	1.42
Indoor-II	Optimistic	56.1	0.93	54	8.1
	Balanced	54.26	2.03	54	8.1
	Pessimistic	49.48	0.87	50.24	2.1
Outdoor-I	Optimistic	54.33	14.81	56.3	16.46
	Balanced	53.62	10.55	52.83	15.68
	Pessimistic	47.4	9.65	46.12	9.2
Outdoor-II	Optimistic	22.02	18.43	39.64	26.5
	Balanced	21.72	19.71	38.45	35.32
	Pessimistic	19.55	15.9	17.36	8.99

7. IMPLEMENTATION

RATS has been implemented and is successfully working on Mica2 motes. The implementation is in NesC and the underlying operating system is TinyOS. Due to space constraints, we only briefly discuss the implementation details but interested readers can refer to a more detailed design document [15].

7.1 Components

Figure 14 shows the TinyOS component graph for RATS. *FRClockM* implements a free running 32-bit local clock based on the hardware Timer1 of the AVR microcontroller. *VClockC* maintains one (or multiple) virtual clock(s); it uses the model parameters (β_0, β_1) to map the local time given by *FRClockM* to the local time of the neighboring node. The estimation error and the model parameters are calculated and maintained by the module *LinearEstM*, which in turn uses the *SoftFloatC* component. The latter implements the double precision floating point arithmetic and is a TinyOS ported version of the SoftFloat C library [16]. *TSCommC* component is responsible for sending, receiving and timestamping the RATS packets. The *CC1000RadioC* component is a modified version of the corresponding component of the TinyOS source tree. *CC1000RadioC* uses the component *HTimerM* (hardware abstraction of the AVR hardware Timer 3) for accurate timing of the handler that duty-cycles the radio. Note that *VClockC* instead of *FRClockM* governs the duty-cycle of the radio. VClockC also provides an interface for the application writer to specify an alarm-like event based on a virtual clock.

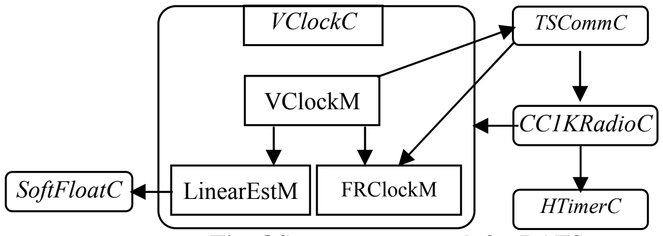


Figure 14. **TinyOS component graph for RATS**

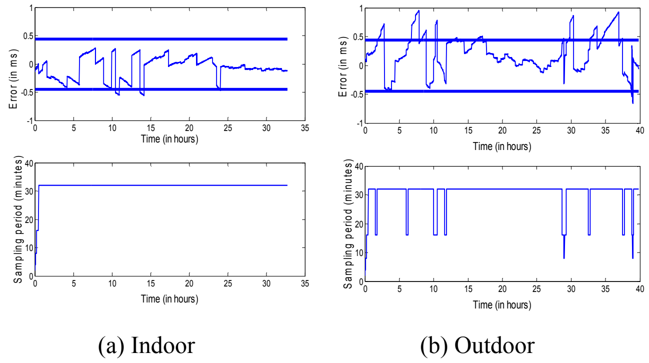
7.2 Troubleshooting

Implementing RATS on Mica2 motes involved numerous challenges, a few of which we highlight in this section. First, the single precision of the emulated floating-point arithmetic on a mote results in large quantization errors that impact accuracy of linear regression. The truncation error in terms of clock ticks is +/- 128, and depending on the resolution of the local clock, the error can vary from +/- 17µs to +/- 17.7ms. Double-precision operations require 64-bit arithmetic which is computationally intensive (linear regression and error estimation over a window of 16 samples takes 70ms and 120ms respectively), but it is significantly more accurate. As a proof of concept, we were able to replicate all the results obtained via MATLAB in Section 6. Since the model and error estimation needs to be done once every few minutes, this overhead may not be significant. For this reason we decided to port SoftFloat [16], a C implementation of the IEEE 754 Floating Point Standard, to TinyOS.

Second, the choice of the resolution of the timer offers a tradeoff. If we choose a very high resolution, the clock wraps around too often that limits the choice of the highest sampling period (Refer to [15] for the mathematical details). In our current implementation, we choose the resolution of the clock to be 8µs, which restricts the highest sampling period to be 32 minutes. As a result, the average sampling period numbers in the next section are smaller than their counterparts in the simulations.

7.3 Performance Evaluation

We ran several RATS experiments on motes, with varying error bounds and environmental conditions to gauge its performance in real settings. The duration of experiments varied from 20-48 hours. We use a scaling factor of 4 and the optimal time window of 8 minutes in all our experiments. Future work involves implementing the learning phase in TinyOS, as described in Section 5.3, to learn these parameters. Figure 15 shows an instance of such an experiment – it shows the sampling period at which RATS operate and the actual error between the motes in an indoor (inside an air conditioned lab) and an outdoor (open ground with temperature varying from 24-40^{0}C) setting with an error bound of 0.45ms. As can be observed from Figure 15, RATS is able to self-adapt; whenever the error crosses the desired threshold, it lowers the sampling period. Table 7 summarizes the results which are consistent with our earlier analysis.



(a) Indoor (b) Outdoor

Figure 15. **Real-time performance of RATS**

Table 7. RATS performance on motes

Setting	Error Bound (µs)	Av. Sampling period (minutes)	Faulty ratio(%)
Indoor	225	29.8	1.8%
Indoor	450*	31.7	3.7%
Indoor	700	31.34	0.0%
Outdoor	225	25.6	2.1%
Outdoor	450*	30.49	18.7%
Outdoor	700	31.75	6.4%

* Experiments of Figure 15

8. PREDICTIVE DUTY-CYCLING

Duty-cycling of the radio is a commonly used energy-management technique for event-response applications. The dominant factor that prevents optimal usage of the radio in real settings is clock uncertainty (as described in Section 1). All existing protocols incur overhead to handle this but are currently limited by the lack of approaches that accurately estimate this uncertainty. Our approach seeks to develop predictive techniques for long term timing synchronization so that energy wastage both due to time uncertainty and re-synchronization overhead is minimized. Our proposed predictive duty-cycling and synchronization architecture is shown in Figure 16.

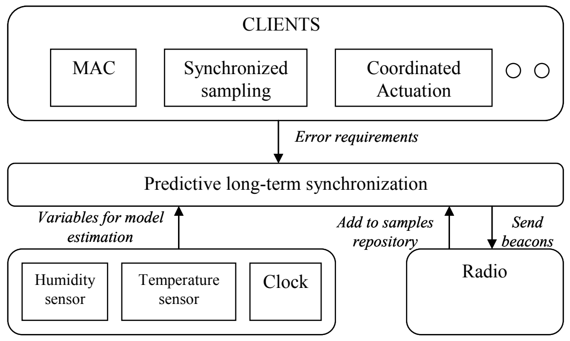


Figure 16. **Predictive Synchronization Architecture**[1]

At the core is the prediction engine (same as Figure 2). The MAC layer can use this engine to obtain a tight bound on the error with a neighboring node. This knowledge of uncertainty can be used to send a message with minimal radio overhead, while ensuring that the neighbor will be awake to receive the message. Note that other applications such as synchronized sampling and coordinated actuation can also use this engine simultaneously.

8.1 Uncertainty-driven BMAC

BMAC [2] is a carrier sense media access protocol for wireless sensor networks. It provides a flexible interface to obtain ultra low power operation, effective collision avoidance, and high channel utilization. To achieve low power operation, BMAC employs low-power listening which reduces energy cost of idle network listening by pushing an increased cost to the transmitter. In its current form, the transmitter uses a preamble with every packet that is guaranteed to handle the worst case time uncertainty between the transmitter and receiver. This results in a huge overhead. For example, with 11.5% and 2.22% duty-cycle a preamble size of 250 and 1212 bytes respectively is used for transmitting a 29 byte payload. Of these bytes, only four bytes are required if the sender and receiver were perfectly synchronized. Two of these bytes correspond to the minimum preamble size, and two bytes are normally added to take care of miscellaneous factors such as radio on/off time, software variations etc. Extra bytes over the minimum of 4 bytes are added by BMAC to take care of time uncertainty between the nodes. Specifically, addition of 1 byte allows a leverage of around 416µs. We demonstrate an easy integration of RATS and BMAC, termed as Uncertainty-driven BMAC (UBMAC) that can help reduce this energy overhead. UBMAC has two modes of operation:

Fixed preamble: In this proactive mode, UBMAC specifies an uncertainty bound to RATS, which achieves the bound through appropriate time synchronization beacon packets. UBMAC can now use a fixed preamble corresponding to this bound *irrespective* of the duty-cycle. For instance, if the uncertainty bound provided to RATS is 900us, UBMAC needs to use a fixed preamble size of ($4+\lfloor 900/416 \rfloor = 6$) 6 bytes. Such a mode is useful when there is frequent traffic in a sensor network since the cost of sending beacons can be amortized over the reduction in preamble size for many packets.

Variable preamble: This is a reactive mode in which no uncertainty bound is provided to RATS. Instead, when the application requests to transmit a packet, UBMAC uses the RATS module to estimate the uncertainty between the clocks of the sender and receiver. Based on this, the preamble size is decided on the fly. For example, if RATS predicts the time uncertainty between the clocks to be 2 ms, a preamble size of ($4+\lceil 2000/416 \rceil = 9$) 9 bytes is used. Such a mode is useful when traffic is infrequent and bursty. Note that some minimum time synchronization traffic (say 1packet/4hours) will always be present so that the sample repository does not become obsolete.

Note that the time synchronization control packets are always transmitted using the worst case preamble size. Thereby, even when the drift between the nodes suddenly changes, they will always be able to exchange RATS packets and hence, will be able to rectify the relative clock model between them.

8.2 Implementation

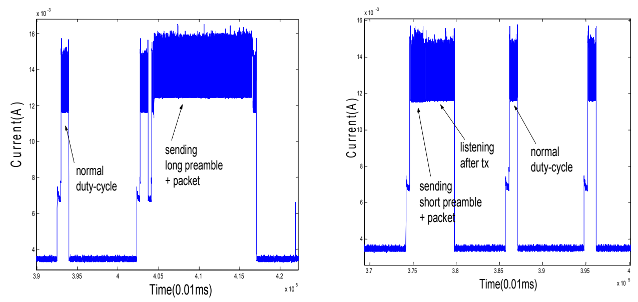
We have developed a prototype implementation of UBMAC on Mica2 motes. In this section, we present the results for the fixed preamble mode. We decided to use 6 bytes of preamble giving us a leverage of 2 bytes (or 832µs). The objective of RATS is to keep the nodes in sync within this error bound. Unlike the last section, we use a clock resolution of 32µs[2] so that the highest allowed sampling period of RATS can be 64 minutes.

The experimental set up consists of three motes each running UBMAC with 35.5% duty-cycle. We designated one node to be the parent and the other two to be the children nodes. Each child

[1] Our current clock model is agnostic about the variations in ambient conditions but in future we plan to develop more efficient clock models.

[2] Higher accuracy requirement of 832µs allows us to use a lower resolution clock.

maintains a virtual clock with respect to the parent, which is used for the duty-cycling of the radio. Each child also runs the predictive engine shown in Figure 2, calculates the desired sampling period and sends it back to the parent in a special RATS packet. The parent node chooses the minimum (maximum) of the two sampling periods (rates) at which to broadcast the timing synchronization packets. Besides this, the parent node sends an application level packet every 30s. As mentioned earlier, BMAC will use a preamble of 94 bytes (corresponding to 35.5% duty-cycle) for this packet, whereas UBMAC uses a preamble of 6 bytes. Figure 17 shows the current consumption in transmitting a packet, as observed on an oscilloscope, using BMAC and UBMAC respectively. This figure clearly highlights the energy gains brought about by the integration of RATS with BMAC.



(a) BMAC (b) UBMAC

Figure 17. **Current consumption**

We ran a 24hr experiment in an indoor setting and measured the packet drop rate at both the children nodes and an always-on (100% duty-cycle) receiver. A relative comparison with an always-on receiver helps us remove the bias of channel effects. Table 8 summarizes the results.

Table 8. Packet loss rates (Indoors)

First child	Second child	Always-on
2.2%	1.95%	1.92%

Note that the packet drop rates are comparable at all the three receivers. Furthermore, we observed that the packet drops were correlated, implying that they are mainly lost owing to the underlying channel characteristics. There were no bulk packet losses at any of the two children nodes, indicating that no packets are lost because of the time uncertainty between the children and the parent node. The average sampling period of RATS was about 54 minutes. Over the course of the complete experiment, the parent node would transmit 2800 packets in BMAC, each with a preamble size of 94 bytes. In our system all these packets were transmitted with a preamble size of 6 bytes. In addition 28 time synchronization packets are transmitted and as mentioned earlier, we use a preamble size of 94 bytes for transmitting these control packets. Overall, this converts to 3x improvement in transmit energy consumption. Clearly, these benefits will increase as the number of packets transmitted grows. We conducted a similar experiment in an outdoor setting with three children nodes. The packet loss rates are summarized in Table 9. The average sampling period of RATS was around 51 minutes. Both these experiments clearly highlight the efficacy of our approach and implementation. The time synchronization information can also be piggybacked with event data packets, whenever possible, further increasing the energy gains. Similar energy gains will also be observed at the receiver, as they would have to keep the radio in receive mode for a shorter duration.

Table 9. Packet loss rates (Outdoors)

First Child	Second Child	Third Child	Always-on
2.45%	3.1%	3.45%	2.95%

8.3 Energy Gains

Figure 19 demonstrates the relative transmit energy gains of UBMAC over BMAC. Note that the time synchronization overhead is also added in the energy consumption of UBMAC while obtaining this plot. The energy gains are calculated vis-à-vis the relative frequency of the occurrence of events and re-synchronization. For example, imagine a scenario where, on average an event packet has to be send after every 5 minutes. Using the results from previous sections, the average RATS re-synchronization period will be of the order of 45-55 minutes. Thereby, the relative frequency (x-axis in Figure 19) will be around 9-10. With a duty-cycle of 11.5% and 2.2% energy gains of 5x and 25x (y-axis in Figure 19) can be obtained. Figure 19 reveals two interesting observations. First, there exists a cut-off point in terms of the relative occurrence of events and re-synchronization (~5) beyond which the energy gains of UBMAC over BMAC almost becomes a constant. Second, the relative energy gains go on improving with shorter duty-cycle. This is because with shorter duty-cycle, BMAC uses a higher preamble size to take care of the worst-case uncertainty whereas UBMAC can keep on using the same preamble size of 6 bytes. This has a strong implication. The choice of the duty-cycle is typically governed by the latency constraints; lower the duty cycle, higher is the latency. For example, with 1% duty-cycle, the worst case latency can be 1.5s. Thereby, for applications with mild latency constraints, UBMAC can provide up to two orders of magnitude reduction in the transmit energy consumption at a node.

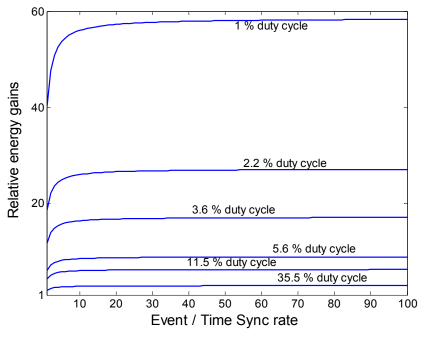


Figure 18. **Energy gains at the transmitter**

9. DISCUSSION

In this section we will briefly discuss some of the ongoing research efforts in the context of RATS.

Multihop performance: This paper focuses on long-term synchronization of adjacent nodes as this is the fundamental building block of network-wide long-term synchronization. A simple translation of virtual clocks can be used to synchronize nodes that are multiple hops away from each other. The problem arises when one of the local clocks at a node overflows. We have developed a comprehensive multihop extension of RATS that takes care of this wrapping around of local clocks. Due to space constraints we cannot cover the details but the interested readers can refer to the corresponding technical report [15]. We have developed a prototype implementation of this protocol in TinyOS; albeit it has not been comprehensively tested. Some of the initial results are encouraging. We were able to synchronize nodes across 3 hops within an accuracy of 0.5ms with each node running RATS at an average sampling period of around 51 minutes.

Atomic clocks: A hardware-based solution for removing the time uncertainty completely is to use a very stable atomic clock. Recent advances suggest the possibility of a low-power atomic clock that consumes 75mW [17]. This is still much too high for sensor nodes. To put it into perspective, the energy overhead of re-synchronizing the nodes in RATS will be comparable to the energy overhead of running a stable clock, if the power consumption of the clock is of the order of hundreds of nW.

10. CONCLUSIONS

We perform a detailed empirical study of long-term time synchronization across sensor nodes in many different environments including indoors and outdoors. A thorough empirical and analytical study reveals complex relationships between the sampling rate, window of past samples and the estimation scheme. We show that there is an optimal time window that will provide best estimation and error prediction. We provide analytical techniques to bound the prediction error and demonstrate an empirical verification of this bound. We use the measurement-based study to design a multiplicative increase, multiplicative decrease synchronization protocol. For a given user-defined synchronization error bound, this scheme consumes an order of magnitude less energy than the best possible periodic synchronization scheme. Our algorithm is adaptive to varying environmental conditions as well as unpredictable factors that impact clock precision. It can handle arbitrary time synchronization precision requirements from different applications without any customization and can be integrated with any lower-level timing synchronization approach.

This protocol paves the way for the development of uncertainty-driven duty-cycling techniques for sensor networks. We demonstrate a prototype implementation of such an approach by integrating our time synchronization scheme with BMAC. We show that this scheme achieves one to two orders of magnitude reduction in the transmit energy consumption at a node.

11. ACKNOWLEDGMENTS

This material is based on research supported in part by the Center for Embedded Networked Sensing (CENS), a NSF Science & Technology Centre, by the Office of Naval Research (ONR) under the AINS Program and by the DARPA under the PAC/C program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the CENS or the ONR.

12. REFERENCES

[1] Dutta, P., Grimmer, M., Arora A., Bibyk S., Culler D.. Design of a wireless sensor network platform for detecting rare, random, and ephemereal events. *Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS),* 2005.

[2] Polastre, J., Hill, J., Culler, D.. Versatile low power medium access for wireless sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys),* 2004.

[3] Ye, W., Heidemann, J., Estrin, D.. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the 21st International Conference of the IEEE Computer and Communications Societies (Infocom),* 2002.

[4] Dam, T. V., Langendoen, K.. An adaptive energy-efficient MAC protocol for wireless sensor network. In *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys),* 2003.

[5] Maroti, M., Kusy, B., Simon, G., Ledeczi, A.. The flooding time synchronization protocol. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys),* November 2004.

[6] Elson, J., Girod, L., Estrin D.. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[7] Ganeriwal, S., Kumar, R., Srivastava, M. B.. Timing-sync protocol for sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys),* Los Angeles, CA, November 2003.

[8] CENS habitat sensing group at James Reserve. http://www.jamesreserve.edu/

[9] Simon G., Maroti, M., Ledeczi, A., Balogh, G., Kusy, B., Nadas, A., Pap, G., Sallai, J., Frampton, K.. Sensor network-based counter sniper system. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys),* November 2004.

[10] Mills, D. L. Internet time synchronization: The network time protocol. *Global States and Time in Distributed Systems*, IEEE Computer Society Press, 1994.

[11] Sundararaman, B., Buy, U., Kshemkalyani, D.. Clock synchronization for wireless sensor networks: A Survey. *Ad-hoc Networks,* 3(3): 281-323, May 2005.

[12] Mills, D. L.. Adaptive hybrid clock discipline algorithm for the Network Time Protocol. *IEEE/ACM Transactions on Networking 6*, 5 (October 1998), 505-514.

[13] Rao, C. R.. *Linear statistical inference and its applications*. John Wiley & Sons, New York, 1973.

[14] Allan Variance. http://www.allanstime.com/AllanVariance.

[15] Tsiatsis, V., Sim, H., Ganeriwal, S., Ganesan, D., Srivastava, M. B.. *Implementation of rate adaptive time synchronization protocol in TinyOS*. Technical Report, NESL 2005.

[16] Softfloat. http://www.jhauser.us/arithmetic/SoftFloat.html

[17] Chip-scale vapor-cell atomic clocks at NIST.

http://www.bouldernist.gov/timefreq/ofm/smallclock/