# Management of Solar Harvested Energy in Actuation-based and Event-triggered Systems

Carlo Bergonzini
University of California San Diego
Computer Science and Engineering
cbergonz@ucsd.edu

Benjamin Lee
University of California San Diego
Computer Science and Engineering
bwlee@ucsd.edu

Joaquin Recas Piorno
Depto. ACyA
Facultad de Informática
Universidad Complutense de Madrid, SPAIN
jrecas@fis.ucm.es

Tajana Simunic Rosing
University of California San Diego
Computer Science and Engineering
tajana@ucsd.edu

## ABSTRACT

Today's sensing applications require low energy consumption while managing various types of tasks, ranging from processing and communication, to actuation and sensing. In this paper we develop a method to trade off energy availability, from both energy harvesting and storage, with the energy costs of the tasks the sensor node needs to accomplish. Our strategy consists of three main parts: a solar energy harvesting predictor, an energy recharge estimator, and an energy management scheme that takes into account task characteristics and external event triggers. Our results show that the solar energy predictor and energy recharge estimator give results within 10% of the actual values. The energy management scheme we implement is two times more efficient than one based on binary search in maximizing the overall energy consumption while delivering the needed performance.

## Categories and Subject Descriptors

C.3 [Special-Purpose and Application-Based Systems]: Realtime and embedded systems; C.4 [Computer Systems Organization]: Performance of Systems

## General Terms

Algorithms, Management, Performance

**Keywords:** Solar Energy Prediction, Energy Harvesting, Power Management, Embedded Systems, Actuation, Event-triggered, Sensors

## 1. INTRODUCTION

Recently proposed battery powered embedded sensor nodes, such as Motes [10], are designed to gather data and then periodically transmit it wirelessly over a network. They are usually very low power, so the type and amount of data collected tends to be simple (e.g. temperature or humidity). A lot of research has gone into minimizing the energy consumption due to communication since this is by far the largest contributor [13]. On the other hand, the power efficiency of processing has increased at a much faster rate than of communication, so more recent sensor nodes have been capable of more significant in-site computation. This has led to a development of a whole new set of applications that use more complex features such as actuation and significant data processing. Some examples include NIMS [15] and Robomote [18]. However, these systems need human maintenance in order to replace batteries. One way to extend it is by recharging batteries with energy harvesting. An example of such a sensor node is Shimmer [20]. There are many different types of harvesting technologies such as solar, vibration, wind, piezoelectric, and thermoelectric. However, out of all harvesting technologies, solar energy harvesting by far provides the most power per square centimeter as shown in [26, Table 1].

Clearly, the demands on battery lifetime can only grow with more sophisticated design. Thus, sensor nodes need to carefully consider strategies for spending energy on actuation, sensing, processing, and communication with the available energy from energy harvesting sources such as solar cells, and stored energy in batteries or supercapacitors. Managing the energy between these tasks is critical in order to ensure sufficient sensor node lifetime. Two main issues need to be solved in order to maximize the energy consumption of the sensor node:

1. The sensor node should use the extra energy available from energy harvesting sources once the batteries or supercapacitors are charged up. The energy can be used for executing tasks such as actuating, sensing, processing, and transmitting.
2. When energy harvesting is either not available or minimal, the sensor node still needs to be able to respond to the outside queries for data (event-based triggering). Therefore, adaptation is needed as the energy availability, and the outside demands change.

In this paper we propose an energy management algorithm that takes into account not only radio transmission but also actuation, sensing and complex data processing. Furthermore, we take into account both the available stored energy, as well as potentially available harvested energy from solar cells. To complement our energy management

scheme, we designed a new solar energy prediction algorithm gives highly accurate estimates on likely available energy over the next time period. It is computationally simple and has a small memory footprint thus allowing it to be implemented in many different solar harvesting platforms.

The rest of the paper is organized as follows. Section 2 gives an overview of the related work. Section 3 gives a brief overview of the system and then discusses the energy manager. In Section 4 we discuss our solar energy harvesting predictor and our energy recharge estimator. Section 5 shows our results and finally, in section 6, we conclude.

## 2. RELATED WORK

Energy management is a major topic in wireless sensor networks. Research aims to increase the lifetime of the sensor nodes while meeting the performance demands and wireless connectivity.

Most sensor nodes designed to date primarily gather data and transmit it to a host. They are either powered by battery, energy harvesting technology, or both. Some examples or basic sensor nodes include Duranode [12], Mica2 and T-Mote Sky[22], u-AMPs [5], Medusa MK-2 [4] and Prometheus [17]. These nodes use lower power microcontrollers to control passive sensors such as accelerometers, gyroscopes, thermo-sensors, light-sensors, and infrared sensors. They cannot perform heavy processing but rather focus more on transmitting gathered data to a host for future processing. Hence, they focus more on low power wireless transmission.

There are a few recently developed ystems that perform actuation, such as Networked Infomechanical System built at UCLA [15]. It uses actuation to move sensors for environmental data aggregation. It also performs complex statistical processing and uses wireless connectivity to transmit data and results among nodes using solar power. Another example is Robomote [18] which is a mobile mote that is used for localization and routing in mobile ad-hoc networks. It uses solar energy for mobility, processing, and transmission. A distributed network of Robomotes requires effective energy management for efficient movement to localize and communicate with other nodes [19]. Finally, SHiMmer is a wireless sensor node for structural health monitoring which uses actuation for more accurate data acquisition [20]. It also supports complex signal processing for data analysis and wireless transmission for communication.

Research in dynamic power management is fairly mature and can be applied to the design of embedded sensor nodes. Good overviews are given in [9][10][11]. Managing the power consumption of components in microsensor networks is analyzed in [13]. Scheduling techniques for low energy tasks with the goal of maintaining the desired level of performance are studied in [16]. Dynamic voltage scaling or dynamic frequency scaling algorithms are used to scale the voltage and frequency to consume less power while meeting timing requirements [5][12][14]. Dynamic power management techniques use either predictive or non predictive algorithms to determine whether or not to turn on or off a component [2][3][7]. Balancing tasks such as processing and communication is also widely researched in order to minimize the consumption of energy.

Management techniques for energy harvested devices can be realized in hardware and software [8]. Maximum power point tracking is used for solar panel energy harvesting in order to maximize the amount of energy obtained [6][11]. There have been many algorithms developed to manage the duty cycle of the application running on the node [1][2][3]. Most algorithms have been designed for the node to run in an energy neutral state in which it never needs to shut down to wait for the energy storage devices to recharge. To do that, a prediction algorithm computes the amount of energy remaining with respect to workload variations in order to meet scheduling deadlines. These algorithms also track energy consumption to adjust the duty cycle to maximize system performance. Less complex versions of algorithms could be executed in a depleted energy state versus more complex run in a maximum energy state to get maximum accuracy [14]. System reconfiguration during times of plentiful energy is used in order to maximize performance in relation to available energy. Finally, lazy scheduling is an optimal scheduling algorithm for energy harvesting sensor nodes [21] in which the execution times and deadlines of tasks are known a priori. It uses a scheduling strategy to meet as many deadlines as possible while taking into account the energy constraints of energy harvesting systems.

Our contribution in this paper is an algorithm that revolves around three parts. The first is a simple and efficient energy management scheme that manages the energy used for actuation, sensing and processing and increases the energy efficiency compared to similar methodologies. The second is a solar energy harvesting prediction algorithm that estimates the amount of energy that will be available in the future from a solar panel It takes into account not only the seasonal changes during the year like the previous methods, but also the current weather conditions. The third part describes a new scheme to estimate the amount of time needed to recharge the energy storage units so that more tasks can be run. These algorithms reach great efficiency while respecting memory and computational constraints.

## 3. ENERGY MANAGEMENT

In this section we describe an energy manager that is able to schedule tasks of different priority while taking into account their energy needs. We first look at the constraints of the system to see how they affect the energy manager so that it can operate with maximum efficiency. Then we analyze it to determine the average queue time for a task. Finally, we look

at different scenarios that the energy manager can operate in to show its flexibility for maximum performance.

## 3.1 System Constraints

In this work we assume that the sensor node has a way to both store and harvests energy, and use that energy to perform a combination of actuation, sensing, processing and communication tasks. Furthermore, in contrast to previous work [21], we do not assume hard deadlines, since many sensor node applications work in more flexible regimes. A block diagram of the class of sensor nodes we consider in this work is shown in Figure 1.

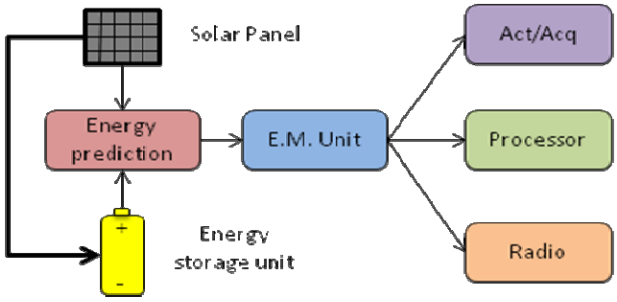


**Figure 1. System Description.**

In Figure 2, the sensor node is powered by a solar panel and an energy storage unit. The Energy Predictor uses both the solar panel value and the amount of energy available in the storage unit to determine how much energy can be harvested during a future period of time. The Energy Manager Unit then takes that into account and schedules tasks of type actuation/acquisition, process, and transmit accordingly.

We denote with $E_{Ti}$ the energy consumed by task $i$ and $E_{Hi}$ as the energy harvested during the execution of task $i$. In most of the systems previously studied [1][21], the amount of energy stored is comparable to the amount needed for processing, but in high computing and active sensing systems, the difference in energy storage capacity versus the energy needs can be quite large: $E_{Ti} >> E_{Hi} \ \forall \ i$. This particular constraint imposes an extremely low duty cycle in order to operate in an energy neutral state, and motivates us to use the extra available energy when the energy storage is at capacity. Also the current used by tasks in similar systems [20 Table 3] is usually much higher than the average current that a small solar panel for sensor nodes can provide. With this assumption we will consider that the system is not able to use the energy directly from the solar panel but rather must need some kind of energy storage unit. Clearly, the amount of energy we are able to store is limited by the capacity. Therefore, at time $t$, $E_A(t) \leq E_{max}$ where $E_A(t)$ represents the energy available at $t$ in the energy storage unit, and $E_{max}$ is the maximum energy it is able to store. Since we can only harvest energy during periods of light (assuming solar harvesting), we must have a minimum amount of energy saved that ensures the system will be alive during periods of darkness. We denote $E_{min}(T_{max})$ as the minimum amount of energy that the system needs for it to stay alive during a period $T_{max}$ when energy is scarce.

Next, we assume that sensor node needs to be able to respond to queries for data upon a trigger (event-triggering). node, so it needs to keep a minimum amount of energy in reserve. So, let $E_{trig}$ be the amount of energy necessary for the node to execute the necessary tasks when triggered. Thus, the new minimum energy required is $E_{Tmin}(T_{max}) = E_{min}(T_{max})+E_{trig}$.

The energy manager must maintain the system's energy level below $E_{max}$ and above $E_{min}$. If the amount of energy in the storage unit reaches $E_{max}$ and if more energy can be harvested, then energy should be spent by executing tasks, as depicted in Figure 2. Otherwise the extra available energy would be wasted since the system is unable to harvest more energy.

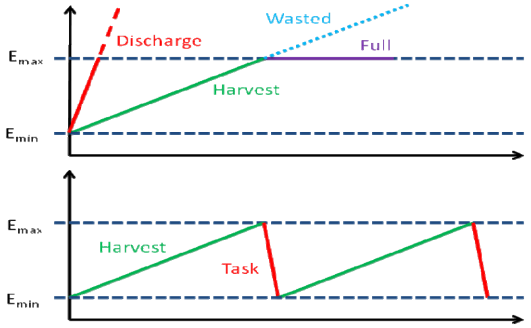


**Figure 2. Energy wasted vs. efficiently used.**

Also, the upper graph in figure 2 shows that the slope of the energy harvested is not comparable to the slope of the energy used by some high power tasks such as actuation and complex processing. In order to save as much energy as possible, the system must enter into a low power sleep state. Furthermore, if we can estimate how much energy is likely to be harvested over the next period of time, then we can determine when we can wake up the node to execute tasks. Our energy prediction unit uses actual and past values from the solar panel, as will be explained in section 4, to compute the time necessary to recharge the energy storage unit. The prediction is then used by our energy management system in order to schedule different types of tasks so that the system maximizes its performance and energy availability.

## 3.2 Energy Management

The block diagram of our energy management system is shown in Figure 4.

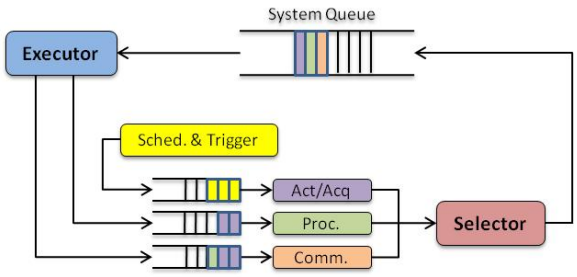


**Figure 3. Energy management unit.**

### *3.2.1 Energy Manager Description*

The system manages two subsections of the design the energy supply and sink. The energy supply consists of energy harvesting via solar cells, storage and the prediction algorithms related to those which are described in section 4. The energy sink section has three parts: actuation and sensing, processing and communication. The first task usually executed is the acquisition and sensing of data which may involve actuation, either by mobilization or by active sensing. Then, some performance processing may be necessary, for example using a DSP to compute a FFT to process the acquired data. Finally, the results may need to be transmitted with a wireless radio. Different types of tasks are added to the system and put into their respective queues depending on the current needs. The task selector selects tasks from the heads of the three queues and puts them into the system queue. It determines the execution order by considering priority, memory availability, and the application objectives of the sensor node.

The task executor executes the task at the head of the system queue if there is energy available. If not, then it will estimate the length of time it will take to recharge the energy storage units and enter the sleep state for the estimated time. After each task is executed, the executor determines which queue the results should enter next. For example, after executing an actuation/acquisition task, the executor can put the results into the process or transmit queue for either processing or transmitting respectively.

The energy management system we are considering faces the problem of how to spend the energy harvested in the correct way. Various tradeoffs may be selected and explored. For example, sometimes a system will decide to only gather then send data to save energy since more energy may be consumed during processing versus transmitting. On the one hand, this kind of approach may save energy, but the energy consumption during transmission may increase because the data length may be larger than expected. Another consideration that the scheduler must take into account is how to process the collected data. Sometimes it is better to process every time the system senses new data; other times processing more data together can provide more accurate results at the expense of memory usage.

In a node that performs actuation, processing and radio transmission, different patterns can be applied Let us call actuation A, processing P and transmission T.

- ***Act-Process-Send***: This is the most general pattern which sequentially performs: *A P T A P T A P T…*
- ***Act-Send***: If the consumption of the radio is much lower than the processing, then the pattern: *A T A T A T A T…* may be used. This is useful when we need real-time data streaming.
- ***Multiple Act***: The scheme: *A A A A P T…* could be used to actuate multiple times before processing. This could be useful in considering many samples together, for example taking the average of the set.
- ***Event Triggering Sending***: In some systems the radio may not always be available. In this case the pattern: *A P A P A P…* and eventually T when the radio becomes available.

Taking into account these different tradeoffs and patterns, we will introduce some possible scenarios to apply the energy management system. All scenarios consider wireless sensor nodes with high performance, active sensing, and computation powered by energy harvesting.

The first scenario we consider is an Electrocardiogram (EKG) monitoring system, using the reconfigurable embedded sensor node SunSPOT. Methods of EKG health monitoring involve having patients visit their doctors where they will then be monitored for a short period of time. However, such monitoring can not accurately assess the complete health of the patient, nor is it able to predict infrequent occurrences of abnormalities. For periodic EKG analysis, the Multiple Act profile could be used to acquire data for a certain length of time, process it by selecting an algorithm depending on the energy state of the node and finally transmit the results when a connection is available. The Act-Send profile may be used if real-time streaming data is needed.

A more complicated scenario that could exploit more possibilities of the energy manager would be for Structural Health Monitoring (SHM) applications. SHM is the process of observing a structure over time, identifying a damage sensitive feature in the observations and performing a statistical analysis of these features to determine the health of the observed structure. The Act-Process-Send profile may be used to periodically check the health of a structure. If a SHM node is placed in an inaccessible location, (the top of a bridge where a radio link may not always be available), or the node is used for the rapid assessment of structural state after an extreme event, (earthquake), the Event Triggering Sending profile can be used. Because, some SHM algorithms require high accuracy by actuating and measuring the same area

many times to compute an average, the Multiple Act profile should be used.

Now, we must determine how long tasks will spend waiting to be executed. In the next section, we will analyze this queueing system so that we can determine the average queue time for a given task.

### *3.2.2 Theoretical Analysis*

The model we use for the energy manager is similar to the Foreground-Background Scheduling Algorithm ($FB_N$) [25]. $FB_N$ algorithm can be described as a $M/G/1/\infty/PSFB_\infty$ queuing system. It takes into account priority and recycles tasks if not completed. However, tasks are not run to completion in each queue but rather receive a certain amount of processing time. If sufficient, then the task will depart but if not, then it is recycled to the tail of the next highest priority queue where it waits for more processing time.

In our system tasks are not moved to the next priority queue unless they have run to completion. In addition to the performance of the scheduler, we are also need to consider the energy consumed by tasks. Thus, in this section we adapt the $FB_N$ scheduling algorithm, to our particular energy management problem. We make the following assumptions:

- Arrivals are Poisson distributed according with a mean rate $\lambda$
- There is one server or executor (see Figure 3) which processes these tasks sequentially.
- Each queue type only holds tasks of that type, i.e. A queue only holds tasks of type A. The queue length is infinite. However, in actual implementation, the queue size is determined by how much memory is available in the system. If a task uses more memory than there is available, then it cannot be added to the queue.
- Each task $i$ has a generally distributed service time $z_i$, energy usage $e_i$, and type $k_i$ associated with it. ($i$ is a task index number)
- There are three priority levels. For s*i*$_m$plicity, in thi*s* derivation $w_e$ assume the following *o*rder of decreasing priority: actuate/sense (A), process (P), and transmit (T). Furthermore, we assume that only tasks of type A enter the system and the execution order is A → P → T. Thus, a **complete** task set begins with actuation/sensing, then processing, and finally transmission. The derivation can be generalized to any order/priority set.

Let *Q(z)* be the conditional-mean queueing time of a tagged task that has just entered the system and $z$ be the required execution time. Q(z) and $z$ vary depending on the task type that needs to be executed. The execution time $z$, is as follows:

$$z = \begin{cases} Z_A & \text{for a task of type A} \\ Z_A + Z_P & \text{for a task of type P} \\ Z_A + Z_P + Z_T & \text{for a task of type T} \end{cases} \quad (1)$$

Note that $Z_A$, $Z_P$, and $Z_T$ have a mean of $1/\mu_{z,k}$ where k is the task type. Applying the same principles to energy consumed, e is defined as follows:

$$e = \begin{cases} E_A & \text{for a task of type A} \\ E_A + E_P & \text{for a task of type P} \\ E_A + E_P + E_T & \text{for a task of type T} \end{cases} \quad (2)$$

Note that $E_A$, $E_P$, and $E_T$ have a mean of $h_{e,k}$.

*Q(z)* is determined as a function of the following:

$$Q(z) = T_{Wz'} + T_{Wz''} + kT_H \quad (3)$$

where

- $T_{Wz'} = E[W_z']$ (not to be confused with $E_i$ for energy) is the mean delay due to tasks found in the system due to prior admittance and/or higher priority which need to be executed before the tagged task.
- $T_{Wz''} = E[W_z'']$ is the mean delay due to the tasks which arrive to the system while the tagged task is in the system.
- $kT_H$ is the total time spent either recharging during the day or sleeping through the periods of darkness. We are unable to recharge while running because from section 3.1, we have the constraint $E_{Ti} >> E_{Hi} \ \forall i$

*Q(z)* changes depending on which type of the task needs to execute. We begin by analyzing Q(z) for the actuate/sense tasks and next apply our analysis to processing and transmission.

Sensing:

$T_{Wz'}$ can be found by analyzing an $M/G/1/\infty/FCFS$ system. Since actuation/sensing has the highest priority, the tasks in that queue are executed in a first-come-first-serve (FCFS) manner.

$$T_{S,Wz'} = \frac{\lambda E[z^2]}{2(1-\rho_z)^2} \text{ where } \rho_z = \frac{\lambda}{\mu_z} \quad (4)$$

$T_{Wz''}$ is unnecessary because tasks of type actuate/sense enter the sense queue which is the highest priority. Tasks that enter after our tagged task must wait for our this task to be executed.

To determine $T_H$, let us assume that the weather for a region is a Gaussian distribution with mean $\mu_H$ and $\sigma^2 = \sigma_H^2$. For certain regions, the mean may be small due to infrequent sunny days. Other regions may have a large $\sigma$ if the weather is highly variable. Thus, we let $T_H$ be a function of both $\mu_H$ and $\sigma_H$ that changes from region to region. $T_H$, however, is practically calculated in the next section.

At this point, we must determine the amount of energy in the system for task execution. Let $E_{min}$ be the minimum amount of energy necessary for the node to be able to respond to external requests. Let $E_{present}$ be the present energy, and let $E_R$ be the energy available for executing tasks. $E_R$ can then be calculated by:

$$E_R = E_{present} - E_{\min} \tag{5}$$

Now, let us denote $n$ as the number of the actuation tasks that have entered the system before our tagged task. Therefore, the total energy consumed by all actuation tasks, $E_{t\_act}$ before our tagged task would be: $E_{t\_act} = ne_A$

We can now determine the average queue time for a task of type A. Notice that if the total task energy for all queued actuation tasks is less that $E_R$, then the system will not have to sleep. However, if greater, then $Q_S(z)$ increases by $kT_H$., since we need to allow for the energy reserves to recharge before proceeding

$$Q_S(z) = \begin{cases} T_{A,Wz'} & \text{if } E_{t\_act} < E_R \\ T_{A,Wz'} + kT_H & \text{if } E_{t\_act} > E_R \end{cases} \tag{6}$$

where $k = E_{t\_act}/E_R$. Finally in section 3.1, we stated that $T_H >> T_R$. Therefore, we can see that $T_H >> Q(z)$. Thus, $Q_S(z)$ strongly depends on $T_H$ if $E_{t\_act} > E_R$. The throughput for actuate/sense tasks is simply $\lambda$.

Processing:

$T_{P,Wz'}$ is the same as $T_{S,Wz'}$ because we are looking for the average wait time of tasks already in the system. Tasks that already have entered the system are executed in a FCFS manner.

$T_{P,Wz''}$ can be determined by the following method. If the tagged task spends on average $T(z)$ seconds in the system, then on average there will be $\lambda T(z)$ new arrivals during this time. Each arrival will delay our tagged task by $1/\mu_z$ on average. Therefore, the average delay due to arrivals, $T_{Wz''}$ can be defined as:

$$T_{Wz''} = \lambda T(z)\frac{1}{\mu_z} \tag{7}$$

$E_{t\_process}$ is found in a similar way as before. It is $E_{t\_process} = n(e_A + e_P)$. Therefore, the average queue time for a task of type P is:

$$Q_P(z) = \begin{cases} T_{P,Wz'} + T_{P,Wz''} & \text{if } E_{t\_process} < E_R \\ T_{P,Wz'} + T_{P,Wz''} + kT_H & \text{if } E_{t\_process} > E_R \end{cases} \tag{8}$$

where $k = E_{t\_process}/E_R$. The throughput for processing is $\mu_S$.

Transmitting:

$E_T[W_z']$ and $E_T[W_z'']$ are the same as for processing since the transmission queue must consider tasks ahead of the tagged task and higher priority tasks to execute. $E_{t\_transmit}$ is found in a similar way as before. It is $E_{t\_transmit} = n(e_A + e_P + e_T)$. Therefore, the average queue time for a task of type T is:

$$Q_T(z) = \begin{cases} T_{T,Wz'} + T_{T,Wz''} & \text{if } E_{t\_transmit} < E_R \\ T_{T,Wz'} + T_{T,Wz''} + kT_H & \text{if } E_{t\_transmit} > E_R \end{cases} \tag{9}$$

where $k = E_{t\_transmit}/E_R$. The throughput for transmitting is $\mu_T$.

We can see that the average queue time strongly depends on $kT_H$ if the energy needed for all tasks is greater than the energy stored in the node. $T_H$ is highly variable and very difficult to determine probabilistically because it is based on the weather. A thorough analysis of how to practically determine $T_H$ is discussed in section 4.2. Equation 19 shows how to compute it if the solar energy can be predicted accurately. Generally, if it is a sunny day, then $T_H$ will be small thereby decreasing Q(z). If the day is cloudy, then $T_H$ will be large thereby increasing Q(z).

Also, in this analysis, we only discussed the A, P, T order. Similar analysis may also be done for the other priorities and task orders. However, the main difference will be in the task set because of the energy involved. $E_{t\_k}$ will change depending on how many of each type of task will need to be executed. If a complete task set is variable, then a distribution of the probabilities of how many of each type will be executed can be used in analyzing the system. Depending on how many tasks need to be executed, this may delay results of other tasks.

This analysis can be used to determine the average queue time for tasks in the A, P, T order. However, since this time ultimately depends on the harvesting time, the system must be able to predict how this. In the next section, we will describe a method accomplish this prediction.

## 4. ENERGY HARVESTING PREDICTION

We saw in the previous section that an accurate algorithm to predict the energy entering the system is necessary in order to optimize task scheduling. In this section we introduce a novel, low cost, and high performance algorithm for estimating the solar energy entering the system. The

algorithm is divided in two separate parts. The first part introduces a new strategy to predict the energy that is available to harvest from a solar panel. The second part uses this information to estimate the length of time needed to recharge the energy storage unit.

## 4.1 Solar Energy Prediction

In this section, we describe a simple and accurate method to predict the solar energy in the next time period. The challenge is that solar energy harvesting does not provide a reliable, consistent source of energy because of the suns diurnal cycles and the ever-changing weather conditions. The algorithm we present can accurately account for seasonal and weather changes with minimal overhead.

Our prediction algorithm takes a weighted sum of the actual energy collected at the present time with the values gathered during the past days from a matrix E of size DxN that stores N energy values for D days:

$$E_{(d,n+1)} = \alpha \cdot E_{(d,n)} + (1-\alpha) \cdot \frac{\sum_{i=1}^{D} E_{(i,n+1)}}{D} \quad (10)$$

Where E(i,j) is the energy in the matrix of the jth sample on the ith day and a is and alpha weighting factor that decides how important the actual value is.

This algorithm considers the seasonal changes by adapting to the change in the hour of sunrise and sunset and the difference in solar power between seasons. To improve the algorithm's accuracy, we also take into account the changing weather conditions on a specific day d, and use a weighting factor, GAP(d,n,K) where K is the number of past intervals considered. The factor K should be large enough to consider weather during the day but small enough to ignore periods of time that are not related to present conditions. For example, we do not want to consider night time when we are predicting energy available during the day. To compute the GAP factor, we define a vector $V^{d,n}$ :

$$\overline{V}^{d,n} = \left[ v_1^{d,n}, v_2^{d,n}, \ldots, v_K^{d,n} \right] \quad (11)$$

This vector contains the average solar energy at a certain time period n, over a number of days d:

$$v_k^{d,n} = E_{(d,n-k+1)} \Big/ \frac{\sum_{i=1}^{D} E_{(i,n-k+1)}}{D} \quad (12)$$

A weighting vector P, is defined as:

$$\overline{P} = \left[ p_1, p_2, \ldots, p_K \right] \quad (13)$$

Each element in vector P is inversely proportional to the distance from the present value $p_k$:

$$p_k = 1 - \frac{K-k+1}{K} \quad (14)$$

The weighting factor, GAP(d,n,K), for a day d at an interval n using K past values is computed using equation 15:

$$GAP_K^{d,n} = \overline{V}^{d,n}(K) \times \left( \overline{P}(K) \Big/ \sum \overline{P}(K) \right)^T \quad (15)$$

The GAP value is close to 1. If it is above 1, then there is more energy available because the actual weather is "better" than before, whereas if it is below 1, then there is less energy because the weather is "worse." Finally the predicted energy for the next time interval is computed using:

$$E_{(d,n+1)}^{GAP} = \alpha E_{(d,n)} + (1-\alpha) GAP_K^{d,n} \frac{\sum_{i=1}^{D} E_{(i,n+1)}}{D} \quad (16)$$

In experimental section we show that our algorithm is within 10% of measurement. In addition, we also present how to determine the appropriate size of matrix E and the coefficients K and α.

Once we have predicted the amount of energy likely to be available from the solar cell, we next need to find out how long it will take to maximize our energy storage. In this work we use supercapacitors for storage, but the ideas could be generalized to rechargeable batteries.

The rate at which the energy storage units recharge is directly dependent on the amount of solar energy available. Figure 4 shows the solar energy changing during the day in a semicircular manner. As the solar energy increase at the beginning of the day, the charge rate also increases and as the solar energy decreases toward the latter part of the day, the charge rate also decreases:

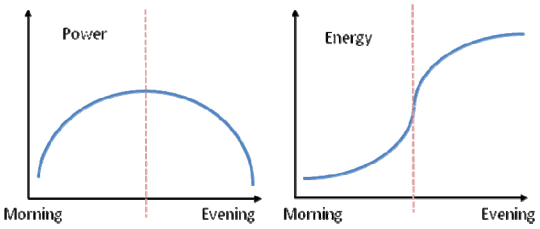


**Figure 4 Ideal power and energy during a day**

Our goal is to track how much energy is stored in the system and provide that information to our energy manager. It then can decide when and if to execute some tasks, and in what order. In order to estimate how much energy is going to be stored in the system, we assume that the solar energy behaves as a current source. If we use a super-capacitor as an energy storage unit, the voltage in the super capacitor increases linearly during static solar conditions, so the charging rate can be defined as:

$$R = \frac{dE}{dt} \quad (17)$$

To calibrate the maximum possible storage rate, we measure the voltage of the super capacitor during a sunny time period and use it to calculate the overall energy stored $E=CV^2/2$:

To determine the time the system needs to stay asleep in order to recharge, $T_{Sleep}$, we take the difference between the energy level at which we wish to wake up, $E_{wUp}$, and the present energy level, $E_{present}$.

$$T_{Sleep} = \frac{E_{wUp} - E_{Act}}{R} \quad (18)$$

We have now obtained the sleeping time for the system in order to charge the energy storage unit. It is calculated by using the solar energy harvesting predictor. If the sleeping time calculation is accurate, then the system will be able to harvest a maximum amount of energy from the solar panel and then use it effectively with the energy management strategies to maximize the performances, as will be shown in the next section.

## 5. RESULTS

In this section we evaluate the efficiency of our energy management framework and the quality of energy harvesting prediction with simulations in Matlab and measurements on an actual platform, SHiMmer [20]. SHiMmer is a great example of an energy-harvesting sensor node that performs a nice mix of tasks and does significant processing. It performs active sensing by actuation and sensing, high performance signal processing, and low power wireless communications by the use a ZigBee radio [27]. It is powered by solar energy harvesting and stores collected energy in super-capacitors. For actuation and acquisition SHiMmer uses 25 Joules and requires 8000 data points. It uses 10J and 5J respectively for processing and transmitting. For simulation, we have collected 45 days worth of solar data in various weather conditions, and have programmed a highly dynamic queueing system so that we can easily modify arrival rates, execution time, energy consumption and task priority.

### 5.1 Power Management

#### *5.1.1 System Queue Evolution*

In this section we see how the energy management system selects and executes tasks to efficiently use energy. We will use a simple Actuate-Process-Send profile, with a task energy consumption of 25, 10 and 5 Joules respectively based on the measurements on the SHiMmer sensor node. In this scenario ten actuation tasks are added to the system everyday to perform actuation, sensing, processing and transmission. These actuation tasks will be executed depending on energy and memory availability. After each one completes, it will be sent to the processing queue then the transmitting queue. By adding 10 tasks to the system each day, the system is usually unable to execute all tasks in the same day unless the whole day is sunny. Thus, we would be able to see what happens when tasks stay in the system for several days.

Figure 5 shows the evolution of the System Queue over 11 days. The dotted line is the task queue event counter which keeps track of the number of elements in the system task queue. The dots are the type of tasks that enter or exits the system queue where type 1 is actuation, type 2 is processing, and type 3 is transmitting. We can see in figure 5 how the number and type of tasks in the system queue changes during the day due to solar energy and memory availability. On most days there is not enough solar energy available to execute all tasks in the System Queue. However, on day 4, all tasks are executed (the system queue becomes empty) because enough energy was harvested. On day 8, though, only one task is executed because solar energy is scarce. Also, tasks can only be executed if there is enough memory. Actuating and sensing data on SHiMmer requires 8000 data points. If the manager were to only execute actuation tasks, then the memory would soon be full. To reduce memory usage, process and transmit tasks must be executed. The total amount of energy used by the tasks during 45 days is 11650J.

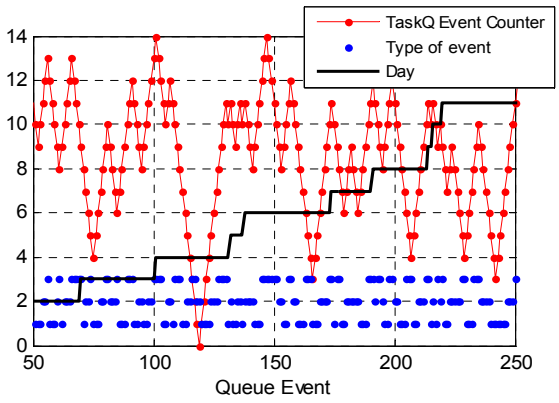


**Figure 5. System queue evolution. Change axis**

#### *5.1.2 Queue Waiting Times*

Now we will observe how long complete tasks sets of the pattern A, P, T take to execute in the energy manager. In figure 6, an acutation task arrives every 160 minutes and the tasks consume the same amount of energy as specified in section 5.1.1. Notice that the queue time increases between days 1 and 30 because there is insufficient solar energy. However, as more solar energy becomes available, the delay decreases as seen on days 35 to 65.

To compare these results to the theory discussed in section 3.2.2, let us determine that at a certain time, the state of the system is as follows: 5 actuation tasks, 2 process tasks, and 2 transmit tasks in their respective queues. The total energy required to execute all these tasks to completion using the

measurements given in 5.1.1, is 240J. At this time, a new actuation task enters the system. The amount of available energy for execution, if the super-capacitor is full, is 135 Joules, and $T_H$ is 120 minutes if the day is sunny. If the day is cloudy, then $T_H$ may even be greater than 24 hours! Given a sunny day and that the super-capacitor empty, then it will take a mean of 120 minutes to recharge to begin execution. Calculating k = 1, the number of recharge cycles, the total time before the new task is fully completed in the A, P, T order would be 240 minutes. However, during this time, another new task would arrive due to the average arrival rate of 3.2 tasks per day. Since priority is considered, 25 more Joules would be consumed before our taks is fully complete. This would mean that another recharge cycle of 120 minutes is necessary. Taking all things into consideration, the waiting time for our new task to fully complete would be 6 hours if in sunny conditions. We can see that this time would dramatically increase during cloudy days since the mean time is 5 hours.

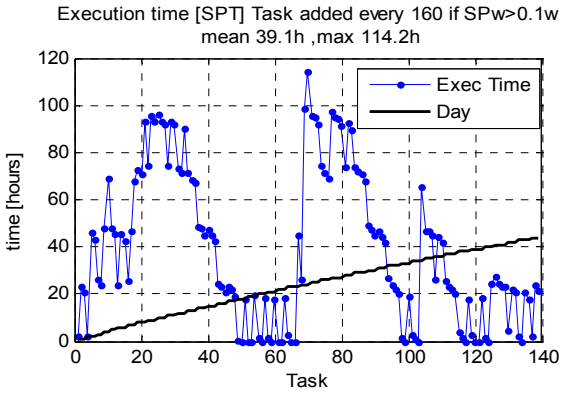


**Figure 6. Low arrival rate**

In Figure 7, we increase the arrival rate to 120 min per actuation task.

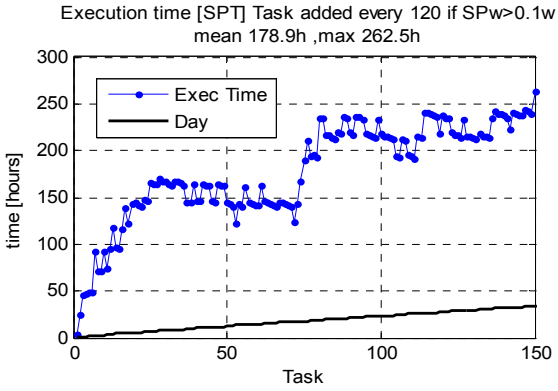


**Figure 7. High arrival rate**

Because there are cloudy days and the arrival rate is higher than before, the delay increases significantly to the point where the system will be unable to handle this many tasks. the delay will eventually go to infinite. If we compare the mean queue timeof 39 hours in figure 6, to the mean queue time of 178 hours in figure 7, we can see that by increasing the arrival rate just by 40 minutes, the queue time will eventually go to infinite. Thus, we can clearly see that the average queue time is solely base on the time it takes to recharge the super-capacitors and that task execution time has now become insignifcant.

### *5.1.3 Energy profiles*

We will now demonstrate the energy manager as it dynamically utilizes two different energy profiles. The first is a high energy profile that actuates several times, runs a complex signal analysis algorithm, and then transmits both data and results. This will use the original SHiMmer task energy values defined in section 5.3.1 and will be run in times when energy is abundant. The second is a low energy profile that actuates once, executes a simple processing algorithm, and only transmits the result. The energy consumption for this is 10J, 1J, and 0.5J respectively and will be used when energy is scarce. Figure 8 shows a period of four days, the first and the last being sunny.

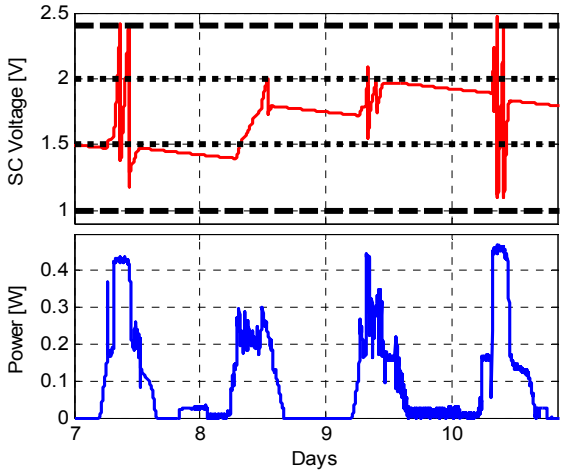


**Figure 8. Simulation with two energy profiles**

We can notice that the system uses the second profile during cloudy days and the first during sunny days. The red line is the super-capacitor's voltage, and the blue is the amount of power that the solar panel provides throughout the day. The long, broken lines are the upper and lower bounds of the super-capacitor's voltage during cloudy conditions and the short broken lines are the bounds during sunny conditions.

Figure 9 shows how long tasks wait in the queue while the energy manager switches between high and low energy profiles. We can see that the mean queue time is only 5.5 hours because the system is not executing energy-expensive tasks when solar energy is scarce. Longer wait times occur if solar energy is especially scarce for several days at a time, as seen by the maximum queue time of 45 hours. Therefore, by

utilizing different energy profiles in our energy manager, we can minimize queue time for energy-expensive tasks.

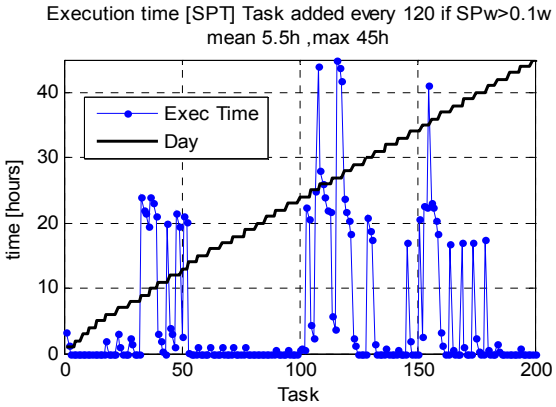


**Figure 9. High arrival rate with two profiles**

## 5.2 Solar Energy Harvesting Prediction

In previous sections we explained how an inaccurate prediction can affect the amount of energy harvested by wasting energy, consequently reducing the efficiency. In this section we determine the values of parameters used by our solar energy harvesting algorithm and study the accuracy.

### *5.2.1 Optimization*

In this section, we demonstrate how to minimize the error in prediction by calculating the optimal values for number of samples taken per day $N$, number of past days $D$, number of past values $K$, and weighting factor $\alpha$.

We used 45 days of solar data to simulate our algorithm and compared the predicted values of the next period with the measurements values. The error between the estimated and the actual value is given in a percentage by equation 20. However, only hours of sunlight are considered (those that have a minimum energy threshold of 20% of the maximum energy). Lower energy values, such as those during the night, affect this average error. However, the performance of the algorithm is unaffected when using all values.

$$Err = \frac{1}{N}\sum_{i=1}^{N} abs\left(1 - \frac{E_{Real}}{E_{Pred}}\right) \qquad (19)$$

When more samples are collected per day, the estimation of the next value is more precise at the cost of increased computation and memory energy consumption. Figure 11 shows that as the samples per day increase, the average error percentage decreases while the memory usage and the computational cost increase dramatically. On the other hand, too low of a sampling rate would not give us the sufficient data to calculate the rate of charge in the energy storage unit which would make the sensor node calibration difficult. We have found that for Shimmer the accurate and effective solution is with $N$=48 samples per day resulting in a duty cycle of 30 minutes and a memory usage of less than 100 bytes per day.

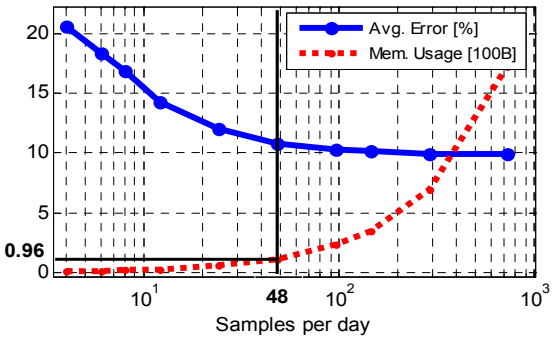


**Figure 10. Trade off between accuracy & memory usage**

Figure 11 shows the estimated error in prediction with as a function of the weighting factor α, and the number of days $D$, for a fixed number of past values $K$=6. Selecting a weighting factor α at 0.7 gives a minimal error independent of the number of days used. The resulting plot of the prediction error versus $D$ and $K$ is shown in Figure 12.

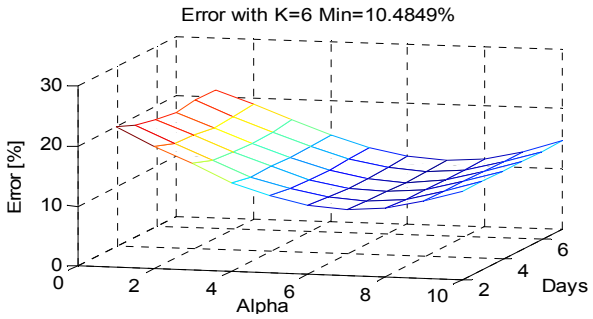


**Figure 11. Estimated error for $N$=48 and $K$=6**

If the number of past samples $K$ is above 5, then the error increases quickly because it takes into account too many past solar energy values. Since the number of past days does not influence the error as much as the number of past samples, we can use fewer days to lower the computational cost with great accuracy of estimation.

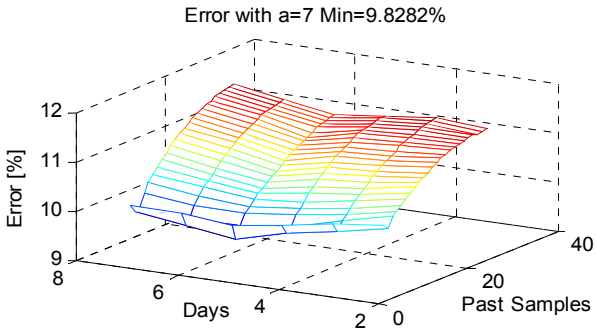


**Figure 12. Estimated error for $N$=48 and $\alpha$=7**

Given this analysis, we select the following parameter values to minimize the error: $D$=4 days, $N$=48 samples per day, $K$=3 past samples, and $\alpha$=0.7. Figure 14 shows four consecutive days of real solar samples and predicted values. Even for variable weather conditions, as in days 8 and 9, the predictor rapidly adapts to provide accurate results. Estimate error is shown in the figure below the actual power values. We found that an average error is only 9.8% over all 45 days of the collected data.

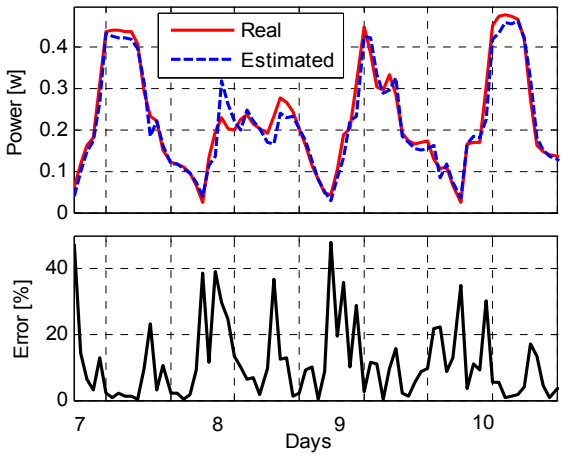


**Figure 13. Error evolution for $N$=48 and $\alpha$=7**

### *5.2.2 Comparison with EWMA*

We next compare our algorithm to commonly used Exponentially Weighted Moving Average (EWMA) [23]. It has been used for solar evolution prediction in [1]. $E_t(d)$ is the energy available by the solar panel at a time $t$ on the day $d$. $E_t^{EWMA}(d)$ is the predicted energy (shown in equation below) and ρ is the weighting factor. We obtained an optimized value of ρ=0.5 using the Matlab Optimization Toolbox on equation 20 [23].

$$E_t^{EWMA}(d+1) = \rho \cdot E_t(d) + (1-\rho) E_t^{EWMA}(d) \quad (20)$$

Figure 15 shows four consecutive days of the actual measurements in different weather conditions, and predicted values using our and EWMA predictors. The first and the third days correspond to cloudy conditions and the second and fourth are sunny. Since EWMA only uses values from previous days at the exact same time period, if the weather condition changes from one day to another, this method has a large error in prediction. On the other hand, our prediction performs much better since it not only uses the previous day's past values as at the same hour, but also the past values from the same day which contain the actual weather condition information. EWMA gives an average error of 28.6% compared to 9.8% obtained by our new algorithm.

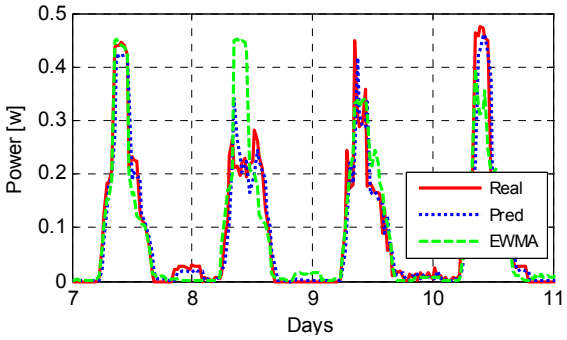


**Figure 14. Prediction Algorithm vs. EWMA**

## 5.3 Recharge Estimation

As we have seen from our energy manager, it is very important that the node wakes up at the right time so that no energy is wasted. In this section we evaluate our recharge estimator and compare it to a binary search algorithm.

First, we determine the rate of charge at any given solar intensity level. To do this, we program a calibration function that measures the voltage of the super-capacitor as it is changed during a small window of time. Since the curve is very close to linear, we estimate it using a linear equation as shown in Figure 15

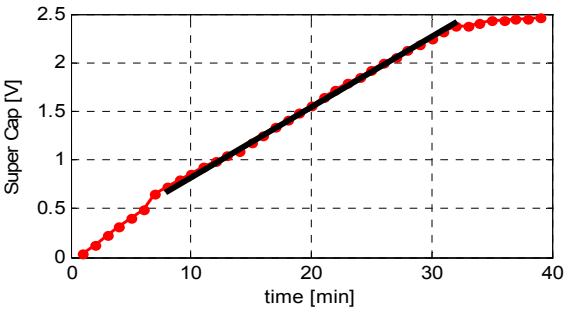


**Figure 15. 2.5V 50F super cap charging during noon.**

This is acceptable because most of the time, the super-capacitor is operating in the linear region. Also, the charge rate is higher here than in the regions where it is almost full (above 2.4V). If estimated linearly, then we can overestimate the voltage level which is beneficial because we avoid the region at which the super-capacitor recharges slowely.

The calibration algorithm measures the super-capacitor's voltage every minute for a period of ten minutes. Then it takes the difference between each two consecutive voltages to find the change in voltage per minute. Finally, an average is taken over the set of differences resulting in the charge rate at a measured solar intensity as shown in Eq 22. We ran the calibration algorithm every 30 minutes (corresponding to the duty cycle of the predictor) for three

days resulting in a table of solar intensities with corresponding recharge rates.

$$R = \sum_{i=0}^{N-1} \left( V_{SC,i+1} - V_{SC,i} \right) \Big/ N - 1 \qquad (21)$$

Figure 17 shows a sample of the data taken during the calibration algorithm showing the voltage of the 2.5V 50F super-capacitor fluctuate as time passes for different solar intensities.

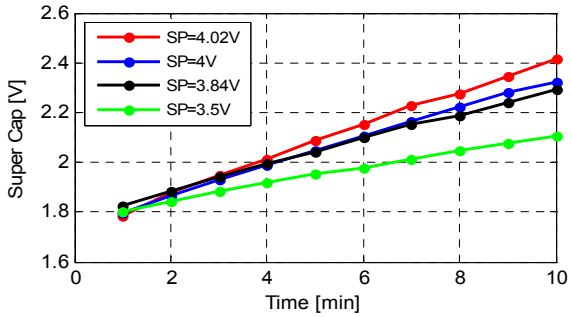


**Figure 16 S.Cap charge for different solar conditions.**

Figure 18 shows all the calibration experiments that we have done in order to estimate the rate of charge given a solar energy value.

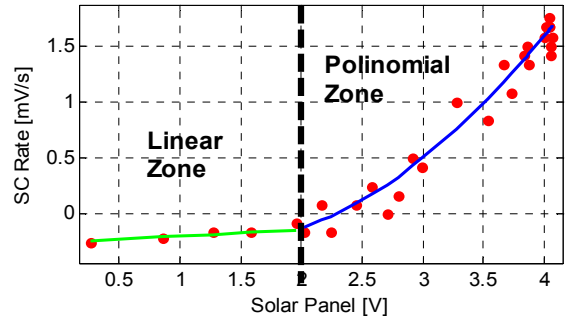


**Figure 17. Charging rate estimation.**

We can then relate all these data points with two equations. A second order polynomial $R_P$, is used to fit values greater than 2V and a linear equation $R_L$ is used for values between 0 and 2V. The fit is performed using least means squares. The results are shown in the equations below.

$$R_P = 0.23x^2 - 0.53x + 0.01 \qquad (22)$$

$$R_L = 0.06x - 0.25 \qquad (23)$$

By using these two equations, they can easily be adapted to many different microcontrollers without requiring any mathematical libraries and are computationally simple.

Using these estimates and the predictor equation 16, we determine the rate of charge during the next time period. Simulating this by using the solar data we have collected, we can estimate the recharge time by using equation 18. Figure 19 shows the super-capacitor charging and discharging according to this estimation.

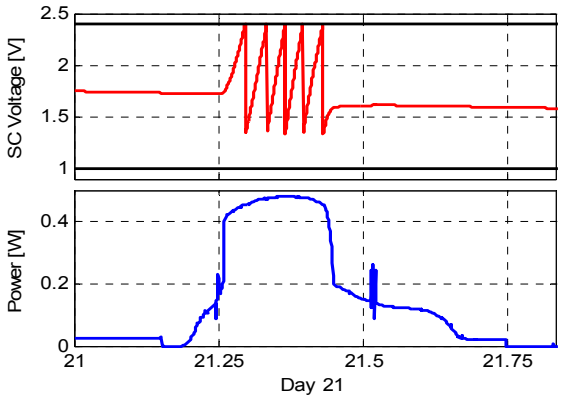


**Figure 18. Simulation of the energy harvesting prediction.**

Equation 19 is used to estimate the length of time to recharge the super-capacitor up to 2.4V. When it wakes up, tasks are executed to consume energy so that more energy may be harvested because if the super-capacitor is full, then energy could not be added. Our algorithm does a very good job as it very rarely wakes up after the super-capacitors are above 2.4V. Thus, not much energy is wasted in the "slow" charging zone or at full capacity. Figure 20 shows the super-capacitor charging during a sunny day

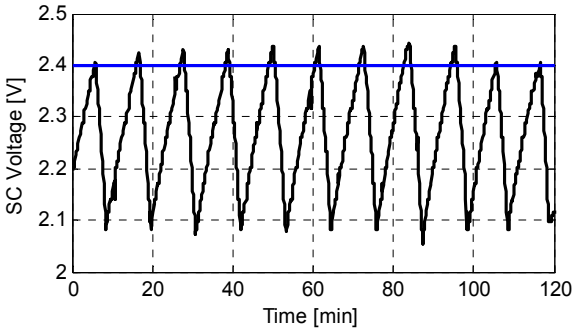


**Figure 19. Shimmer test of the wake up estimator.**

The algorithm is able to predict quite accurately when it reaches 2.4V and never reaches full capacity at 2.5V. This means that all available solar energy is harvested so that the node can use it.

To compare the recharge estimator algorithm, we used a simple binary search algorithm [24] to also determine the recharge time. Figure 21 shows the results of this algorithm as also implemented on SHiMmer. Notice that the node usually wakes up after the super-capacitor reaches 2.45V and sometimes even reaches 2.5V. This means that when it is at 2.5V, then it is unable to harvest anymore energy, thereby wasting available harvestable energy. Using our recharge estimator, we stay above 2.4V (region of slow recharge) for

12.6 minutes as opposed 36.6 minutes using binary search. This shows that our recharge estimator is 3 times more efficient in harvesting energy than binary search.

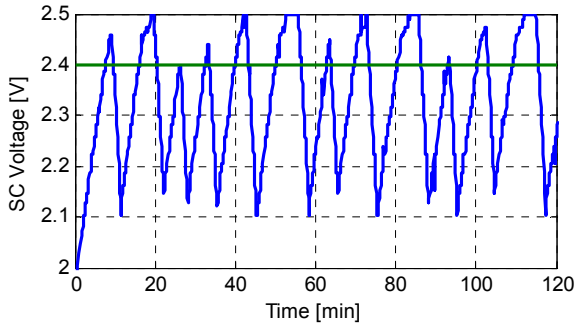


**Figure 20. Shimmer test of the binary search wake up.**

Figure 21 shows the system queue using the same profile as in section 5.1.1 except that the binary search recharge estimator is used to estimate how long it will take for the super-capacitors to recharge.

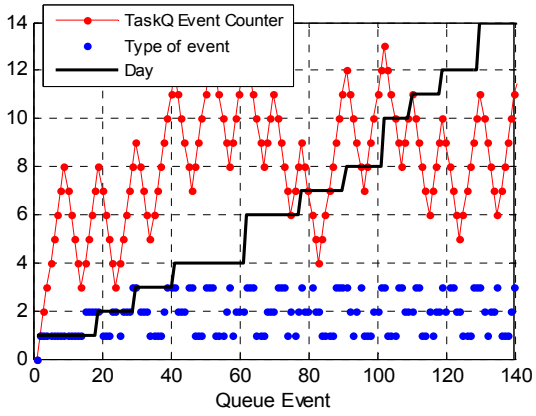


**Figure 21. System queue evolution with binary search**

We can see that in the same period of time as before, the number of executed tasks is less. Also, there are several days (5, 9, 13) when the system is unable to execute any tasks. This shows that if an imprecise algorithm is used to estimate the recharge time, then a non-optimal number of tasks will be executed. The total amount of energy used during the 45 days is 5225J, which is less than half of the amount of energy used when our estimator was utilized. This shows the accuracy of our predicator and estimator.

## 6. CONCLUSIONS

We have shown in this paper an energy manager that can schedule energy-intensive tasks. It requires an accurate solar energy predictor and an energy recharge estimator. As opposed to other scheduling work, we consider tasks that consume a significant amount of energy thereby causing the average queue time to be quite long. To offset this, we harvest as much energy as feasibly possible and run different energy profiles depending on energy availability.

Our solar energy harvesting predictor has shown to be very accurate and reliable in days of variable weather. This is extremely necessary so that our recharge estimator can accurately determine how long before the energy storage units are full. We have shown that if using an inaccurate predictor, the system' energy storage units will be unable to store more energy because they are full. By never reaching 100% capacity, we optimize the recharging rate for super-capacitor-based sensor platforms.

Finally, our energy manager is able to utilize these accurate estimations to determine when to execute tasks and what kind of tasks to execute depending on the available solar energy. We determined that since we are able to harvest a maximum amount of energy when available, our system will reach maximum performance even for tasks that consume significant amounts of energy.

## 7. REFERENCES

[1] Christopher M. Vigorito, Deepak Ganesan, and Andrew G. Barto, “Adaptive Control of Duty Cycling in Energy-Harvesting Wireless Sensor Networks”, in: in: Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON '07. 4th Annual IEEE Communications Society Conference on, 18-21 June 2007

[2] J. Hsu, S. Zahedi, A. Kansal, and M. B. Srivastava. “Adaptive duty cycling for energy harvesting systems”. Technical Report TR-UCLA-NESL-200604-02, Networked and Embedded Systems Laboratory, UCLA, April 2006.

[3] Vijay Raghunathan, Papleologos Spanos, Mani B. Srivastava. “Adaptive Power-Fidelity in Energy-Aware Wireless Embedded Systems”. 22nd IEEE Real-Time Systems Symposium (RTSS'01), p106, 2001

[4] Savvides, A. Srivastava, M.B. “A Distributed Computation Platform for Wireless Embedded Sensing”, Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02), p.220, September 16-18, 2002

[5] R. Min et al., “An architecture for a power aware distributed microsensor node”, in: Proceedings of the IEEE Workshop on signal processing systems (SIPS'00), October 2000

[6] D. Brunelli, S. Raggini, L. Benini , C. Moser and L. Thiele, “An efficient solar energy harvester for wireless sensor nodes.”, Submitted to International Symposium on Low Power Electronics and Design (ISLPED), Portland, 27-29 Aug, 2007.

[7] Aman Kansal , Mani B. Srivastava, “An environmental energy harvesting framework for sensor networks”,

Proceedings of the 2003 Int. Symp. Low power electronics & design, August 25-27, 2003, Seoul, Korea

[8] Igino Folcarelli , Alex Susu , Ties Kluter , Giovanni De Micheli , Andrea Acquaviva, "An opportunistic reconfiguration strategy for environmentally powered devices", Proceedings of the 3rd conference on Computing frontiers, May 03-05, 2006, Ischia, Italy

[9] Luca Benini , Alessandro Bogliolo , Giovanni De Micheli, "A survey of design techniques for system-level dynamic power management", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v.8 n.3, p.299-316, June 2000

[10] Chou, P.H., "Challenges on Low-Power Platform Design for Real-World Wireless Sensing Applications" in: VLSI Design, Automation and Test, 2006 International Symposium on April 2006.

[11] V. Raghunathan and P. H. Chou. "Design and power management of energy harvesting embedded systems". In ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design, pages 369--374, New York, NY, USA, 2006. ACM Press.

[12] Chulsung Park Qiang Xie Chou, P.H. Shinozuka, M. "DuraNode: Wireless Networked Sensor for Structural Health Monitoring" This paper appears in: Sensors, 2005 IEEE 30 Oct.-3 Nov. 2005

[13] V. Raghunathan, C. Schurgers, S. Park and M. Srivastava, "Energy-Aware Wireless Sensor Networks," IEEE Signal Proc., vol. 19, no. 2, pp. 40-50, Mar. 2002.

[14] C. Rusu, R. Melhem, and D. Mosse. "Multi-version scheduling in rechargeable energy-aware real-time systems". In: Proceedings of EuroMicro Conference on Real-Time Systems, 2003.

[15] Pon, R., Batalin, M., Gordon, J., Kansal, A., Liu, D., Rahimi, M., Shirachi, L., Yu, Y., Hansen, M., Kaiser, W., Srivastava, M., Sukhatme, G., Estrin, D.: "Networked infomechanical systems: a mobile embedded networked sensor platform". Information Processing in Sensor Networks, (2005)

[16] Aman Kansal , Dunny Potter , Mani B. Srivastava, "Performance aware tasking for environmentally powered sensor networks", Proceedings of the joint international conference on Measurement and modeling of computer systems, June 10-14, 2004, NY, USA

[17] Xiaofan Jiang , Joseph Polastre , David Culler, "Perpetual environmentally powered sensor networks", Proceedings of the 4th international symposium on Information processing in sensor networks, April 24-27, 2005, Los Angeles, California

[18] Gabriel T. Sibley, Mohammad H. Rahimi, and Gaurav S.Sukhatme. "Robomote: A Tiny Mobile Robot Platform for Large-Scale Ad-hoc Sensor Networks". In Proc. of the IEEE Intl. Conf. on Robotics and Autom. (ICRA), pages 1143–1148, Washington D.C., May 2002.

[19] M. Rahimi, H. Shah, G. S. Sukhatme, J. Heidemann, and D. Estrin. "Studying the feasibility of energy harvesting in a mobile sensor network". In IEEE Int'l Conference on Robotics and Automation, 2003.

[20] D. Musiani K. Lin T. Simunic Rosing. "Active Sensing Platform for Wireless Structural Health Monitoring". IPSN'07, April 25–27, 2007, Cambridge, MA, USA.

[21] Clemens Moser, Davide Brunelli, Lothar Thiele, Luca Benini "Lazy Scheduling for Energy Harvesting Sensor Nodes" 5th IFIP Working Conference on Distributed and Parallel Embedded Systems DIPES 2006, Braga, Portugal, October, 2006

[22] Mica Motes, Crossbow, http://www.xbow.com/Products/Wireless_Sensor_Networks.htm

[23] J. Stuart Hunter. "The Exponentially Weighted Moving Average" J Quality Technology, Vol. 18, No. 4, pp. 203–207, 1986.

[24] S. Singh and C.S. Raghavendra, PAMAS—Power Aware Multi-Access protocol with Signaling for ad hoc networks, ACM Comput. Commun. Rev. 28 (1998)

[25] L. Kleinrock, Queueing Systems Vol. II: Computer Applications, Wiley-Interscience, N.Y. 1976, Pp. 172

[26] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proc. 4th Int. Conf. on Information Processing in Sensor Networks*, pages 457–462, Apr. 2005

[27] ZigBee Radio, Digi, http://www.digi.com/products/wireless/zigbee-mesh/