Scheduling Recurring Tasks in Energy Harvesting Sensors

David Audet daudet@uvic.ca
Leandro Collares de Oliveira leco@uvic.ca
Neil MacMillan nrqm@uvic.ca
Dimitri Marinakis dmarinak@kinsolresearch.com
Kui Wu wkui@uvic.ca

***Abstract*—We consider the problem of periodic task scheduling in sensor nodes powered with energy harvesters. In particular, we focus on systems with stochastic energy sources such as solar panels, and we present two energy-aware scheduling algorithms that reduce the likelihood of task violations. Our algorithms, called *Smooth to Average Method* (STAM) and *Smooth to Full Utilization* (STFU), are static schedulers that do not require prescience of the incoming energy to operate effectively.**

***Index Terms*—Real-Time Scheduling, Recurring Tasks, Energy Harvester, Sensors**

I. Introduction

A wireless sensor network (WSN) consists of collaborating sensor nodes with capabilities of sensing, computation and communication [16]. Wireless sensor networks can be deployed for a plethora of purposes such as habitat monitoring [6], earthquake detection [18], or healthcare [15].

For ease of deployment, wireless sensor networks usually do not rely on existing infrastructure, and sensor nodes are typically battery powered. Therefore, the lifetime of these embedded devices is limited by the amount of energy that can be stored in the batteries. Furthermore, in many applications such as forest monitoring, the number of sensors and their locations might render the activity of replacing nodes' batteries infeasible or very costly [8]. There is a need for green solutions capable of powering sensor network applications with ambient energy.

To solve the above problem, intensive research has been conducted on energy harvesting as a way to extend the lifetime of wireless sensor networks. Several types of energy such as solar, eolic (wind), vibrational, and thermal among others can be scavenged from the surroundings of a sensor node to replenish its battery [14]. Promising as it may seem, energy harvesting poses new challenges to the scientific community [5]:

- Environmental energy sources behave stochastically, making the accurate prediction of incoming energy levels very difficult.
- Conventional task scheduling techniques were not designed for energy-limited scenarios and cannot deal properly with uncertainty in energy availability.

It has been pointed out that the traditional scheduling methods, such as Earliest Deadline First (EDF), may not work well under energy-limited conditions [8], and as such new algorithms such as the Lazy Scheduling Algorithm (LSA) have been proposed to "solve" the problem [8]. Although it has been theoretically proven that LSA is optimal, it requires an accurate prediction on the incoming energy source to operate well. Energy prediction, however, is non-trivial, and it is challenging to implement a suitably intelligent prediction algorithm on a typical sensor node due to the computational resources available on such a platform. Installing a pre-trained energy prediction model does not work either, because such a model depends on where and when the model was built and may not generalize when the sensors are deployed in different places and function over a long time period.

In this paper, we make contributions by proposing two new scheduling techniques, the Smooth to Average Method (STAM) and Smooth to Full Utilization (STFU), to handle energy uncertainty and deadline constraints without relying on any energy prediction model[1]. In our simulation-based evaluations, we consider solar energy scavenging through photovoltaic conversion, as it provides the highest power density of conventional environmental energy harvesting techniques [12].

In the remainder of the paper we first discuss related work on energy-aware scheduling, and formalize the problem of periodic real-time task scheduling. We then present two static, energy-aware scheduling algorithms and give an evaluation of their performance compared to related work through simulations. Finally, we give some concluding remarks.

II. Related Work

Important early work in real time scheduling by Liu and Layland [4] presented two classic scheduling algorithms, rate-monotonic priority assignment and deadline-driven scheduling, and assessed their performance based on processor utilization. Their work, however, did not consider energy constraints. Moser *et al.* [8], in more recent work, described energy-aware LSA scheduling and proved that it optimally deals with time and energy constraints in a system whose energy storage is replenished *predictably*. The suitability of this approach under realistic energy harvesting conditions, however, is unclear.

Research into energy-aware algorithms for sensor nodes is an active area. Kansal *et al.* [3] presented power management algorithms based on duty-cycling between active and low-power modes of sensor nodes with energy harvesting capabilities to achieve perennial operation at a desired performance level. Niyato *et al.* [10] investigated the impact of sleep and wake-up strategies on data communication among solar-powered nodes. These strategies are dependent on battery charge, solar radiation level and number of packets in the data queue. In [19], Vigorito *et al.* proposed an adaptive duty-cycling algorithm that ensures that power supplied to sensor nodes is kept within operational levels in several energy harvesting scenarios. The algorithm does not require previous information on the energy source dynamics and presents low

[1]Later in this paper we build an energy charging model for solar energy harvesting; however this model is purely for the purpose of performance comparison and in a real implementation such a model is not required.

computational demands. In [9], Moser *et al.* presented an adaptive power management model that can be customized to address different constraints and optimization objectives in energy harvesting systems, such as tradeoffs between communication and memory usage.

Predicting stochastic energy sources is non-trivial. Lu *et al.* [5] assessed three prediction techniques: regression analysis, moving average, and exponential smoothing. Their techniques met the requirements of high accuracy and low computation and memory demands imposed by a real-time energy harvesting embedded system. Recas *et al.* [13] employed the Weather-Conditioned Moving Average (WCMA) model, which adapts to seasonal changes in solar power harvesting as well as sudden weather changes. Moser *et al.* [8] introduced energy variability curves to predict the power provided by a harvesting unit. Energy predictions based on these curves are highly accurate when a sensor node's utilization is low. In [17] Şuşu *et al.* used a discrete-time Markov chain in which only transitions between consecutive states, representing energy levels generated by a solar panel, were allowed. This restriction hinges on the fact that abrupt changes in the energy provided in a time step are highly improbable. Niyato *et al.* [10] made use of a Markov chain model that takes into consideration the influence of clouds and wind on solar radiation intensity. Due to the limited computational and memory resources available on a typical sensor node, however, implementing suitably accurate dynamic energy prediction models appears challenging, making scheduling algorithms based on energy prediction impractical.

In [2], Jiang *et al.* present a hybrid implementation of environmentally powered sensors based on two supercapacitors and one rechargeable battery.

III. Problem Formulation

To model task execution in the sensor nodes, we assume the following:

- (A1) The requests for all tasks are periodic, with constant interval between requests. Such tasks are also called recurring tasks.
- (A2) Each request of a task has a hard deadline, which is defined as the time when the next request for the task arrives.
- (A3) A task has constant run-time. Run-time refers to the time to execute the task without interruption. We assume that the priority of tasks may change, but once being executed, a task cannot be interrupted.
- (A4) The task drains energy with a constant rate during its execution time[2].
- (A5) The tasks are independent in that requests for a given task do not depend on the initialization or the completion of requests for other tasks.
- (A6) The sensor node includes an energy harvester to supply power. It also has an energy storage module (capacitors or rechargeable batteries) with the maximum capacity of C.

[2]Energy consumption on sensor nodes largely depends on the operations of peripheral devices (e.g., sensors and wireless transmitters) associated with the task rather than executing code in the microprocessor.

We can denote a set of recurring tasks by $\{\tau_1, \tau_2, \ldots, \tau_n\}$, with each task represented by a tuple $\tau_i =< T_i, D_i, P_i >$, where T_i denotes the periodic interval time between requests for the task, D_i denotes the task's execution duration, and P_i denotes the task's energy consumption per time unit (*i.e.* power consumption). For a set of tasks scheduled according to some scheduling algorithm, we say that a *task violation* occurs at time t if the node's energy level drops to zero or t is the deadline of an unfulfilled request.

In this paper we consider the question of scheduling tasks to reduce the likelihood of task violations.

IV. New Energy-Aware Scheduling Algorithms

We have developed two new techniques that, when combined with known scheduling algorithms, reduce the likelihood of an energy violation while meeting task deadlines. We call the algorithms the *Smooth to Average Method* (STAM) and *Smooth to Full Utilization* (STFU).

A. Smooth to Average Method

First we illustrate a core concept, *virtual tasks*, used in our methods.

Definition 1: Given a task $\tau_i =< T_i, D_i, P_i >$ and a power threshold value $\bar{P}$, its equivalent *virtual task* is defined as the task $\bar{\tau}_i =< T_i, \bar{D}_i, \bar{P}_i >$. For STAM, $\bar{D}_i = \lceil D_i \times P_i/\bar{P} \rceil$ and $\bar{P}_i = D_i \times P_i/\bar{D}_i$. We use a power threshold $\bar{P} = \sum_i (P_i \times D_i/T_i)$.

To distinguish, we call the real task τ_i a *physical task* in the rest of the paper.

Remark 1: When the power threshold value $\bar{P}$ is larger than P_i, the task τ_i is the same as the task $\bar{\tau}_i$. When the power threshold value $\bar{P}$ is smaller than P_i, task τ_i and the virtual task $\bar{\tau}_i$ will consume the same amount of energy, but the virtual task $\bar{\tau}_i$ spreads over a longer execution time than task τ_i. Any scheduling algorithm that can schedule the virtual task without violating its deadline constraint will meet the deadline for the physical task as well.

Remark 2: The motivation of introducing virtual tasks is to smooth the energy consumption in the long run. If we make a schedule using the virtual tasks, and then execute the physical tasks according to that schedule, the system will implicitly wait for energy replenishment before running a request that consumes a large amount of energy. The waiting time is proportional to the energy amount consumed by the request.

Intuitively, it would be a good choice to smooth the energy consumption to the average power requirement of all tasks in a given task list. The STAM is designed for such a purpose. We generate a set of equivalent *virtual tasks* by increasing the duration of any task that uses greater than average power, thus *smoothing* each task to approximately the average power. In these virtual tasks, the total energy remains the same as that in the real tasks. Virtual tasks cannot be scheduled to run at the same time and are not preemptible. Once the virtual tasks are scheduled, the physical tasks are inserted at the end of the corresponding virtual task's timeslot. Thus a physical task that consumes high energy is guaranteed to run after an idle period during which energy is harvested, and so the likelihood decreases that the system will run out of energy when the task runs.

Algorithm 1 Generate STAM Task List

```
INPUT: $realTasks$ {list of [period, duration, power]}
INPUT: $N$ {number of tasks}
OUTPUT: $vTasks$ {same format as $realTasks$}
$\bar{P} \leftarrow mean(realTasks[:, 3])$
for $i = 1$ to $N$ do
  if $taskList[i, 3] > \bar{P}$ then
    $E \leftarrow realTasks[i, 2] \times realTasks[i, 3]$
    $\bar{D} \leftarrow \lceil E/\bar{P} \rceil$
    $\bar{P} \leftarrow \frac{E}{\bar{D}}$
    $vTasks[i, :] \leftarrow [taskList[i, 1] \quad \bar{D} \quad \bar{P}]$
  else
    $vTasks[i, :] \leftarrow taskList[i, :]$
  end if
end for
```

The STAM algorithm calculates the energy consumption of each task by multiplying its runtime by the task's power consumption. After taking the mean energy consumption across all of the tasks in the task list, each task is compared to this value and virtual tasks are generated accordingly. If the given task's power consumption is above the mean power value, the virtual duration is calculated by taking the ceiling of the energy of the task divided by the calculated power mean. This will extend the duration of the virtual task allowing the total energy consumed to be more evenly distributed across the duration of the task's runtime. If the given task's power consumption is below the calculated energy mean, the algorithm is unable to perform any smoothing and will use the unchanged physical task to generate a schedule. The pseudocode of generating STAM task list can be found in Algorithm 1.

B. Smooth to Full Utilization

A potential problem of STAM is that the virtual tasks may be spread across too long a duration, such that no scheduling is possible to meet the virtual tasks' deadline constraints. This may happen if some physical tasks require very high energy and thus the corresponding virtual tasks force the system to wait for a long time. To avoid this problem, we propose a different heuristic to smooth the energy consumption, called *Smooth to Full Utilization* (STFU). A *virtual task* generated by STFU is defined as the task $\bar{\tau}_i =< T_i, \bar{D}_i, \bar{P}_i >$ where $\bar{D}_i = \lceil D_i \times P_i/\bar{P} \rceil$ and $\bar{P}_i = D_i \times P_i/\bar{D}_i$.

The STFU algorithm is similar to STAM, but instead of smoothing all tasks to the average energy usage, STFU attempts to create a virtual task list with 100% *virtual utilization*[3], U_V. In other words, in a schedule generated from a virtual task list output by STFU, the likelihood of there being a virtual task scheduled at any arbitrary time is as close as possible to 100%.

Utilization U is defined in equation 1, where k is the number of tasks, D_i is the duration of the i^{th} task, and T_i is the period of the i^{th} task.

$$U = \sum_{i=1}^{k} \frac{D_i}{T_i} \tag{1}$$

[3]The CPU utilization calculated based on virtual tasks is called virtual utilization.

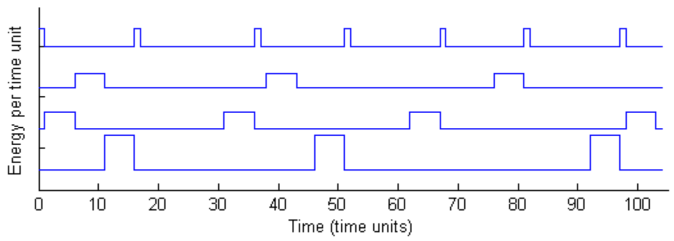


Fig. 1. Four tasks scheduled by EDF with no smoothing.

To generate a virtual task list with STFU, first each task is given a virtual duty cycle d_V representing what proportion of the total runtime will be allocated to the corresponding virtual task. The goal of STFU is to allocate more time to tasks that use greater energy, so that a high-energy task has more time to harvest energy before executing. For example, a task that uses 40% of the total energy consumed by tasks should be given a virtual duty cycle of $d_V = 40\%$. Virtual tasks cannot have a shorter duration than their real equivalents (otherwise the real task would not fit in the virtual task's timeslice), so if a task's physical duty cycle d is greater than d_V then it will be unchanged. The pseudocode of generating STFU task list could be found in Algorithm 2.

Algorithm 2 Generate STFU Task List

```
INPUT: $realTasks$ {list of $\bar{\tau}_i =< T_i, D_i, P_i >$}
INPUT: $N$ {number of tasks}
OUTPUT: $vTasks$ {same format as $realTasks$}
$E_{total} = 0$
for $i = 1$ to $N$ do
  $d_i \leftarrow D_i/T_i$
  $E_i \leftarrow d_i \times P_i$
  $E_{total} \leftarrow E_{total} + E_i$
end for
for $i = 1$ to $N$ do
  $d \leftarrow E_i/E_{total}$
  $d_V \leftarrow max(D_i, \lfloor T_i \times d \rfloor)$
  $P_V \leftarrow D_i \times P_i/d_V$
  $vTasks[i] \leftarrow [T_i \quad d_V \quad P_V]$
end for
```

Figure 1, Figure 2, and Figure 3 show four tasks scheduled by EDF, EDF with STAM, and EDF with STFU, respectively. Like in STAM, each real task with STFU smoothing is scheduled at the end of its virtual equivalent's time slice. The third task, which uses the most energy over a long run, is scheduled after a long period spent collecting energy. The second task uses very little energy overall, and is given just a short period to collect energy. For this task set, $U \approx 27\%$ and $U_V \approx 96\%$.

V. Evaluation Results

In this section, we present a simulation-based evaluation of the STAM and STFU schedulers.

A. Simulator Details

Our simulation framework includes a stochastic energy harvesting process, a task list generator, the scheduling processes,

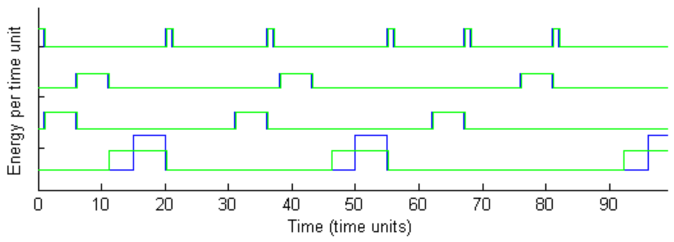


Fig. 2. Four tasks scheduled by EDF with STAM smoothing.

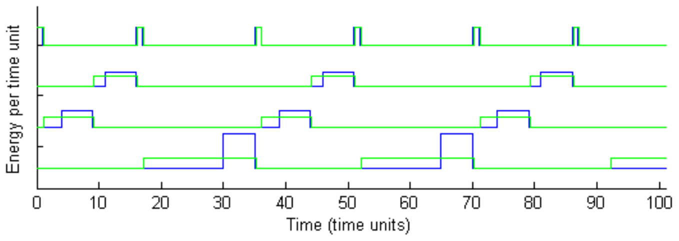


Fig. 3. Four tasks scheduled by EDF with STFU smoothing.

TABLE I
SOLAR PANEL ENERGY OUTPUT (BOLD VALUES ARE USED IN OUR WEATHER MODEL)

G ($\frac{W}{m^2}$)	I_c (A)	G ($\frac{W}{m^2}$)	I_c (A)
50	**0.190**	175	0.665
75	0.285	**200**	**0.760**
100	**0.380**	225	0.855
125	0.475	250	0.950
150	0.570	275	1.045

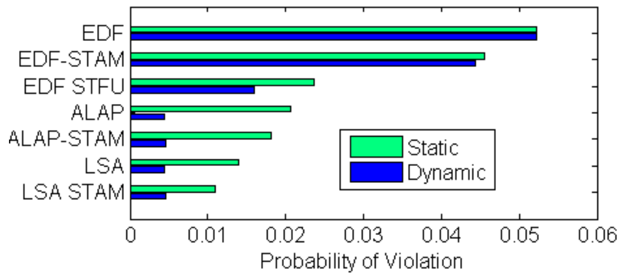


Fig. 5. Violations rate with and without dynamic task rescheduling; results averaged from 1000 task lists of less than 50% utilization.

and an execution process. We execute n simulations on one task list per run, and generate task lists for r runs. Each task list consists of k tasks.

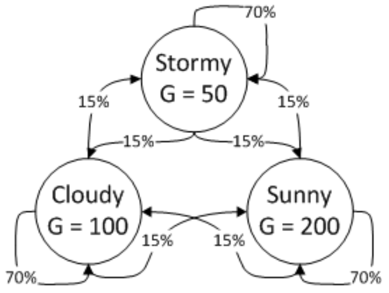
The tasks are generated with random periods, durations, and energy requirements. The periods and durations are distributed uniformly in discrete time steps, ranging respectively from 10 to 40 and from 1 to 4. The energy is half-normally distributed, and proportional to the task's period (*i.e.* a task requiring high energy is expected to run at a low frequency).

We consider only task lists that are *temporally schedulable*, which we define as having a CPU utilization U less than 100% (see Equation 1). For each task list we also generate a corresponding virtual task list using STAM and STFU.

We use the model of a photovoltaic energy harvester, which converts a solar irradiance G into a current I_c, as a stochastic energy source for our simulation. The energy drawn from the environment is modeled as a 3-state Markov chain ([11], [8]) representing three weather conditions (Figure 4). At each discrete time step during the simulation the Markov chain is updated, and the energy is generated and is added to an energy pool (*e.g.* a battery).

We generated a table of energy inputs to the system using the solar cell model provided by Simulink's SimElectronics toolkit, configured with values from [1]. The solar cell's current output with a battery load is related to its radiation input as determined by the linear function $I_c = 0.0038G$. We use the values of $G = 50, 100, 200(\frac{W}{m^2})$ to represent stormy, cloudy, and sunny weather conditions in our weather model, respectively.

Fig. 4. Markov Chain Weather Model.



The output current of the photovoltaic cell, I_c, is governed by a two-diode formula given in [7] and modeled by the Simulink model. The current flows into a battery, which we simulate using a linear model without relaxation effect. The battery capacity at time t, B_t is calculated using Equation 2 per [10].

$$B_t = B_{t-1} + I_c\Delta t - I_d\Delta t \tag{2}$$

where

B_{t-1}	is the battery capacity at the previous time step,
I_c	is the charge current due to solar harvesting during Δt, and
I_d	is the discharge current due to task execution during Δt.

We represent I_c and I_d as constant averages during the interval Δt. Furthermore, the battery is limited in capacity, such that if $B_t = B_{max}$ then any excess energy that is harvested is lost.

B. Simulation Results

For performance evaluation we used earliest-deadline-first (EDF) and as-late-as-possible (ALAP) scheduling to schedule real tasks, STAM virtual tasks, and STFU virtual tasks. In EDF, each task is scheduled as early as possible, in order of increasing deadline. In ALAP, tasks are scheduled at the latest time possible such that no task misses its deadline.

For comparison purposes, we implemented *dynamic* Lazy Scheduling Algorithm (LSA) as described by Moser *et al.* [8][4]. Additionally, we implemented a *static* LSA that incorporates

[4]We implement LSA-I as described in that paper

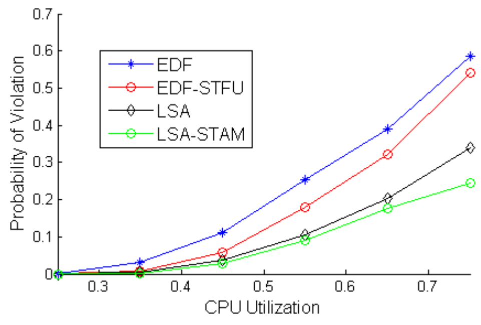


Fig. 6. Violation probability computed over 100 simulations, as a function of physical task utilization for static schedulers.

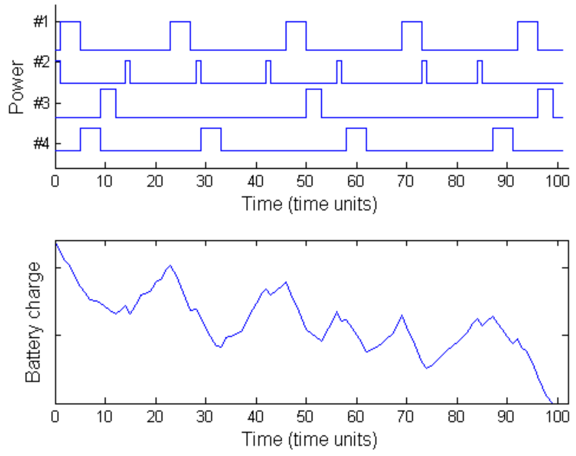


Fig. 7. Schedule (top) and battery charge level (bottom) during simulation of EDF algorithm.

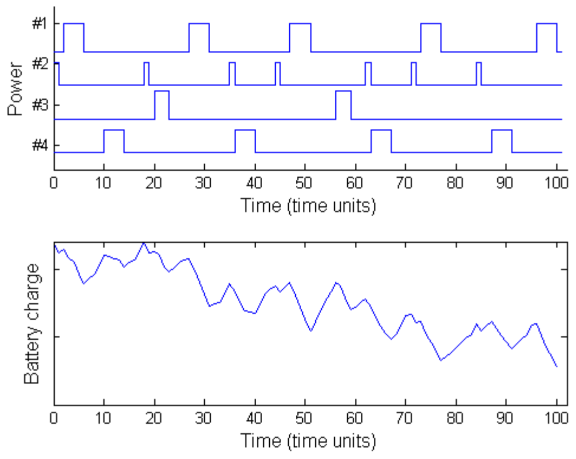


Fig. 8. Schedule (top) and battery charge level (bottom) during simulation of EDF algorithm on STFU tasks.

an off-line prediction of when the battery is at full capacity. To create a static LSA schedule we pre-process an ALAP schedule using our dynamic simulation routine, with a constant minimal energy input in place of the stochastic input. This constant input is the prediction we give to LSA. As a result, in LSA tasks will be statically rescheduled when the model can guarantee that the battery is at maximum capacity, *e.g.* at the start of the simulation before any tasks have run. Energy may still be wasted in the static-schedule, stochastic simulation model when the battery reaches its maximum capacity unexpectedly.

Although our focus is on static scheduling, we also implemented *dynamic* versions of each of the schedulers described above. In the dynamic version of each scheduler, the simulation detects in real time when the battery unexpectedly reaches full capacity and responds by immediately scheduling the next pending task.

In Figure 5 we show simulation results for statically scheduled systems, and for systems that support dynamic rescheduling. In this experiment we performed 1000 trials on different task lists with one trial consisting of 100 simulations for each scheduler on the task list. Each simulation covered a period of 100 time units, and if the battery charge dropped to 0 during the simulation we considered the simulation to generate a violation; hence producing an estimate of the probability of a violation in a 100 time unit trial. The static simulation routine executes each task as it appears in the input schedule. The dynamic simulator monitors the battery's energy level and, if the battery is at its maximum capacity (*i.e.* harvested energy cannot be stored), tries to re-schedule a task to run immediately.

As expected, EDF—the optimal periodic scheduling algorithm in systems with unlimited energy—results in the most violations. Schedules generated by applying the EDF scheduler to the virtual task lists generated by the STAM and STFU algorithms perform better. Scheduling STFU virtual tasks using EDF results in a significant improvement over plain EDF, approaching the performance of the more complex scheduling algorithms.

The ALAP and LSA static schedulers performed much better than the EDF-based algorithms (Figure 6). Results for scheduling STFU using ALAP were not reported as the performance of the scheduler approaches that of EDF for high CPU utilization task lists. As expected, using STAM virtual tasks to generate ALAP and LSA schedules improved the performance.

The dynamic versions of the scheduling algorithms resulted in very low violation rates because the model detected and responded to a full battery. Running ALAP and LSA on STAM tasks produced slightly worse results than on physical tasks. This is a result of the small idle time inserted before the physical tasks, which causes energy to be wasted when a STAM task is rescheduled. The dynamic versions of ALAP and LSA perform equally well, since our version of LSA is equivalent to ALAP when pre-processed.

Figures 7 to 10 show the change in battery charge level when simulating a particular task list scheduled with four different algorithms. Figures 7 and 8 show the relative performance of plain EDF and EDF performed on STFU tasks. The battery levels for the two simulations both have a downward trend at approximately the same rate, but in the STFU simulation the battery level rate of change is smoother. The large dip in charge that causes a violation at time 99 is "filtered out" by STFU, which gives the system a chance to collect more energy and recover. The smoothing effect is also demonstrated in Figures 9 and 10 for the static LSA algorithm with and

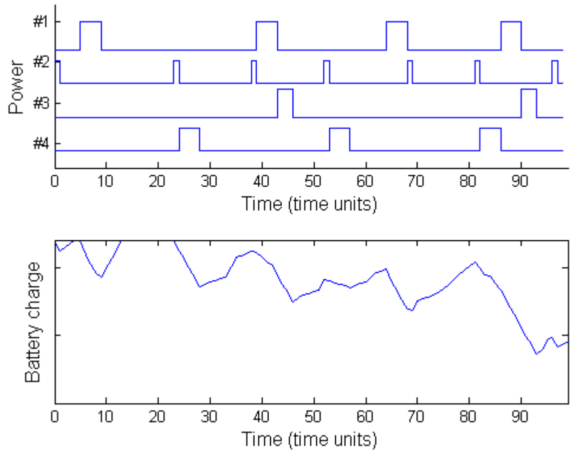


Fig. 9. Schedule (top) and battery charge level (bottom) during simulation of static LSA algorithm.

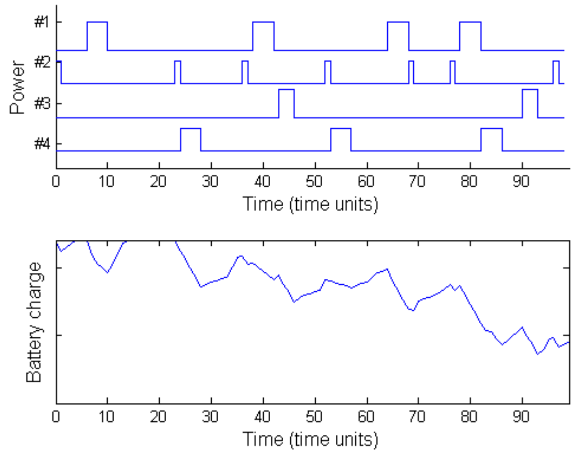


Fig. 10. Schedule (top) and battery charge level (bottom) during simulation of static LSA algorithm with STAM tasks.

without STAM.

VI. Conclusion

We have presented novel algorithms appropriate for the scheduling of hard real time periodic tasks for sensing devices powered through energy harvesting. Unlike most previous work in this area, our approach to task scheduling is static, and does not require a model of energy replenishment.

Experiments conducted through simulations that incorporate a *dynamic* energy replenishment model show that our scheduling algorithms perform better than classic, non-energy-aware, static scheduling algorithms. Furthermore, our static scheduling approaches perform at a level similar to the current state of the art, energy-aware scheduling algorithms that require prediction models such as proposed by Moser *et al.* [8].

As an on-going project, we have implemented a solar-powered wireless sensor node, and we are developing current monitoring to accurately measure energy charging and discharging rate. In future work, we will evaluate our approach by running a data acquisition application on our solar-powered sensor nodes.

Acknowledgements

We acknowledge the Natural Sciences and Engineering Research Council of Canada for their funding.

References

[1] F. González-Longatt. Model of photovoltaic module in matlab. *II CIBELEC*, 2006:1–5, 2006.

[2] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Proc. 4th International Symposium on Information Processing in Sensor Networks*, pages 463–468. IEEE, 2005.

[3] A. Kansal, J. Hsu, S. Zahedi, and M. Srivastava. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(4):32, 2007.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[5] J. Lu, S. Liu, Q. Wu, and Q. Qiu. Accurate modeling and prediction of energy availability in energy harvesting real-time embedded systems. In *International Green Computing Conference*, pages 469–476. IEEE, 2010.

[6] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97. ACM, 2002.

[7] M. Marwali, S. Shabidehpour, and M. Daneshdoost. Probalistic production costing for photovoltaics-utility sytems with battery storage. *IEEE Transactions on Energy Conversion*, 12.

[8] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems*, 37(3):233–260, 2007.

[9] C. Moser, L. Thiele, D. Brunelli, and L. Benini. Adaptive power management in energy harvesting systems. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pages 1–6. IEEE, 2007.

[10] D. Niyato, E. Hossain, and A. Fallahi. Sleep and wakeup strategies in solar-powered wireless sensor/mesh networks: Performance analysis and optimization. *IEEE Transactions on Mobile Computing*, pages 221–236, 2007.

[11] P. Poggi, G. Notton, M. Muselli, and A. Louche. Stochastic study of hourly total solar radiation in Corsica using a markov model. *International Journal of Climatology*, 20:1843–1860, 2000.

[12] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 457–462. IEEE, 2005.

[13] J. Recas, C. Bergonzini, D. Atienza, and T. Rosing. HOLLOWS: A power-aware task scheduler for energy harvesting sensor nodes. *Journal of Intelligent Material Systems and Structures*, 21:1300–1332, 2010.

[14] S. Roundy, D. Steingart, L. Frechette, P. Wright, and J. Rabaey. Power sources for wireless sensor networks. In *Wireless Sensor Networks*, volume 2920 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin / Heidelberg, 2004.

[15] S. Saadaoui and L. Wolf. Architecture concept of a wireless body area sensor network for health monitoring of elderly people. In *Proc. 4th Consumer Communications and Networking Conference*, pages 722–726. IEEE, 2007.

[16] S. Sudevalayam and P. Kulkarni. Energy harvesting sensor nodes: Survey and implications. *Communications Surveys & Tutorials*, pages 1–19, 2010.

[17] A. Susu, A. Acquaviva, D. Atienza, and G. De Micheli. Stochastic modeling and analysis for environmentally powered wireless sensor nodes. In *6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops, 2008.*, pages 125–134. IEEE, 2008.

[18] M. Suzuki, S. Saruwatari, N. Kurata, and H. Morikawa. A high-density earthquake monitoring system using wireless sensor networks. In *Proc. 5th international conference on embedded networked sensor systems*, pages 373–374. ACM, 2007.

[19] C. Vigorito, D. Ganesan, and A. Barto. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*, pages 21–30. IEEE, 2007.