Sparrow: An Energy Harvesting Wireless Sensor Node

Ioan Deaconu
Computer Science Department
University Politehnica Bucharest
Bucharest, Romania
ioan.deaconu@cti.pub.ro

Dan Ștefan Tudose
Computer Science Department
University Politehnica Bucharest
Bucharest, Romania
dan.tudose@cs.pub.ro

Abstract— **Powering a device using solar generated energy can be difficult, especially when that device is meant to function constantly over a long period of time. In this article we present an architecture for energy harvesting wireless sensor networks that can be used to develop solar powered applications. It will cover the hardware as well as the software requirements and specifications for a truly autonomous energy harvesting wireless sensor network. The hardware is composed of a new low power node designed to be a powerful development platform and an efficient energy harvesting module. The software is designed to efficiently use the stored energy by implementing a lightweight but powerful algorithm for scheduling data transmission.**

Keywords— wireless sensor networks; energy harvesting; scheduling; photovoltaic harvesting; dynamic scheduling

I. Introduction

A big problem encountered when developing an application for Wireless Sensor Networks is autonomy. Big batteries can be used to power the nodes, but because some can be deployed in locations difficult to reach, the simple task of changing the batteries becomes impossible.

The solution to this problem is powering the devices from alternative sources of energy, a process called energy harvesting. In recent years, energy harvesting has become more and more used in the field of Wireless Sensor Networks. There are plenty of alternative energy sources, such as solar cells, vibration absorption generators, wind mills, thermoelectric generators and others that can be used to power the nodes or charge their batteries in order to become autonomous.

While the energy source problem has a solution, another problem appears in the form of finding a method to store the generated energy. Most solutions employ conventional rechargeable batteries for this task, but the disadvantage of this approach is that current technology allows only for a limited number of recharge cycles, which dramatically reduces the application lifetime to a couple of years [1]. An alternative to the rechargeable battery are super capacitors, which offer a lifetime measured in decades or hundreds of thousands of recharge cycles, but are more expensive and still have a lower energy density than conventional rechargeable batteries.

In order to alleviate this energy density issue, sensor nodes can employ a scheduling algorithm which runs in conjunction with a duty cycling scheme. The main goal of the algorithm is to maintain functionality of the sensor node in situations of energy scarcity by actively adapting its transmission and sensing tasks and alternating them with periods of low-power sleep. This is achieved by dynamically varying the frequency with which the node performs various tasks or sends data.

In this article we will describe the architecture of an Energy Harvesting Wireless Sensor Network (EHWSN). We will present a new node and development platform, the Sparrow R, which is specifically designed for low power operations and the problems encountered when creating an EHWSN application. As the final part of the architecture, we developed an efficient but lightweight algorithm for efficiently using stored energy.

II. Related Work

Energy harvesting is a technique that has been the topic of multiple studies in conjunction with wireless sensor networks. The most promising technology for energy harvesting remains the use of photovoltaic panels, both in terms of cost and of energy conversion efficiency. However, this solution has the disadvantage of not providing a constant flow of energy, being subject to diurnal and seasonal variations. In this section we will present the current state of the art in solar powered EHWSN.

A. *Predicting generated energy*

Due to the fact that solar energy is not constant, in order to predict the generated energy, a history of past-days weather conditions or generated energy must be taken into account. The state-of-the- art algorithm for this is Weather-Conditioned Moving Average (WCMA) [2]. In order to predict the generated energy in the next hour, it needs to keep a history of generated energy for the past six days. The results of the algorithm are shown to be precise, with an average error of 9.8% in 45 days of testing.

Unfortunately, the algorithm requires the continuous measurement of the generated energy in order to have an exact history for the past number of days. Because of this, the algorithm is not feasible in applications where the sensor node has low energy storage capabilities and spends most of its operational lifetime in a low-power sleep state. A simpler solution for hardware as well as software must be found for those applications.

B. *Using stored energy*

In order to compensate for the variations in energy harvesting for photovoltaic systems, transfer speed scheduling algorithms have been developed.

The common approach to optimal packet scheduling is using a water-filling algorithm [3], [4], where the time is divided into slots and given a level of energy to be used in that slot. For better power optimization, this approach is modified into backward water-filling, directional water-filling and generalized iterative water-filling [5] for offline (deterministic) scenarios. Real world applications are stochastic, so in order to simulate online scenarios, Gaussian noise is added. The algorithms that can be used in this case are constant water level policy, energy adaptive water-filling [6] and time-energy adaptive water-filling [7]. Because these are complex algorithms, simpler ones have been attempted, such as fuzzy power management [8], where a table with predetermined levels is used as the main policy. This means that the above algorithms will work best in high power scenarios, where leakage currents of the sensor node electronics are small enough to be considered non-existent. Furthermore, the results of all the algorithms were obtained using simulated data with programs such as GreenCastalia [9]. These algorithms do not take into account parameters such as circuit leakage or shifts in power consumption due to temperature variation, which are important to a real-life deployment scenario for a sensor network and can cause significant discrepancies between the theoretical model and the actual implementation.

All the above algorithms need to measure how much energy the node is using when performing different tasks. As previously mentioned, this is not feasible in real conditions because precise current measurement can only be performed at a high sampling rate, which in turn consumes a large amount of energy.

Considering the presented problems, we developed a lightweight and efficient prediction algorithm that relies only on the voltage of the storage capacitor as a metric for energy production.

III. System Architecture

The majority of sensor nodes that are available as research or commercial platforms are built around 8-bit processor cores [10][11]. Few attempts have been made using the newer ARM Cortex-M3 32-bit architecture, the benefits of which are presented in this study [12], where a 4.6 throughput can be obtained compared to an 8-bit CPU over the same wireless network. Unfortunately, the downside is higher power consumption that can create difficulties for certain power supplies and energy sources. Even though the same average power consumption can be obtained for nodes using this ARM architecture, due to higher power consumption peaks the voltage of the power supply can drop to a level that is lower than the minimum voltage required for the node to function.

We propose a new approach in designing a wireless sensor node by using a state-of-the-art microcontroller, the ARM Cortex-M0+ 32-bit Atmel SAMR21. We will present its advantages compared to our previous sensor node platform, the Sparrow V4 [11], which is based on an 8-bit MCU, the Atmega128RFA1.

TABLE I. Comparison between Atmega128RFA1 and ATSAMR21

Criteria	Atmega128RFA1	ATSAMR21
Speed	16MHz	48MHz
CPU Architecture	AVR 8-bit	Cortex M0+ 32-bit
CPU Power	4.1mA	6.5mA
Flash	128kB	256kB
RAM	16kB	32kB
Flash Endurance	50000	150000
Rx Consumption	12.5mA	11.8mA
Tx Consumption	14.5mA@3.5mA	13.8mA@4dBm
Receiver Sensitivity	-100dBm	-101dBm
Tx Max Power	3.5dBm	4dBm
Package	QFN64	QFN48 or QFN32

TABLE II. Speed Comparison

Criteria	Atmega128RFA1	ATSAMR21	Total Advantage	Advantage per MHz
Integer Iterations	44890	403950	8.99	2.99
Branch Iterations	27782	93552	3.36	1.12
While(1) Iterations	191536	6693086	34.94	11.64

A. *Performance*

We present in Table 1 and 2 the main differences between the two MCUs, SAMR21 and ATmega128RFA1 [13].

Being a 32-bit architecture, even though SAMR21 requires 5.5mA compared to 4.1mA of the Atmega128RFA1, for simple 32-bit integer addition, the SAMR21 consumes only 49nJ per iteration while the 8-bit microcontroller consumes 274nJ, which is almost five times more. Considering performance figures, the SAMR21 was 9 times faster with 403950 iterations per second while Atmega128RFA1 managed only 44890 iterations.

Testing the performance of the branch predictor, revealed that the M0+ is only 12% better than the older 8-bit counterpart when running at the same speed, but due to the frequency difference, it ends up being 3.36 times faster.

The SAMR21 microcontroller is very similar to SAMD21 [14], which is used in the Arduino Zero boards. This allowed us to use an existing code-base for the development of the project. Even though the Arduino software is well designed, it was not designed with low power consumption in mind. We will describe some of the problems encountered in the current software stack.

TABLE III. Energy efficiency comparison

Criteria	Atmega128RFA1	ATSAMR21
Integer Iteration	274nJ	49nJ
Branch Iteration	442nJ	208nJ
While(1) Iteration	64nJ	2.9nJ

The first problem we noticed was that the Arduino Zero board had no sleep functionality implemented. The ideal idle current consumption should have been less than 5µA, but the actual current consumption of the board was around 350µA. Further tests revealed that the USB device was always initialized, which accounted for the extra 200µA. Almost all of the remaining 145µA came from default initializations of the pins as input pins and the clock generators which were never disabled at start-up.

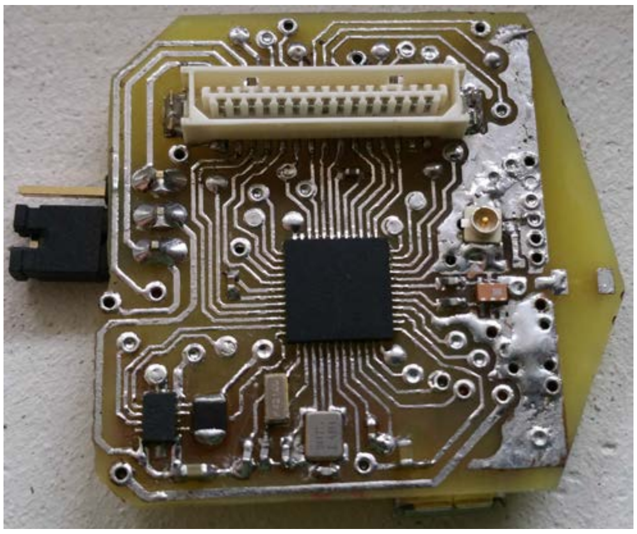
Fig. 1. The Sparrow R Wireless Sensor Node

We managed to a decrease the idle current consumption for the platform from 350µA to about 30µA @ 3.2V, but it is still far from ideal. Surprisingly, lowering the voltage from 3.2V to 1.8V leads to a decrease in sleep current consumption down to 3.3µA. When examining the power trace using a digital oscilloscope, we found that a very low frequency clock remains active, which at 3.2V has high spikes in power consumption.

After optimizing idle current consumption, we focused on the consumption in active mode. Even though it was specified in the official datasheet that this would be around 70µA/MHz @ 3.2V, or around 3.5mA@48MHz, we measured that the microcontroller actually drew 8mA@48MHz. However, we managed to reduce this figure to 5.5mA@48MHz, by implementing the clock optimizations presented below.

The first modification was to change the clock of the peripheral interfaces, instead of 48 MHz, we run them at 12 MHz. Also if peripherals are not used, we completely disable them. Due to this, we ran into problems related to SERCOM implementation, a generic module that handles USART, SPI and I2C. It was working on Arduino Zero, because the CPU and the BUS were configured to run at the same speed, but because of previous clock source modifications, the SERCOM did not set the correct speed. Also, there are 6 SERCOMs, and instead of enabling the clock for each one only when it is used, all of them were enabled, which lead to extra power consumption during run time.

B. *Hardware*

Because not all sensors are designed to run at 1.8V up to 3.2V, we need to be able to dynamically change the operating voltage of the node. We used the TPS62742, a step-down switching DC/DC converter with up to 90% efficiency, voltage selectable output from 1.8V to 3.2V in 200mV steps, 360nA quiescent current and a special MCU controllable load output, with push-pull transistors. In inactive state the load line is pulled to GND and when active is pulled to VCC. When active, it consumes 12µA, but compared to the controlled sensors, this should not be noticeable. Because this allows to completely power down the sensors, the ”stand-by” current consumption is 0µA and it also eliminates the previous design problem of a floating GND which allowed the sensors to be powered parasitically from the data pins even when disabled.

The node can be connected to an extension daughter board which fully respects the Arduino pinout. The advantage of this approach is that it allows to easily test and prototype new configurations in order to prepare the project in the shortest time possible. Also existing hardware designed for Arduino can work with this board, which increases the number of compatible hardware. In total, 20 I/O pins are available, pins that can be used for connecting sensors, either on the daughter board, or directly on a specially designed board.

A jumper can select whether the Node is powered from USB or from other 2.1V+ voltage supplies. Through the same jumper a power measuring device can be used to monitor the total power consumption. In case the DC/DC converter is not needed the node can use other power sources.

C. *Software*

We implemented new modules designed for low power such as sleep and power management, which can dynamically scale the running voltage of the node between 1.8V to 3.2V and enable or disable the voltage supply to the sensors. This allows the user to select which voltage is better required for a certain application. For example, some sensors must be powered at exactly 2.8V while others at 2.5V or lower. The application can dynamically switch between a low-voltage state, in which the processor is kept most of the time in order to conserve power to a higher voltage, which is required for switching on and communicating with the sensors.

The radio transceiver is very similar to an AT86RF233, but it is completely integrated into the microcontroller die. The module for RF was written and integrated in the core of the platform and was based on an Arduino library [15].

For timekeeping when sleeping, a Real-Time Clock (RTC) functionality was implemented. Besides keeping the time, the RTC provides alarm interrupts for a pre-configured date, which can be set to be triggered every minute, every hour, every day, every month, every year, or only once. Together with another peripheral named EventSys, periodic interrupts are provided and the interrupt interval can range from once every second up to 128 times per second, with increments of power base 2.

Also, in order to provide a minimum level of fault tolerance and resilience, a watchdog functionality was also implemented, in order to avoid code lock-up or hardware failure due to extreme environment conditions.

The node has native USB which allows for code upload and also serial communication interface over a Composite Device Class (CDC). Due to the fact that no extra components are needed, the same node can be easily configured to act as a gateway or as a leaf.

D. Algorithm

As seen in the previous section, state-of-the-art algorithms require measurement of the consumed energy in order to dynamically schedule transmission tasks. This is not, however, applicable in a real-world scenario, where numerous variables can generate errors in the estimated energy left in the storage element:

- Variations in temperature can alter the total stored energy
- Leakage currents can vary due to different subsystems of the node being turned on or off during normal operation
- Variations in parameters between supposed identical super-capacitors.

In order to simplify the algorithms for the user to deploy them faster in real EHWSN applications, we have implemented and tested a simple dynamic scheduling algorithm for the transferred data.

The algorithm can be considered to implement a basic water-filling policy, with one slot that is the current discharge period. This simplifies the implementation and reduces the computational requirements. It is designed to be run with an energy harvesting module that has a super capacitor as energy storage unit. Because of this, we simplified the predictor in order to obtain O(1) complexity.

Listing 1. Transfer speed scheduling algorithm

```
procedure ALGORITHM($voltage, time$)
  determine current state Charging/Discharging
  if changed from Charging to Discharging then
    Calculate new target time
    Calculate frequency based on total number of sent data
    $changedVoltage \leftarrow voltage$
  end if
  if current state is Discharging then
    if $voltage < MIN\ COMPUTE\ \ DELTA$ then
      $sendFreq \leftarrow sendFreq/2$
    end if
    $deltaVoltage \leftarrow changedVoltage - voltage$
    if $deltaVoltage > MIN\ \ VOLTAGE\ \ DELTA$ then
      Estimate the new remaining time
      Calculate remaining time until deadline
      $newSendFreq \leftarrow sendFreq * estimated * percentagePenalty/remaining$
      if $newSendFreq != sendFreq$ then
        Change the speed
        $changedVoltage \leftarrow voltage$
      end if
    end if
  end if
end procedure
```

The supercapacitor does not exhibit a linear discharge curve, but during the tests we have conducted, the error is small enough to allow us to estimate the remaining time without the requirement of measuring how much energy was stored in the capacitor and how much energy was consumed. Fig. 3 is an example of how the supercapacitor will discharge.

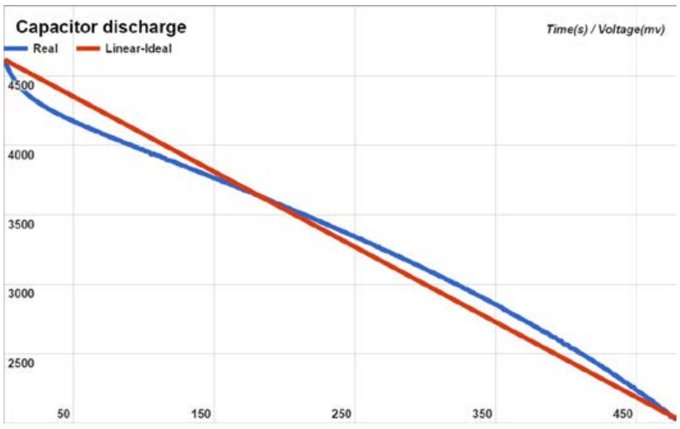

Fig. 2. 1F Electrolytic Double Layer capacitor discharge curve

The algorithm has a total time that it needs to respect, while trying to maximize the total number of packets sent. In order to achieve these two simple requirements, it dynamically estimates the remaining time and, according to the difference between the estimation and the remaining time, it keeps the same transmission frequency or alters it accordingly. The algorithm is very versatile, with many customization options, depending on the type of hardware and requirements of the application in which it is used.

The different parameters that can be modified are presented in the list below:

- DEFAULT TARGET TIME - the default deadline for the algorithm, dynamically adjusted after one iteration.
- TIME PERCENTAGE PENALTY - alters the estimated time by either reducing it or increasing it.
- TIME EXTRA REMAINING - when computing the new frequency, this is added to the remaining time in order to allow for a longer running period.
- SPEED DECREASE PENALTY - when the transmission speed is decreased, a penalty must be applied.
- MIN SEND FREQ - minimum packets per hour for the algorithm. Minimum value that can be set is one per hour.
- MAX SEND FREQ - maximum packets per hour for the algorithm. Maximum value that can be set is 3600 per hour.
- DEFAULT SEND FREQ - the default packets per hour before the algorithm stabilizes.

- MIN VOLTAGE COMPUTE - the minimum voltage under which the algorithm will stop adjusting the frequency
- MIN VOLTAGE - the minimum voltage considered by the algorithm when computing the estimated time and the new send frequency.
- MIN VOLTAGE DELTA - the delta voltage over which the algorithm will start to compute the estimated time and the new send frequency.

The algorithm will recalculate the new deadline every time, so in the case of diurnal variations in sun-rise and sun-set, the target time will be automatically adjusted. In order to compensate for the situation in which the sun will rise later, the algorithm always tries to keep the node alive for a longer time than the given deadline. The extra time the node can be kept alive can vary according to the deadline, from 10 minutes for a 4 hours deadline, to 1 hour for a 12 hour deadline.

IV. RESULTS

Our goal was to deliver a solution that could be deployed in an application as fast as possible, so instead of running software simulations, we decided to implement and test the algorithm on the new node Sparrow R. The algorithm is implemented in C, and compiled with gcc-arm-none-eabi-4.8.3-2014q1 on Arduino 1.6.4.

We selected two super-capacitors of 1F/5V rating, one built on Electrolytic Double Layer (EDLC) technology and other using Aerogel technology. We fully charged the capacitor and then let the algorithm decide how fast the data should be transmitted in order to reach the time deadline.

The only input needed is the current voltage of the capacitor, read using a voltage divider that is controlled by an N-MOS transistor in order to reduce the power dissipated by the divider. The node will run a task that simulates sensor readings and other processing through a delay of 100 ms. The network is configured as single-hop, and the data sent through the radio transceiver has a length of 45 bytes.

When beginning with a deadline of 4 hours, in the first run, the node manages to execute 1593 transmission tasks and to remain functional for a total time of 4 hours and 18 minutes. The second run, with a new deadline of 4 hours and 11 minutes, the node ran 1631 transmission tasks for a total time of 4 hours and 35 minutes. The results of the test are presented in Fig. 3, were we can see that both runs have similar frequency and discharge voltage curves.

With a longer, more realistic deadline of 8 hours, the node managed to transmit 785 times for a total of 8 hours and 14 minutes.

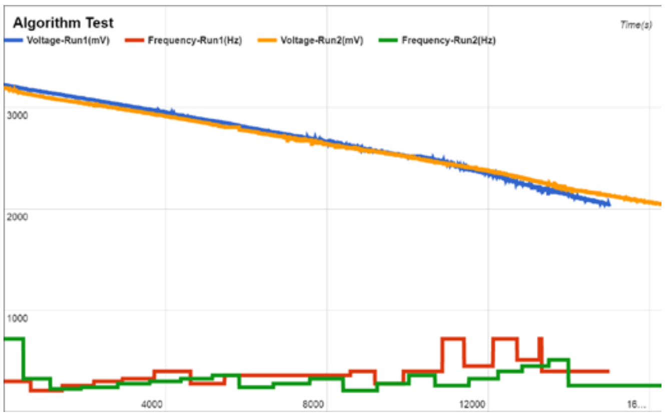

Fig. 3. Two runs of the algorithm with EDLC capacitor and 4 hours deadline

We can easily calculate the average power consumption of the node for the 4 hour and 8 hour deadline scenarios. In both scenarios, the supercapacitor that was powering the node discharged from an initial voltage of 3.3V to 2.1V, the voltage at which the node becomes powered-down. Therefore, the energy available to the node is the same, and is calculated in (1). Average power is deduced from this in equations (2) and (3).

$$E = E_i - E_f = CV_i^2/2 - CV_f^2/2 = 3.64J \tag{1}$$

$$P_{4h} = E / T_{4h} = 235\mu W \tag{2}$$

$$P_{8h} = E / T_{8h} = 122\mu W \tag{3}$$

When the total running time is doubled, the average power consumption is halved, proving that the algorithm is working. Unfortunately, the total number of send data is twice as small, which indicates that the combined power used by the node when sleeping and the leakage of the capacitor is starting to influence the total number of sent data. This is best shown by the difference between the energy used to send 1 frame in the 4 hours situation compared to 8 hours.

We ran into problems when a longer deadline of 12 hours was tested, mainly because the self-discharge of the capacitor combined with the idle current consumption of the node is high enough to waste more than 75% of the energy. This had a significant impact on the total number of executed tasks. The first run of the test send data 143 times and lasted for 13 hours and 10 minutes. The second run started with a more realistic speed of 10 task per hours instead of 600, but the total number of sent data dropped to 18 with a total duration of 13 hours and 15 minutes.

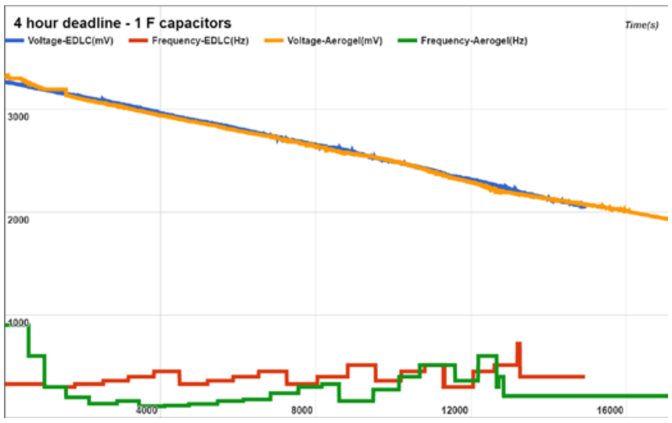

Fig. 4. 1F Aerogel versus EDLC capacitor discharge tests

The 12 hours test revealed that a 1F capacitor is not adequate and that a larger capacitor might be needed. Due to the fact that we tested the node with two types of capacitors, we wanted to determine if there was a difference between them. The result of the experiment is given in Fig. 4, which shows that the Aerogel capacitor had a lower voltage of 1950mV compared to the ELDC of 2050mV. However, the EDLC capacitor managed to transmit more data than the Aerogel, even when the algorithm was tweaked to take into consideration the lower working voltage of the Aerogel capacitor. The EDLC managed to send data 1642 times, while the Aerogel capacitor managed only 1254 data transfers.

V. CONCLUSIONS

The main goal of this article was to present the complete architecture of an Energy Harvesting Wireless Sensor Network. We have presented a new wireless sensor node designed to be a new development platform that can be used to bring new energy harvesting applications to life. One of those applications is using the nodes for indoor monitoring, where photovoltaic harvesting is less efficient and the system is dynamically varying the transmission speed in order to keep alive a node until the energy source (super-capacitor) can be recharged. We implemented an efficient duty-cycling algorithm for scheduling data transmission in an energy-efficient manner, which we deployed and tested on the node. The algorithm proved to be able to keep the node alive and efficiently use all stored harvested energy.

Because the algorithm only needs to read the voltage of the capacitor, the hardware requirements are very small and it can be easily deployed on existing hardware with little to no modification.

REFERENCES

[1] Laura Marie Feeney, Lars Andersson, Anders Lindgren, Stina Starborg, and Annika Ahlberg Tidblad. 2012. Using batteries wisely. In Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys '12). ACM, New York, NY, USA, 349-350. DOI=http://dx.doi.org/10.1145/2426656.2426702

[2] J. R. Piorno, C. Bergonzini, D. Atienza, and T. S. Rosing. Prediction and management in energy harvested wireless sensor nodes. In *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, pages 6–10. IEEE, 2009.

[3] J. Yang and S. Ulukus. Optimal packet scheduling in a multiple access channel with energy harvesting transmitters. *Communications and Networks, Journal of*, 14(2):140–150, 2012.

[4] L. Huang and M. J. Neely. Utility optimal scheduling in energy-harvesting networks. *IEEE/ACM Transactions on Networking (TON)*, 21(4):1117–1130, 2013.

[5] Z. Wang, V. Aggarwal, and X. Wang. Iterative dynamic water-filling for fading multiple-access channels with energy harvesting. *Selected Areas in Communications, IEEE Journal on*, 33(3):382–395, 2015.

[6] O. Ozel, J. Yang, and S. Ulukus. Optimal broadcast scheduling for an energy harvesting rechargeable transmitter with a finite capacity battery. *Wireless Communications, IEEE Transactions on*, 11(6):2193–2203, 2012.

[7] O. Ozel, K. Tutuncuoglu, J. Yang, S. Ulukus, and A. Yener. Transmission with energy harvesting nodes in fading wireless channels: Optimal poli- cies. *Selected Areas in Communications, IEEE Journal on*, 29(8):1732– 1743, 2011.

[8] F. A. Aoudia, M. Gautier, and O. Berder. Fuzzy power management for energy harvesting wireless sensor nodes. In *IEEE International Conference on Communications (ICC16)*, 2016.

[9] D. Benedetti, C. Petrioli, and D. Spenza. Greencastalia: an energy-harvesting-enabled framework for the castalia simulator. In *Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems*, page 7. ACM, 2013.

[10] S. R. N. S. Mridula Maurya. Current wireless sensor nodes (motes): Performance metrics and constraints. *International Journal of Advanced Research in Electronics and Communication Engineering*, 2(1), 2013.

[11] A. Voinescu, D. Tudose, and D. Dragomir. A lightweight, versatile gateway platform for wireless sensor networks. In *Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition*, pages 1–4. IEEE, 2013.

[12] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brünig. Opal: A multiradio platform for high throughput wireless sensor net- works. *Embedded Systems Letters, IEEE*, 3(4):121–124, 2011.

[13] Atmega128rfa1 datasheet. http://www.atmel.com/Images/doc8266.pdf

[14] Atmel ATSAMD21 datasheet. http://www.atmel.com/images/atmel-42181-sam-d21datasheet.pdf. Accessed: 2016-06-05.

[15] Arduino-at86rf233. https://github.com/msolters/arduino-at86rf233. Ac-cessed: 2016-06-05.