Experimental runtime power consumption measurements in Wireless Sensor Networks

Carlos Villavieja
Isaac Gelado
Nacho Navarro
Marisa Gil

* Dept. of Computer Architecture
Universitat Politecnica de Catalunya
{cvillavi,igelado,nacho,marisa}@ac.upc.edu

Loreto Mateu
Ferran Martorell
Francesc Moll

Dept. of Electronic Engineering
Universitat Politecnica de Catalunya
{mloreto,ferranm,moll}@eel.upc.edu

Abstract

Wireless Sensor Networks (WSN) are becoming an important field in ubiquitous computing. These tiny devices with sensors and communication capabilities form wireless networks that provide information ranging from environmental to human behavior. These systems are battery powered and its lastingness depends on battery lifetime since the replacement or re-charge of batteries are often not possible due to the location and high number of devices in a WSN. Some simulators allow developers to evaluate the power consumption of their applications, but there are still some energy consumption issues that simulators do not take into account. Our work proves necessary an experimental evaluation in order to get good power management analysis before deployment. In this paper, we have experimentally measured the energy consumption of some applications running under TinyOS. The energy consumption of our results provides hints for computer architects and programmers in order to design efficient power aware runtimes. The measurements show that different runtime abstractions and code optimizations affect the energy consumption up to 22% in the worst case.

1 Introduction

Power management is becoming very important in all digital systems but especially for those that are portable and ubiquitous. One kind of digital devices in which power management is very important are WSN motes. They form the building blocks of wireless sensor networks.

Motes are tiny battery-powered devices with radio links, which enable them to communicate and exchange data with other motes, and to self-organize into ad hoc networks. The decrease in energy consumption makes possible to increase battery life and even to power WSN by Energy Harvesting [1]. Motes applications usually collect information from the environment, do a little processing of this information, store it in memory, and send the information through other motes. Finally all the information is collected by a base mote.

A lot of work is being done in communication protocol levels [2], dynamic power management [3], probabilistic approaches in turning off CPU and communications hardware and scheduling algorithms [4] for low power devices.

There are also a large number of simulators to evaluate WSN applications but only PowerTOSIM [5] evaluates power consumption in...
motes. Under PowerTOSSIM, TinyOS components that reflect hardware peripherals are characterized in terms of power consumption in order to evaluate the whole system. The power consumption evaluation of the operating system abstractions is not very accurate.

Other operating systems or runtimes like SOS [6], CORMOS [7] or Contiki [8] are applying new techniques for WSN like code update at runtime or runtimes oriented to communication. They do not supply as many components as TinyOS but they provide better code update management or other communications management. Since [9] described TinyOS with a first hardware prototype, this paper is as far as we know the first power consumption evaluation of the runtime in terms of software components communications and events management in TinyOS.

In this paper, we focus in runtime architecture. Experimental energy consumption measurements have been realized to evaluate the runtime environment explained in Section 3.

Mica2 platform [10] has been chosen in order to evaluate TinyOS [11]. Mica2 is based on a single processor (ATmega 128L, MPR400CB). It is a low power microcontroller with 128kbytes program flash memory and a 512kbytes flash for data. Mica2 is formed by a main board with CPU, Radio Frequency (RF) communication hardware (Chipcon CC1000), memory, LEDs and a connector to the sensor board. This last board provides light, temperature, acceleration, sound, and magnetic field sensors.

To enlarge battery lifetime, motes take profit of low power working modes of CPU and Communications hardware. CPU and RF chips are the most consuming hardware components of wireless sensors. To achieve efficient power management, different CPU and RF power modes have to be combined. This paper does not try to provide power management techniques for the programmer but give directions to improve the operating system behavior.

Figure 1 and figure 2 show the importance of the runtime software for good power management. Both figures show the power consumption of a simple Blink Application where a one second timer is programmed to turn on and off a led. Figure 1 is built over TinyOS (version 1.1.1), while Figure 2 is built over SOS (version Beta 1.2.1).

Not only is the burst time for the application 62.86 ms greater in SOS, but also the sleep power consumption mode is 10 mW greater (as shown in figure 1 and figure 2). In the CPU active mode (high power level) the power consumed by TinyOS is 42 mW whereas the power consumed by SOS is 48 mW. In the CPU sleep mode (low power level) the power consumed by TinyOS is 30 mW while the power consumed by SOS is 38 mW. Table 1 shows energy consumption of the execution of Blink Application depending on the OS employed. Therefore, in this case the energy consumed has a 32.48% decrease if TinyOS is used instead of SOS.

<table>
<thead>
<tr>
<th>Energy Consumption</th>
<th>Blink App. in TinyOS</th>
<th>Blink App. in SOS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.23 mJ</td>
<td>15.15 mJ</td>
</tr>
</tbody>
</table>

Figure 1: Power consumption of Blink Application in TinyOS

Table 1: Energy consumption for SOS and TinyOS.

The goal of our measurements is to explore if there is room for improvements to consider Energy Harvesting a feasible possi-
bility. The rest of the paper is organized as follows. Section 2 describes the methodology used for the measurements. Section 3 describes TinyOS, the operating system we have evaluated and some power considerations. In Section 4, the results of our measurements are described and compared with PowerTOSSIM. Finally, in Section 5 some conclusions and future lines are given.

2 Methodology

In order to evaluate energy consumption in Mica2 we do voltage and current measurements using a digital oscilloscope (Tektronix TDS 784D) with a 10:1 passive probe (P6134C) and a differential probe (P6247) [12]. The data obtained from measurements has been analyzed in order to obtain energy consumption of the Mica2 platform. These measurements are done at different levels: software instructions, blocks or components from the application and the operating system. The consumption measurements are expressed in terms of energy instead of power (meaningless without duration). This is because energy permits a direct evaluation of battery lifetime.

Figure 3 shows the scheme of the consumption measurement of Mica2. By using an active probe, the voltage supplied to Mica2 is measured. A series resistor, $R_m$, with a value of 18 Ω has been placed between the DC voltage supply and the Mica2 to measure the voltage difference in its terminals with the differential probe. $C_{f1}$ and $C_{f2}$ are decoupling capacitors to reduce power supply noise. The energy consumed during a code execution is calculated as:

$$E = \int_{code} v(t) \frac{vR_m}{R_m}$$

Large number of code tests must be done in order to get accurate measurements and to know the overheads in the operating system and in the application. Evaluation of alternatives in runtime abstractions implies a small number of instructions and therefore the decrease of energy is very small. Thus, the code evaluated is repeated a large number of times to ensure that the energy decrement is appreciable and to give realistic data about the percentage of decrement in energy consumption.

To mark execution entry and exit points in the software, we have used a software interrupt to force a trigger in the oscilloscope and start the acquisition of voltage and current measurements. Two one-instruction macros (TOSH_SET_INT3_PIN, TOSH_CLR_INT3_PIN) move the interrupt pin level up and down and allow to mark instruction blocks in order to evaluate their energy consumption.

A measurement of the energy consumption of an empty code block has been made in order to take into account the interruption firing energy consumption. This extra energy consumption is mainly due to the change of the pin level.

Figure 4 shows the energy consumption of the interrupt which changes every 100ms. The energy consumption is mainly due to the change of the pin level.

3 TinyOS runtime environment

TinyOS is an open-source operating system designed for wireless embedded sensor networks. It is component oriented and event driven. The OS is mostly written in NesC,
Figure 3: Scheme of the circuit used for measurements of Mica2 energy consumption.

Figure 4: Interrupt Power consumption sample

Figure 5: Measurement setup

a C-based language which provides support for TinyOS component and concurrency model allowing fast development by wiring components.

TinyOS is compiled with user application into the same binary image and lacks of memory isolation. The kernel has a simple FIFO based task scheduler. TinyOS is popular for the large number platforms supported, the add-ins that surround it, like simulators (TOSSIM, TinyViz), and the large number of components and applications in which it is used.

A TinyOS application is composed of application components, the scheduler, and a set of system components. Each component is formed by:

- **Frame**: Contains all static variables available to the implementation of the component. No dynamic memory is available.

- **Commands**: Non-blocking request to low level components. It can call other commands or create tasks but may not signal events.

- **Events**: They are normally invoked by hardware interrupts and they can signal other events, create tasks or even call commands. They usually store the information in components frame and create a task for managing the information after interrupts are processed.

- **Tasks**: Component function put in a queue and later selected to run by the TinyOS scheduler.

TinyOS architecture as described above is simple and small, and the only overheads [9] is produced by its component model and events management. The component model is achieved through NesC language and the component wiring. The wiring, is done at components configuration files by connecting components interfaces, and allows to link component functions between components. All the code is compiled into a C generated file which can be optimized. Mainly, the optimization has to be
done at code generated by wiring because unnecessary code is produced. Another possible improvement can be done in event management and task management. Events are produced at any level by hardware or software interrupt and they are usually finalized in tasks produced by low level drivers. Some events can be managed without task creations.

In the following subsections the following proposals are extended:

- Multiple levels of software due to componentization.
- Usage of hardware power modes.
- Overhead due to the creation of scheduled tasks for delayed execution.

3.1 Runtime support overheads evaluation

NesC provides easy and fast development of applications by allowing the generation of new components by simply wiring them with other existing ones. All NesC files (application, libraries and operating system) are compiled to C-code and compiled with avr-gcc. NesC components are composed of an implementation and a configuration file. In the configuration file, the component is wired to other component interfaces. This wiring takes form of function calls and although it has low overhead, a lot of this consecutive wirings leads to inefficient C generated code. The Blink application has been used to evaluate it. Manually rewriting of this issue has been made in order to evaluate its impact in energy consumption. Blink application is based on a timer that turns on and off a led of the Mica2 board repeatedly. The timer routine has been modified in order to optimize by in-lining the code which turns on and off the LEDs (*Leds.redOn* and *Leds.redOff*).

Figure 6 shows the power consumption by calling the component *Leds.redOn()* function once the timer is fired, whereas figure 7 shows the power consumption with the code of the function in-lined. The physical led has been disabled in order to avoid its energy consumption disrupt the measurement of the involved software.

![Figure 6: Power consumption in timer function without in-lining](image6)

![Figure 7: Power consumption in timer function in-lining code](image7)

<table>
<thead>
<tr>
<th></th>
<th>Blink App. with no inline</th>
<th>Blink App. inlined code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy Consumption</td>
<td>13.03 mJ</td>
<td>10.21 mJ</td>
</tr>
</tbody>
</table>
Table 2 shows the energy consumption of the execution of Blink Application with and without the inline code optimization. The decrease in energy consumption is due to a reduction of the number of instructions and consequently the execution time as observed in figure 7 vs figure 6. Code optimization is less by 21.64% than the energy consumed without the optimization in the Blink Application.

3.2 Dynamic Power Management

Some microprocessors allow the reduction of power consumption by using both voltage and frequency scaling. However, neither voltage nor frequency scaling is possible in Mica2 platform and many other WSN platforms. The CPU and RF communication hardware are the elements that consume more power in the Mica2. Both elements have low power modes. TinyOS scheduler changes the CPU mode when the task queue is empty to a low power processor mode. In a similar way, SOS has a low priority task that turns the CPU to low power mode when it is executed in order to provide efficient power management. The OS is able to know when the processor can be turned to low power state, for example, when the task queue is empty. Nevertheless, the OS has no information about the state of the RF communication in terms of usage. Our measures confirm that switching CPU from active state to sleep state reduces current consumption from 13 mA to 9 mA approximately. When RF hardware is changed to sleep mode current consumption is reduced to 50 μA.

<table>
<thead>
<tr>
<th>Interrupt Management</th>
<th>Task creation</th>
<th>without Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy Consumption</td>
<td>1.38 μJ</td>
<td>25.05 nJ</td>
</tr>
</tbody>
</table>

We will study the possibility to do all interrupt processing when the interrupt is received, and therefore to avoid a task creation. We can show that the elimination of a task creation causes a reduction in energy consumption. Figure 8 shows two different waveforms. The rectangular waveform, in grey color, corresponds to the voltage at the pin that is activated by the software interruption. The energy consumed is measured when the pin is while high level since the it marks the evaluated code. The black waveform displays the power consumption of Mica2 when the task is created. Figure 9 shows the voltage waveform at the interruption pin and the power consumption waveform when the task creation is avoided and the interruption routine runs all selected code.

Table 3 shows the energy consumed by events using Task creation and no using Task creation. An energy consumption reduction of 98% takes place when Task creation is avoided.

![Figure 8: Interrupt management with Task creation](image)

3.3 Scheduler

The TinyOS scheduler is based on a simple FIFO algorithm. Each task is executed until finalization but can be stopped by any interrupt. As most embedded systems, interrupt code in the runtime is as short as possible in order to handle the interrupt and then postpone other not critical related work [13]. Therefore, interrupts are disabled for a short period of time and the possibility of missing other interrupts is small.
4 Comparison with Power Simulator

Applications in WSN have to be evaluated in terms of power consumption before deployment. Some previous experiences deploying a small network of 5 motes with Surge application allow us to familiarize with power management importance. Surge is a simple application that comes packaged with TinyOS. It is a simple send/report program; nodes periodically collect sensor readings and route them to a base station, which collects and graphically displays them for the user. Batteries lifetime achieved was between 5 and 7 days with a 10 minutes timer for sensor sampling, and with Deluge [14] update code capabilities.

PowerTOSSIM characterizes hardware components in order to log its usage during simulation and later analyze its energy consumption. Since in simulation execution the code is translated to native binary of the host used for simulation, it is difficult to quantify energy consumption because of different CPU instructions. The Simulator tries to log how much time the CPU spends in each power state assuming that in a simple microprocessor power consumption is approximately constant for all instructions. In order to perform more accurate measurements the simulator analyses instruction blocks to map with AVR instructions and obtain the corresponding cycle counts, but the simulator resolution is not enough to evaluate our optimizations.

The measurements done in this paper are mainly to evaluate the runtime support and they are of a very fine granularity. All changes done in Section 3 have also been tested with PowerTOSSIM but he has not been able to report any difference with these optimizations. PowerTOSSIM replaces low level components with instrumented ones. Therefore, it cannot detect modifications at the application level and task scheduling.

5 Future Work and Conclusions

Power Management is very important for software architects in order to achieve long lifetime systems in terms of battery or even to substitute batteries for energy harvesting generators. This paper evaluates the energy consumption of TinyOS. Wireless sensor designers need an accurate energy consumption information in order to ensure planned performance and expected lifetime before deployment.

The measurements of the simple proposed modifications done to TinyOS have originated a decrease in terms of energy consumption of Mica2. The reduction in energy consumption is a 21.64% for code optimizations by in-lining code. Moreover, the energy consumption reduction of 98% has been achieved avoiding task creation for handling interrupts. This reduction has been achieved by avoiding task creation for handling interrupts and code optimizations by in-lining code. Therefore, an energy consumption reduction can be obtained by the decrease of consumption of complex digital systems and also by an efficient software management.

The future direction for the work presented in this paper is the design of a runtime oriented to efficient power management in combination with energy harvesting techniques. Another point that will be object of study is the possibility of adding this power analysis to existing simulators to get more accurate results.
References


