Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 10 (2012) 1166 – 1171

Procedia Computer Science

The second International Workshop on Sensor Networks for Information Gathering and Monitoring (SNIGM 2012)

Estimation of energy consumption for TinyOS 2.x-based applications

Stefano Abbate[a], Marco Avvenuti[b], Daniel Cesarini[b], Alessio Vecchio[b]

[a]*Dept. of Computer Science and Engineering, IMT Institute for Advanced Studies, Lucca, Italy*
[b]*Dept. of Information Engineering University of Pisa Pisa, Italy*

Abstract

The development of energy-efficient applications for wireless sensor networks requires mechanisms and tools for run-time monitoring of energy consumption. We propose a software framework that supports energy profiling of applications for the TinyOS 2.x platform. Measurements are obtained through the insertion of software probes within the code of the operating system. As a consequence, since the APIs are not changed, the programmer is not forced to modify the code of existing applications. The technique has been validated by comparing its results with the values registered by dedicated hardware.

Keywords:
Wireless sensor network, Energy measurement, TinyOS

1. Introduction

Designing applications and protocols for wireless sensor networks (WSNs) requires careful management of the limited energy budget. A thorough understanding of how energy is spent is the first step to produce energy efficient code [1]: the developer needs to know which is the fraction of energy dedicated to the different application activities, such as communication, sensing, and computing. On the basis of such information, it is possible to reduce the impact of the most energy-hungry activities both statically, e.g. by re-designing/re-implementing parts of the application [2], and dynamically, e.g. by using on-line estimation of energy consumption to tune the parameters of operation.

Systems for the estimation of energy consumption can be implemented both in hardware and in software. Hardware-based approaches rely on the presence of additional circuits on the node board (to provide run-time measurements) or involve the use of an oscilloscope/multimeter (in this case measurements are collected before deployment). The positive aspect of hardware-based solutions is that they can be very accurate; on the contrary, they suffer from the costs associated with the additional hardware and the analysis of application behavior may be limited to lab sessions, where the operating environment can be significantly different from the real operating conditions. Software-based approaches do not require additional hardware and can be easily integrated with run-time energy management strategies. The drawback is that they may introduce overhead (both in terms of code size and run-time execution), and measurements cannot be as accurate and detailed as the ones provided by hardware-based techniques. In this paper we focus on software-based approaches.

In general, these solutions are based on the idea of measuring the time spent by the hardware subsystems in the different power states. This can be done both in a simulated environment [3] or on real nodes [4]. The advantage of a simulated environment is that different scenarios can be created and tested easily. Nevertheless, simulations are based

1877-0509 © 2012 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.procs.2012.06.167

on simplified models and may miss some important factors. Using real nodes has the main advantage of taking into account all the details of real environments and operating conditions.

We designed and implemented a software framework that adds energy estimation functionality to the widely used TinyOS 2.x operating system. In its current implementation the estimator is compatible with the TelosB/TmoteSky hardware platform [5], but the approach is general and can be extended with limited effort to other boards. The code of the operating system has been instrumented with software probes which measure the time spent by the main components (radio, CPU, persistent memory, etc.) in the different power states. Since the interface between the application and the operating system has not been modified, the programmer is not forced to change his way of coding and existing applications can be profiled without any effort. Information about energy consumption of a given node can be transferred to the base station through the network if the node is not easily accessible, or through the USB port if the node is attached to a PC. On the base station a GUI provides user-friendly access to all relevant information.

2. Power Model

Energy estimation relies on the accurate definition of a power model. Usually it is not possible to have an exact evaluation of the energy consumption, but it is possible to approximate it with a small error. This section describes how to define such an approximation, starting from basic power and energy equations.

In a time discrete system, energy consumption, expressed in Joules, can be calculated as:

$$E = \sum_{i=0}^{i=max} P_i \times \Delta t_i = \sum_{i=0}^{i=max} I_i \times V_i \times \Delta t_i \quad (1)$$

Real devices have a non-linear power dissipation behavior dependent on time and supply voltages, but using a power model based on non-linear functions is hard or computation intensive. A linearization process is then necessary. In this work, an approximate power model is used based on the ones proposed in [6] and [7]. Such models are proven to be sound with battery-powered devices as long as the voltage supply remains within the typical operating range.

Referring to Eq. (1), I_i is calculated as: $I_i = I_0 \times (V_i/V_{ref})$, where I_0 is the current absorption value taken from data-sheets, V_i, which depends on the charge of the battery, is measured by the system from the pins of the power supply, and V_{ref} is the data-sheet reference voltage. In our system, values for V_i are taken by a native function of the operating system, values for Δt_i are calculated by software probes inserted within the device drivers.

3. TinyOS and energy management

TinyOS [8], an open source operating system written in nesC [9], is the de-facto standard for WSNs. A nesC application is composed of one or more components wired together. Each component can provide interfaces to other components or use interfaces provided by other components. Interfaces are bidirectional and include a set of commands, implemented by the interface provider, and a set of events, implemented by the interface user. Components can be of two types, modules and configurations: modules provide the implementations of one or more interfaces, configurations are used to wire components together.

The radio is among the most energy expensive subsystems of a sensing device. For this reason, TinyOS provides a radio feature known as Low Power Listening (LPL). By this technique, a sensing device turns the radio on every *LPL period* to check the presence of a carrier on the channel. If a carrier is detected, the radio is kept on to receive the packet.

The processing unit has several power states, with different power drain and wake-up latencies. The transitions between the power states are known. In TinyOS, a processing unit is in a low power state and switches to an active state when an interrupt arrives. When the task queue is empty, the processing unit returns to a low power state.

4. Approach and implementation

Software-based run-time energy estimation is based on tracing the state transitions experienced by hardware components when executing software. The cumulative time spent in each state is then computed and used to estimate the total energy consumption, according to a power model that takes into account the current drain on each power state.

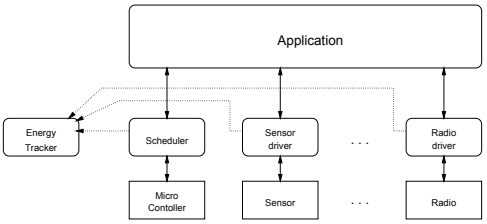

Figure 1: Estimator architecture.

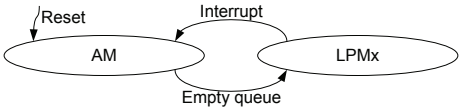

Figure 2: Finite State Machine of the MCU (MSP430).

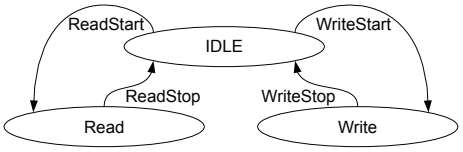

Figure 3: Finite State Machine of the EEPROM.

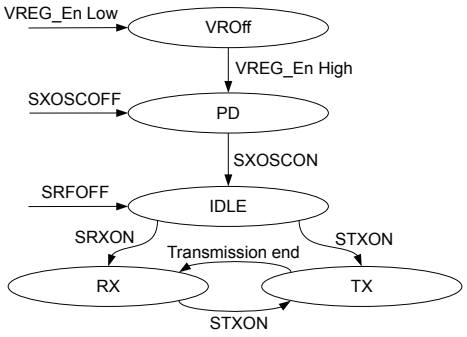

Figure 4: Finite State Machine of the CC2420 radio.

To implement our system we adopted the same approach used in [7] and [2]: the operating system is instrumented to capture and log state transitions. This can be achieved by modifying the device drivers and make them able to time stamp and denote state transitions. Such a mechanism is fully transparent to programs but, in order to limit system overhead, it requires to set the granularity with which state transitions of monitored components are traced.

Here we describe how we implemented the mechanism in the TinyOS v2.x operating system, and how we modeled the finite state machines for the CPU, EEPROM, Radio, and Leds subsystems. The implementation refers to the TmoteSky/TelosB hardware, but it can be easily ported to other sensor node platforms.

Sensor nodes include several subsystems: the micro controller unit (MCU), the Radio, three LEDs, the EEPROM, a number of passive sensors and active sensors. We define an *A*ctivity as the time interval during which a given subsystem remains in one of its power states. Considering the current drain of each state, we can identify the activities that significantly contribute to energy consumption and monitor them.

The system architecture is depicted in Fig. 1. A new module, called `EnergyTracker`, is added to the operating system for accounting the activities of each subsystem. For each Activity, the module manages a separate time counter through a commands pair, provided by the following generic interface:

- *notify*Activity*Start()*
- *notify*Activity*Stop()*

Commands must be called by the scheduler and by the device drivers every time a state transition occurs in the corresponding hardware subsystem.

Relevant activities of the MCU are Active Mode (AM) and Low Power Mode (LPMx), the latter representing five different power states. Activities are monitored by adding code to the scheduler that, according to the graph shown in Fig. 2, will call the `EnergyTracker` when the task queue becomes empty ($AM \rightarrow LPMx$), and when timer or I/O port interrupts occur ($LPMx \rightarrow AM$).

The EEPROM's activities can be modeled as shown in Fig. 3. The commands called by the EEPROM driver are the pair `notifyEepromReadStart()` and `notifyEepromReadStop()` for recording a Read activity, and the pair `notifyEepromWriteStart()` and `notifyEepromWriteStop()` to notify a Write activity.

0-32s	receive (R): fully on - send (S): every second to fully on listeners
32-64s	(R): fully on - (S): every second to LP-listeners with 100ms interval
64-96s	(R): LP-listening, 250ms interval - (S): every second to LP-listeners with 250ms interval
96-128s	(R): LP-listening, 250ms interval - (S): every second to fully on listeners
128-160s	(R): LP-listening, 10ms interval - (S): every second to LP-listeners with 10ms interval
160-192s	(R): LP-listening, 2000ms interval - (S): every 7 sec. to LP-listeners with 2000ms interval

Table 1: TestLPL Application time sequence.

The behavior of the CC2420 radio transceiver [10] is described by several states. For the purpose of energy consumption, we considered the following five relevant activities: Voltage Regulator Off (VROff), Power Down (PD), IDLE, RX and TX. The radio's behavior was modeled by the simplified finite state machine shown in Fig. 4. The modified device driver calls command `notifyRadioStart(state)` to notify a started `state`, and the command `notifyRadioStop(state)` to notify a terminated `state` .

The three LEDs are simple subsystems that can switch only between the On and Off states. To keep track of the ON activity, the `EnergyTracker` provides the `notifyLedsOnStart(ledId)` command to store the time the LED `ledId` is switched on, and the `notifyLedsOnStop()` command to calculate the ON activity time interval of LED `ledId`.

The `EnergyTracker` periodically sends the collected activities to the base station, where time data are processed according to the power model. A graphical user interface (written in Java language) presents information and graphs about time usage of each component and an estimation of their energy consumption. The system also provides a logging facility that creates comma separated value files. To use the instrumented version of TinyOS 2.x a nesC application must be compiled with the flag `CFLAGS += -DENERGY_EST` set in the *Makefile*.

5. Validation and evaluation

System validation has been carried out by measuring the real current absorption while running a set of benchmark applications. Each application keeps active only one hardware subsystem at time. Currents were measured by using a Digimaster DM3900 multimeter serially connected between the battery pack and the power supply pins of the mote under test. The benchmark applications are the following: i) MCU-Lpm and MCU-Act that always keep the MCU of a mote in LPM1 mode (one out of five possible modes) and in Active Mode, respectively; ii) Radio-Rx and Radio-Tx that always keep the radio subsystem in receive and transmission modes respectively; iii) LedOn that keeps one LED always on.

Fig. 5 shows a comparison between real current drain measured using the multimeter and the values measured by our software system. The figure highlights that when the MCU is in Low-Power mode only a few errors are present, whereas measurement errors increase in Active Mode. The system performs better considering the Radio subsystem, in Rx and Tx modes. In LedOn application there are also considerable errors. Errors can originate from the linearization process of the power model.

To evaluate how the estimator tracks the activities of subsystems we used a publicly available application taken from the TinyOS repository[1]: the *TestLPL* application tests the activity of the radio using the LPL mode. The application changes various parameters related to LPL mode according to an activation schedule, as reported in 5.

Fig. 6 shows the estimator output in terms of time usage of radio's states for the *TestLPL* application. As we can see from the figure, the system is able to track changes in radio-subsystem usage coherently with the schedule in 5. However, during this test we experimented little errors of the system reporting correct times, that led the system to underestimate actual radio-subsystem usage. This happens beacause time calculation is done using a 32KHz oscillator. A switching activity comparable to that frequency increases Δt_i calculation errors. Errors become higher when the radio in LPL mode uses short wake-up periods.

[1] We chose to work with standard application to enable future comparisons of our system with others.

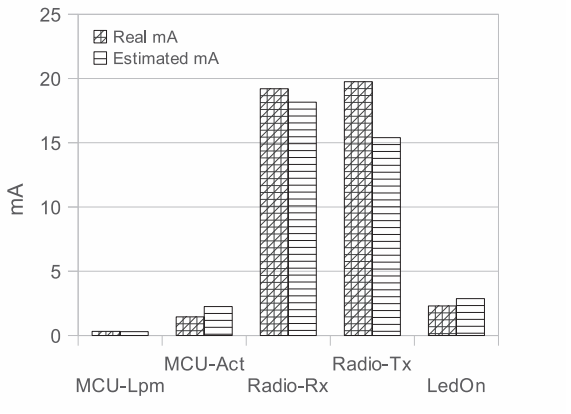

Figure 5: Validation of estimator.

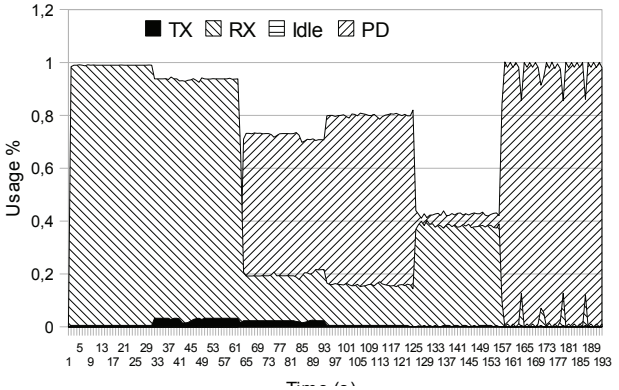

Figure 6: Estimation of radio usage in TestLPL application.

6. Related work

Software-based on-line energy estimation mechanisms for sensor networks are described in [2] and [7]. They use a mechanism, based on software probes inserted within the operating system, general enough to be used in different platforms. The approach requires no changes in applications or protocols, thus it is totally transparent to the programmer. Differently from the system described in [7], which was based on TinyOS 1.x (no longer supported), our system is implemented on top of version 2.x of TinyOS. The system presented in [2] is based on the Contiki OS.

Some simulation tools, such as PowerTOSSIM [3], enable the modeling of WSN nodes in terms of energy consumption. Such simulators provide an estimation of energy consumption with little effort (no need to set up a real testbed, no need to change the application code). However, using abstract models of hardware, they cannot include all the details of real hardware and environments.

In [11] Kellner proposed an energy management system for dynamic sensor networks composed of three components: i) An energy model of the motes; ii) An accounting infrastructure needed to make motes energy-aware; iii) Resource containers to manage the energy accounting information. The energy model represents each hardware subsystem as a finite state machine. In [12] the author describes an extended version that includes two new components: An energy-estimation system to collect information about energy consumption, and an energy-container system to collect energy-consumption information about individual tasks. The system requires the modification of the application code in order to make the correct calls to the accounting system.

Quanto [13] uses the iCount hardware system [14] to measure energy usage. The authors modified device drivers to track and provide hardware power states to the operating system. To use Quanto massive modifications of the application's code are needed.

7. Conclusions

This paper described how to provide WSN applications running on TinyOS 2.x with software on-line energy measurement. By operating at device driver level it is possible to have a fine-grained estimation of the time usage and energy consumption of each mote's subsystem. The estimation process is transparent to the programmer, who only needs to re-compile his applications. The approach taken started from the analysis of the subsystem' activities to design modifications to the operating system. Validation and experiments showed that the estimated energy consumption has little errors compared to real measurements. The proposed framework can be used for both sensor network prototyping and deployment stages.

It should be noted that data about current drains found on technical data-sheets is not sufficient to have a good energy consumption estimation and further improvements can be achieved. As a future work, the current consumption of different subsystem could be calculated once for all at different voltage variations. This could be done by directly measuring the current values using a hardware testbed. In this way a finer grained current estimation could be achieved.

[1] X. Jiang, J. Taneja, An architecture for energy management in wireless sensor networks, in: In International Workshop on Wireless Sensor Network Architecture (WSNA07, ACM Press, 2007, p. 2007.
[2] A. Dunkels, F. Osterlind, N. Tsiftes, Z. He, Software-based on-line energy estimation for sensor nodes, in: in Fourth Workshop on Embedded Networked Sensors, 2007, pp. 28–32.
[3] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, M. Welsh, Simulating the power consumption of large-scale sensor network applications, in: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), ACM, New York, USA, 2004, pp. 188–200. doi:http://doi.acm.org/10.1145/1031495.1031518.
[4] T. Stathopoulos, D. McIntire, W. J. Kaiser, The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes, in: Proceedings of the 7th international conference on Information processing in sensor networks, IPSN '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 383–394. doi:http://dx.doi.org/10.1109/IPSN.2008.36.
[5] Moteiv, Tmote Sky Datasheet http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf (2006).
[6] F. Kerasiotis, A. Prayati, C. Antonopoulos, C. Koulamas, G. Papadopoulos, Battery lifetime prediction model for a wsn platform, in: Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on, 2010, pp. 525 –530. doi:10.1109/SENSORCOMM.2010.85.
[7] T. Yang, Y. K. Toh, L. Xie, Run-time monitoring of energy consumption in wireless sensor networks, in: Proceedings of the IEEE International Conference on Control and Automation, IEEE Press, 2007, pp. 1360–1365.
[8] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler, Tinyos: An operating system for sensor networks, in: W. Weber, J. M. Rabaey, E. Aarts (Eds.), Ambient Intelligence, Springer-Verlag, Berlin/Heidelberg, 2005, Ch. 7, pp. 115–148. doi:10.1007/3-540-27139-2_7.
[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, The nesc language: A holistic approach to networked embedded systems, in: PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, ACM, New York, NY, USA, 2003, pp. 1–11. doi:http://doi.acm.org/10.1145/781131.781133.
[10] Chipcon, Cc2420 datasheet (2007).
[11] S. Kellner, F. Bellosa, Energy accounting support in tinyos.
[12] S. Kellner, Flexible online energy accounting in TinyOS, in: P. Marron, T. Voigt, P. Corke, L. Mottola (Eds.), Proceedings of the 4th International Workshop on Real-World Wireless Sensor Networks (RealWSN'10), Vol. 6511 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 62–73. doi:10.1007/978-3-642-17520-6_6.
[13] R. Fonseca, P. Dutta, P. Levis, I. Stoica, Quanto: tracking energy in networked embedded systems, in: Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 323–338.
[14] P. Dutta, M. Feldmeier, J. Paradiso, D. Culler, Energy metering for free: Augmenting switching regulators for real-time monitoring, in: Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on, 2008, pp. 283 –294. doi:10.1109/IPSN.2008.58.