High-Fidelity Power Metering and Run-Time Energy Management in Wireless Sensor Networks

by

Xiaofan Jiang

B.S. University of California, Berkeley 2004

A thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Engineering - Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

David E. Culler, Chair
Ion Stoica

Fall 2007

The thesis of Xiaofan Jiang is approved.

Chair Date

Date

University of California, Berkeley

Spring 2007

High-Fidelity Power Metering and Run-Time Energy Management in Wireless Sensor Networks

by

Xiaofan Jiang

Abstract

High-Fidelity Power Metering and Run-Time Energy Management in Wireless Sensor Networks

by

Xiaofan Jiang

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

David E. Culler, Chair

In this thesis, we present the design and evaluation of the SPOT power metering system, and its application in run-time energy management.

SPOT is a *scalable power observation tool* that enables *in situ* measurement of nodal power and energy over a dynamic range exceeding four decades and at a temporal resolution of microseconds. Using SPOT, every node in a sensor network can be instrumented, providing unparalleled visibility into the dynamic power profile of applications and system software. Power metering at every node enables previously impossible empirical evaluation of low power designs at scale. The SPOT architecture and design meet challenges unique to wireless sensor networks and other low power systems, such as orders of magnitude difference in current draws between sleep and active states, short-duration power spikes during periods of brief activity, and the need for minimum perturbation of the system under observation.

Using SPOT as an underlying service, we present a run-time energy management architecture (EMA), closing the feedback loop between observation and actuation. EMA adopts a three component decomposition - a policy interface for user input, a mechanism to monitor system and component resource usage, and a management module for enforc-

ing policy directives. EMA provides facilities for 1) individual accountability of energy-consuming units, 2) graceful degradation and predicable operation in a dynamic environment, and 3) expression of network-level policies that can be used by individual nodes for local optimization and shared among nodes to adjust behavior of the network as a whole.

David E. Culler
Thesis Committee Chair

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Energy is often the limiting resource in wireless sensor networks ("sensornets"). As a result, energy-efficiency has pervaded nearly every aspect of sensornet research, from platform designs [1]–[3], to MAC layers [4], [5], to routing protocols [6]–[8], and to applications [9], [10].

However, because it is currently hard to empirically measure the energy and power consumption of a network of sensor nodes in situ and at scale, researchers often have to rely on either simple approximations or model-based simulators. Approximations of nodal energy usage derived from estimates of node duty cycle and communication rates [11] do not capture the low-level system power profile. Simulators that extrapolate system macro-benchmarks – the large-scale, long-term, and system-wide behavior of a sensor network – from models based on micro-benchmarks of a single node cannot assure the accuracy of their generalizations [12].

Furthermore, as sensornets are becoming more widely adopted for commercial and scientific use, in settings where battery replacement or recharging is difficult, it is important for the node to be able to automatically adjust its energy consumption to adapt to its available energy resources at run-time, thus requiring an underlying energy metering service.

It is becoming increasingly apparent that what is needed is a method for high-fidelity, in-situ, and at scale power metering, that will enable both the observation and subsequently

the management of energy. In this paper, we present the design and evaluation of the SPOT power metering system, and its application in run-time energy management. The rest of this paper is organized as follows. We present the hardware architecture and design issues in Chapter 2, followed by the system issues in obtaining and calibrating the data via software drivers in Chapter 3. To demonstrate scalability of SPOT, we describe a SPOT-enabled testbed and network related issues in Chapter 4. And finally we propose a run-time energy management architecture based on SPOT in Chapter 5.

Chapter 2

Hardware

Characteristics unique to sensornets make power and energy monitoring challenging. Figure 2.1 shows the current profile of a typical sensornet application. Long periods of low-current sleep are punctuated by short, periodic bursts of high-current activity.

Since more than three orders of magnitude separate the current draw in the sleep and active states, it might seem reasonable to ignore the energy usage in the sleep state. However, because sensornets operate at very low duty cycles ranging between 0.1% to 1%, both the sleep and active states account for non-trivial fractions of the system power budget. This suggests that a metering system must have a *dynamic range that significantly exceeds three orders of magnitude* to capture sleep current with sufficient resolution.

Figure 2.1a shows that a node may be active for a short time, on the order of 20 ms, before returning to the sleep state. However, the node's current draw is not constant during this 20 ms period. Rather, the current profile includes two large transients lasting tens of microseconds, four different levels, and oscillations during some level transitions (2.1b). Such features in nodal current profiles are common and reflect the wide variety of system components and the sum of their various power states and state transitions. This suggests that *sampling rates approaching tens of kHz or even MHz* may be needed to faithfully capture these ephemeral features.

To prevent excessive perturbation of the system under test, the metering system should

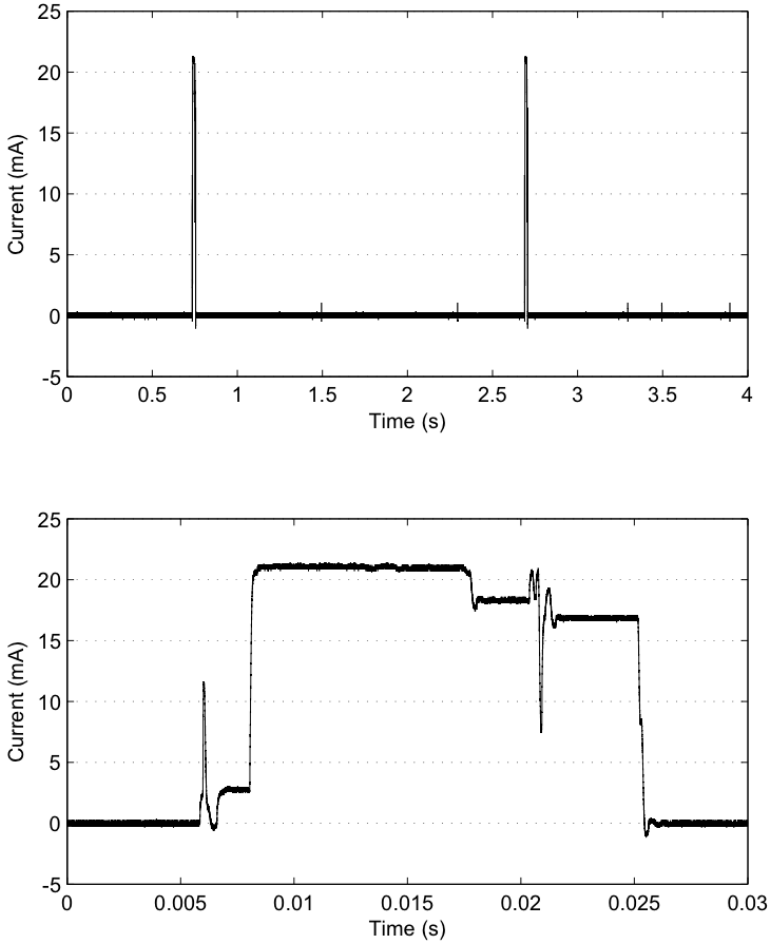


Figure 2.1. A typical nodal current profile consists of long periods of low-current sleep punctuated by short, periodic bursts of high-current activity.

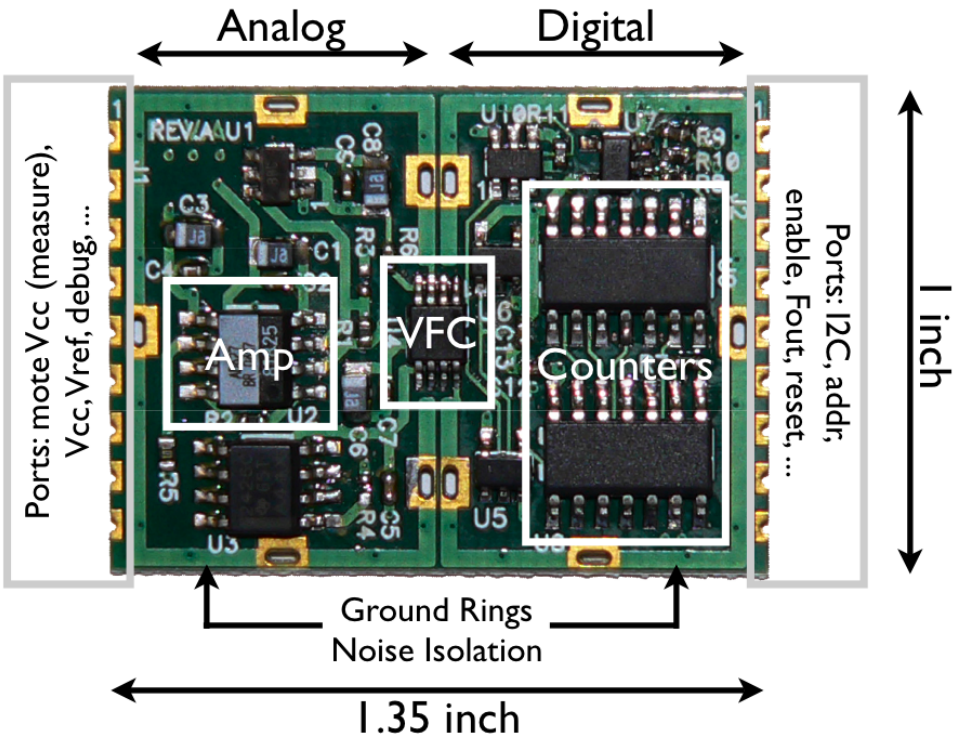


Figure 2.2. The scalable power observation tool (SPOT) consists of a sense resistor, amplifier, voltage-to-frequency converter, and two counters.

be minimally invasive: it should require low computational and storage resources from the host node, and it should be able to operate in a stand-alone manner with limited host interaction. Finally, *in situ* metering *at scale* requires small size and low cost.

Our solution to this metering problem is shown in Figure 2.2. This system, a *scalable power observation tool* (SPOT), enables *in situ* measurement of nodal power and energy with a dynamic range exceeding 10000:1 and a temporal resolution in the order of microseconds. SPOT accumulates current internally and exports digital I/O lines to enable/disable metering and calibration, analog lines to measure instantaneous current, and an I2C interface to read the accumulated current (energy) and reset the counter.

2.1 Challenges

This section presents the four basic requirements of our micropower meter for sensornet nodes ("motes"): dynamic range, sampling rate, perturbation, and ease-of-integration.

2.1.1 Dynamic Range

A sensornet node can exhibit a bewildering array of power profiles, depending on its application profile. For example, a simple *sense-and-send* application with a 0.5 Hz duty cycle might exhibit the profile shown in Figure 2.1. In contrast, a *sense-and-store* application may sample sensors for a few milliseconds every five minutes, buffer these sensor readings in RAM, write them to flash once a day, and attempt to upload the samples once a week. The current profile for such an application would be starkly different from that shown in Figure 2.1. Since SPOT is to be used in a testbed, we cannot assume a particular application profile *a priori*, implying the system needs to have a wide dynamic range spanning the entire spectrum of possible current draws.

2.1.2 Sampling Rate

Wireless sensor nodes are pulsing applications. Their pulse width, or the active cycle time, needs to be considered when determining the appropriate sampling rate. If the duration of an active pulse is shorter than the sampling rate of the energy meter, the energy in that pulse may be missed. In addition, the spectral content within a active pulse should also be sampled at a rate that exceeds the Nyquist rate.

While we can estimate the minimum active cycle time by observation, we cannot guarantee that the spectral content will be band-limited to a particular range. In our example, the width of an active cycle is approximately $20ms$. However, it's power spectral density (PSD) exhibits energy across the entire spectrum, and therefore it is necessary to bandlimit the signal with a low pass filter (LPF). The cutoff frequency for the LPF should be the highest frequency in the PSD that still contains significant energy. The cutoff frequency in

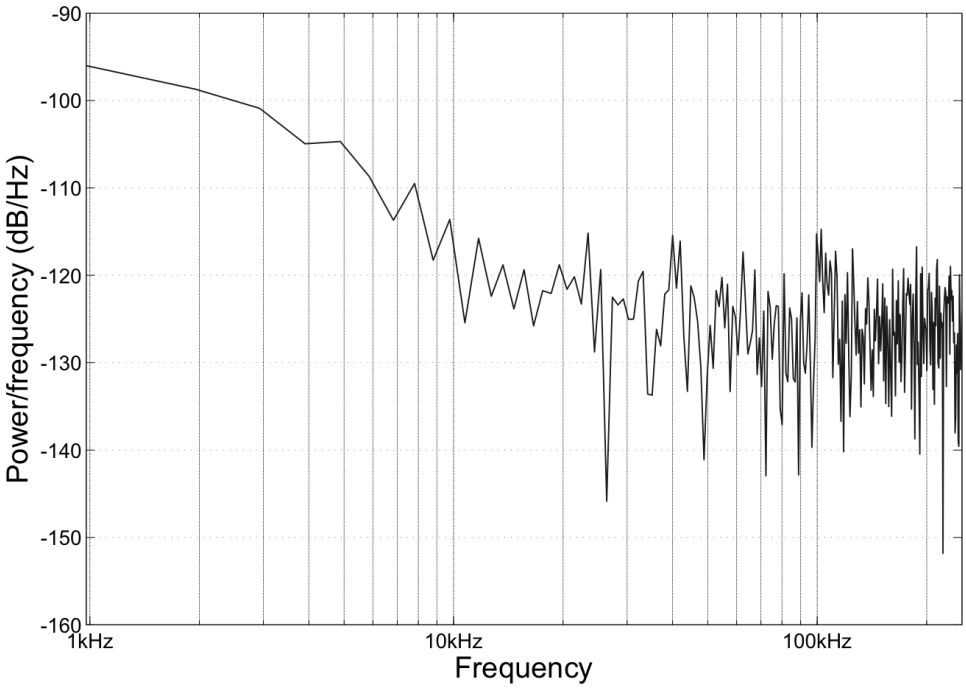


Figure 2.3. The power spectral density of a node current profile shown in Figure 2.1.

our example is around $20kHz$, as shown in Figure 2.3. This implies a minimum sampling rate of $20kHz \times 2 = 40kHz$ per Nyquist's theorem.

Intuitively, we can see that to capture most of the energy content in the $20ms$ active cycle in Figure 2.1, especially the oscillations at $6ms$ and $21ms$, we will need to sample at least every $0.1ms$, which implies a sampling frequency of $10kHz$.

2.1.3 Perturbation

Energy monitoring should not affect the actual energy consumption of the mote under study (hereby referred to as the device under test, or DUT). When the user of the energy monitor is separate from the DUT, accessing data from the energy monitor should not affect the energy consumption; when the user of the energy monitor is part of the DUT itself, then accessing the energy monitor should present a minimal perturbation to the DUT.

The energy monitoring device should not affect the measured power of the DUT. This implies that the the energy monitor should be powered from a separate power source;

however, if it must share the same power supply as the DUT, the point of measurement should be confined to the DUT.

The energy monitor should not require the DUT to perform extensive computation, if any. This implies that for the case when the user and the DUT are the same mote, a meter that can only provide power measurements is out of the question because to obtain an energy measurement, the DUT will need to constantly read from the meter to accumulate energy measurement. Instead, this computation should be offloaded to the energy monitor to minimize the energy and CPU usage of the DUT.

2.1.4 Ease-of-Integration

One of the goals for this project is for every mote on a network to be equipped with an energy meter. This suggests that the system needs to be easy to integrate, both electrically and mechanically, into a sensornet node. To be practical, the meter must be inexpensive, or perhaps comparable in cost to the sensornet nodes.

2.2 Existing Solutions

Many commercially available energy meters operate by measuring the voltage drop around a shunt resistor R tied to the power supply of the circuit under observation, as shown in Figure 2.4. The voltage drop is proportional to the current and can be multiplied by a voltage from a separate channel to obtain the true power usage. The voltage across the resistor is usually first magnified by an amplifier, and then undergoes an analog-to-digital conversion (ADC). The rest of the computations, such as dividing by the resistance and multiplying by the voltage, are usually done in the digital domain. This power computation occurs many times in a second (from a few Hz to MHz) and is summed in an integrator to obtain energy. The results are stored in registers and presented at the output either as PWMs (pulse width proportional to energy) or in digital form.

Oscilloscopes are often used to capture the power profile of a system at bench scales

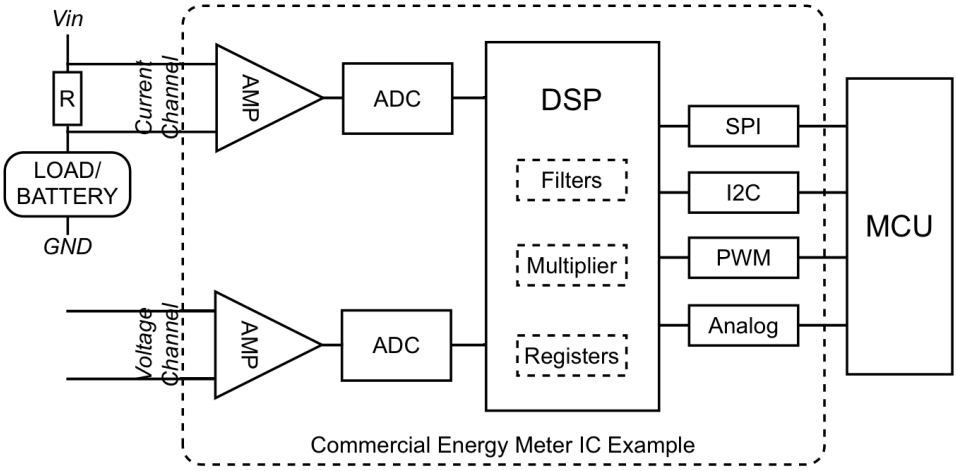


Figure 2.4. Architecture of a typical energy metering circuit.

since their high sampling rates, often in the GHz range, can provide very fine temporal resolution. However, their high cost makes them unsuitable for large scale usage. But, even if cost were not an issues, oscilloscopes would not be suitable. First, most oscilloscopes are limited to milliamp-level vertical sensitivities, rather than the needed microamp levels. Although a pre-amplifier can be used, this adds cost and complexity. Second, they are not easy to integrate with typical sensornet nodes.

Sensornet-specific solutions have been proposed as well. For example, the DS2438 [13] is a battery monitor IC used in the HelioMote [3]. This is an 8-pin IC with a simple 1-Wire interface and integrated temperature sensing. However, the on-board ADC is only 10-bits, providing a theoretical maximum dynamic range of 1000:1, which is far below requirements for motes (10000:1). It samples at a frequency of 36.41Hz, also far below the 20kHz needed to capture interesting spikes.

Commercially available integrated circuits are not designed to meet our dynamic range and sampling requirements simultaneously. For example, battery monitoring ICs such as the Maxim Semiconductor's BQ2019 are targeted towards long term measurement of batteries and provides low temporal resolution. Multi-function metering ICs such as the Analog Devices' ADE7753 [14] have maximum dynamic ranges of around 1000:1, failing our vertical resolution requirement. Microchip's MCP3906 [15], an energy measurement IC supporting

the IEC 62053 international energy metering specification, offers a dynamic range of only 1000:1 and has offset currents that exceed node sleep currents by an order of magnitude, making it unsuitable for our purposes.

A very different approach, proposed by Shnayder et al., is to simulate power draw using an empirically-generated model of hardware behavior [12]. The simulator, called PowerTOSSIM, characterized its underlying model by instrumenting and profiling a single node. While PowerTOSSIM takes an important step toward providing better visibility into nodal power profiles, since its model is based on microbenchmarks of a single node taken in particular environment, it also raises several questions about the model's generality: How representative is the node that was instrumented to calibrate PowerTOSSIM's model? How does interaction with the physical environment shape energy usage? How do temperature variations affect leakage currents? How much variance occurs within a single node, and across different nodes, for the same operation? These questions can only be answered by instrumenting an entire network of nodes *in situ* and *at scale*.

2.3 SPOT Architecture

In this section, we present the SPOT architecture and highlight some of its key features. A more detailed analysis is presented in Section 2.4. Our architecture consists of four stages: sensing, signal conditioning, digitization, and energy output, as shown in Figure 2.5. This configuration shows the DUT and the user of the energy meter to be the same mote, which need not be the case. In the sensing stage, a shunt resistor is put in series with the mote (DUT), converting current to voltage. This voltage is proportional to the power consumption of the mote. In the conditioning stage, a differential amplifier and a low pass filter is used to amplify and band-limit the signal, respectively. In the digitization stage, a voltage-to-frequency converter (VFC) is used to convert voltage into a periodic wave with a frequency proportional to the input voltage, which is also proportional to the power. In the energy output stage, pulses from the output of the VFC are summed (i.e. integrated)

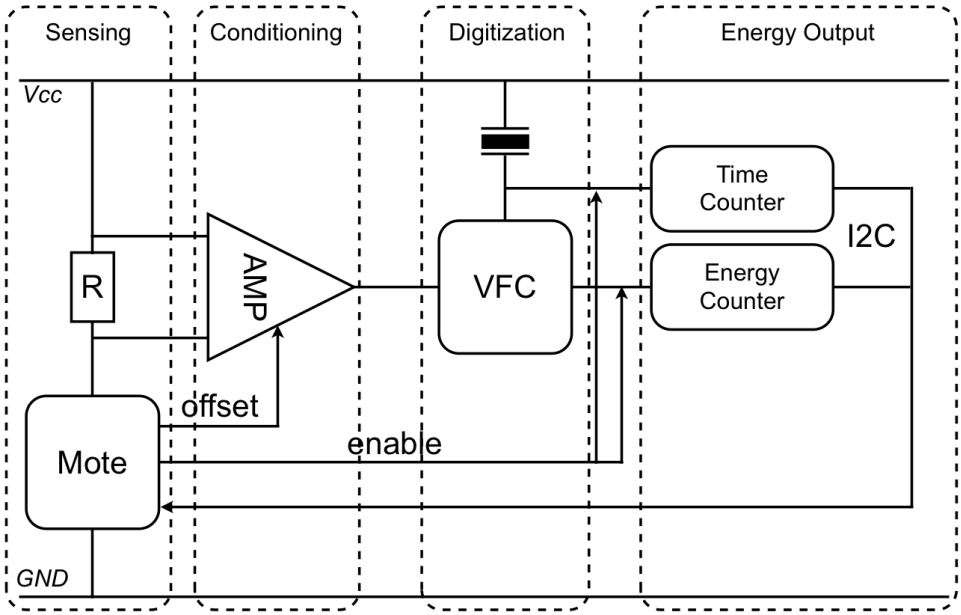


Figure 2.5. Architecture and Primary Components

in a counter to obtain energy measurements while a separate counter is used to keep a time-base; finally, the values of the counters are read back by the mote via I2C.

2.3.1 Differential Amplifier

While differential amplifier is simple in principle, it is more of an art to correctly use one in practice. Because we configured our shunt resistor at the high-side of the mote (see Section 2.4.1), the common-mode voltage is equal to Vcc (3.3V). This implies that not only does our amplifier needs to have a high common-mode input range, we need to introduce a second voltage supply (e.g. 5.5V) to correctly biased the amplifier.

Due to variations in device sizing, amplifiers have offset voltages (i.e. output is non-zero even if input is zero) and it's different for every amplifier. This presents us with a calibration problem. To compensate, we included a calibration switch which zeros the input at startup and records the counter value (corresponding to the offset). This value is stored in the MCU and used in the calibration curve.

Because we are trying to amplify a very small signal, any noise is significant. We placed

multiple RC filters (for different frequencies) around sensitive areas (e.g. voltage supplies of IC chips). Furthermore, we observed that the oscillator for the digital portion of our circuit introduces significant noise to the amplifier. To solve this problem, we carefully placed separate ground rings in the board to separate the analog portion of the circuits from the digital portion (see Figure 2.2).

2.3.2 Voltage-to-Frequency Converter

To achieve the desired dynamic range, we choose a novel approach of using a voltage-to-frequency converter as the analog front end (AFE). It converts the signal from the analog to the digital domain. While VFC has a clocked input, the voltage to frequency conversion is entirely analog in the sense that any arbitrarily small voltage is integrated inside an analog integrator and will eventually trigger an output pulse. This essentially gives the VFC *infinite resolution, limited only by noise*. In contrast, traditional ADCs will fail to capture voltage levels smaller than the specified *bit* resolution. Please refer to Section 2.4.3 for details.

2.3.3 Energy Counter and Internal Timebase

To minimize MCU overhead, we use an internal 32-bit counter to accumulate the power readings to provide direct energy measurement. The counter is free-running at a maximum rate of 0.9MHz or 0.9 million counts per second. It overflows in roughly $\frac{2^{32}}{2^{20}} = 2^{12}$ seconds or about once an hour. This implies that the MCU will only need to read once every hour if long term energy is all the application needs. But it can read as fast as I2C bus speed allows to obtain a fine-grain power profile by differentiating the energy readings.

To find the energy consumption for a given time window, we can read the energy counter at the beginning and again at the end. For this to work, we need some way of keeping the elapsed time. This can be done in the MCU by using the MCU's timer. However, this introduces MCU overhead and the time will not be accurate because many cycles will have elapsed from timer_fired() to an actual read from the energy counter. Instead, we included

another counter dedicated to be the time-base. This counter is triggered by the same oscillator (1MHz) that provides the base frequency to the VFC. We wired the I2C bus address such that one command can capture the two counter readings at the exact same time. This method has the additional benefit of nullifying any jitter in the oscillator due to temperature.

2.3.4 Power Introspection

Because SPOT provides power and energy measurements without significant MCU perturbation, it is perfectly reasonable for a mote to use SPOT to monitor its own power and energy consumption, allowing the application to perform power adaptation.

2.4 Design Issues and Tradeoffs

In this section, we revisit the SPOT architecture with an eye toward design issues and tradeoffs in the sensing, signal conditioning, digitization, and accumulation.

2.4.1 Sensing

As with most sensing systems, the first stage is the sensor itself. In the case of energy monitoring, the physical quantity we are trying to measure is power, which is a product of voltage and current. For the purposes of this paper, we assume that the voltage is fixed and known *a priori*, or can be measured separately. Of course, current can be measured in several ways. We chose to use a shunt resistor, which is one common method. A small resistor is placed in series with the power supply and the voltage across this resistor, which is proportional to the current, is measured.

There are several design considerations with this approach. The resistor can be placed between the mote and either the positive power supply or the negative power supply (ground). The former is called "high-side current sensing" while the latter is called "low-side current sensing". Low-side sensing is desirable because the differential voltage across

the resistor is equal to the voltage measured, with respect to ground, at the connection between the resistor and the mote. This simplifies the amplification stage because there is no common-mode voltage. However, low-side sensing also creates a problem known as ground bounce – as the current draw fluctuates, the negative supply of the system also fluctuates. This is undesirable because many electronic components are sensitive to ground fluctuations. This is also why the resistance must be kept small since the magnitude of this fluctuation is proportional to the resistance.

High-side current sensing places the resistor between the positive supply rail and mote's power input. By placing the resistor on the positive supply side, the voltage fluctuations are shifted from the negative supply side to the positive side. This is more desirable because most components are more resilient to fluctuations in the positive supply rail. However, this introduces common-mode voltage because the voltage on both sides of the resistor, measured with respect to ground, is non-zero. The presence of significant common-mode voltage can cause problems for amplifiers, as we discuss in Section 2.4.2.

We place a 1Ω resistor R between the positive supply rail (Vcc) and the mote's positive supply in a high-side configuration, as shown in Figure 2.5. The value of 1Ω was chosen to limit the supply voltage fluctuation. Assuming a maximum current of 40mA, typical of current mote technology, the maximum drop is 40mV, which is reasonable.

2.4.2 Signal Amplification and Conditioning

The current draw of a typical mote, such as Telos [1], ranges from $2\mu A$ to $40mA$.[1] Using a 1Ω resistor, the minimum voltage needed to capture is $2\mu A \times 1\Omega = 2\mu V$, which is too small for signal processing. Therefore, we first amplify this signal using differential amplifier, as shown in Figure 2.5.

The gain of the differential amplifier is set such that the maximum input voltage of 40mV, multiplied by the amplifier gain, is equal to the maximum input of the next stage and less than the maximum output of the amplifier. In this case, the input of the next stage is

[1] $2\mu A$ can be obtained by lowering the supply voltage below recommended values.

limited by the supply voltage, which is commonly 3.3V or 5V. Therefore, a reasonable gain is $\frac{3.3V}{40mA} = 82.5$. The remainder of this section explores some of the design and implementation challenges of this stage.

Dynamic Range

The required dynamic range is one of the key challenges in our system. We cannot artificially increase the amplifier gain because it is limited by the maximum input range of the next stage. With a gain of 82.5, a $2\mu V$ input translates to 0.165mV output, which is still quite small. At this point, we can either defer this problem to the next stage or try to alleviate it through some sort of signal processing.

One way to "decrease" the dynamic range requirement is to use a low pass filter (LPF) to lower the amplitude and widen the pulses. However, because LPFs are not energy preserving, they require software compensation and present timing difficulties during measurements. We chose to preserve the signal integrity and let the next stage deal with the dynamic range requirement.

Common Mode Voltage

Amplifiers are composed of MOSFET transistors. To operate correctly, transistors need to be biased into the saturation region. The voltage level of the input signal, which is propagated through the gates of MOSFETs, needs to have a sufficient potential difference with respect to the drain and the source. The drains and sources are tied to the voltage source and ground respectively. This implies that the DC component – the common-mode voltage – of the input signal needs to be lower than the supply voltage by a small margin but higher than ground by a small margin as well. Because we chose high-side current sensing configuration, our common-mode voltage is the same as Vcc. Hence, we introduced a second power supply that delivers 5.5V to bias the amplifier.

Reference Voltage

The reference voltage of a differential amplifier is the baseline voltage to which input and output voltages are referenced. The reference voltage also plays a role in biasing the amplifier and must be chosen carefully. For example, for the amplifier we are using, if the common-mode voltage is around 4V and the supply is 5.5V, the reference input needs to be greater than 3V, as required by the specifications. The exact requirement on the reference voltage varies across different chips. It is desirable to experimentally determine the limits. Reference voltage also allows us to adjust or even cancel out the internal offset so that the output is always positive. The internal offset due to sizing differences and manufacturing variations may be positive or negative. Because the next stage may not cope well with negative voltages, it is desirable to offset this value so that the total offset is always positive. This does not eliminate the need for calibration because the total offset still varies across chips.

Additionally, a reference voltage can give us the flexibility to measure power in both directions (i.e. consumption and recharging). For example, if we set the reference voltage to $\frac{Vcc}{2}$ (see Figure 2.7), we can record the nominal (zero power in either direction) voltage by setting the input to zero and capturing the counter values (or rather how fast the counter increments). A reverse flow in current (e.g. recharging) will simply cause the counter to count slower than this nominal rate. This will cut the resolution in half but provide a signed rather than unsigned value for metering.

Input and Output Offset

Differential amplifiers consists of pairs of CMOS gates. Due to process variations, the sizing of these gates are usually not perfectly matched. As a result, there will be non-zero output even when the input is zero, called offset. There are two types of offset, input offset and output offset. Input offset is multiplied by the gain while output offset is directly added on the amplified signal. Because our application is concerned with very small voltages, even

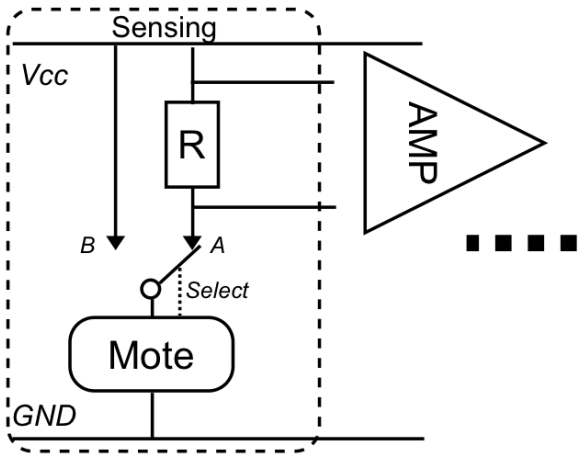


Figure 2.6. SPDT Switch for Calibration. In one configuration, current flow bypasses the shunt resistor, allowing the input offset to be measured. In the other configuration, current passes through the shunt resistor, allowing the mote current to be measured.

a small variation in offset will skew our results. Therefore we need to calibrate the amplifier in order to eliminate the offsets.

There are typically two ways to calibrate offsets. One method involves purely using analog circuits, but this method is expensive both in terms of cost and power. The alternative, and the method we choose, is to do it digitally. As seen in Figure 2.6, we added a digitally controlled switch to the resistor's load. The switch is nominally at A, allowing normal measurement. During calibration, the mote toggles the switch line to B, which results in a zero voltage drop across the resistor. The mote then takes a reading at this configuration before returning the switch to A.

Effect of Noise

As discussed in Section 2.3.1, the differential amplifier is quite sensitive to noise. Figure 2.8 shows the effect of the oscillator on the amplifier before any filtering. To reduce noise, we have placed multiple filters around chip power supplies as seen in Figure 2.7, and separated analog circuits from digital circuits using slotted ground guard rings as seen in Figure 2.2. We also employ separate analog and digital ground planes which meet directly

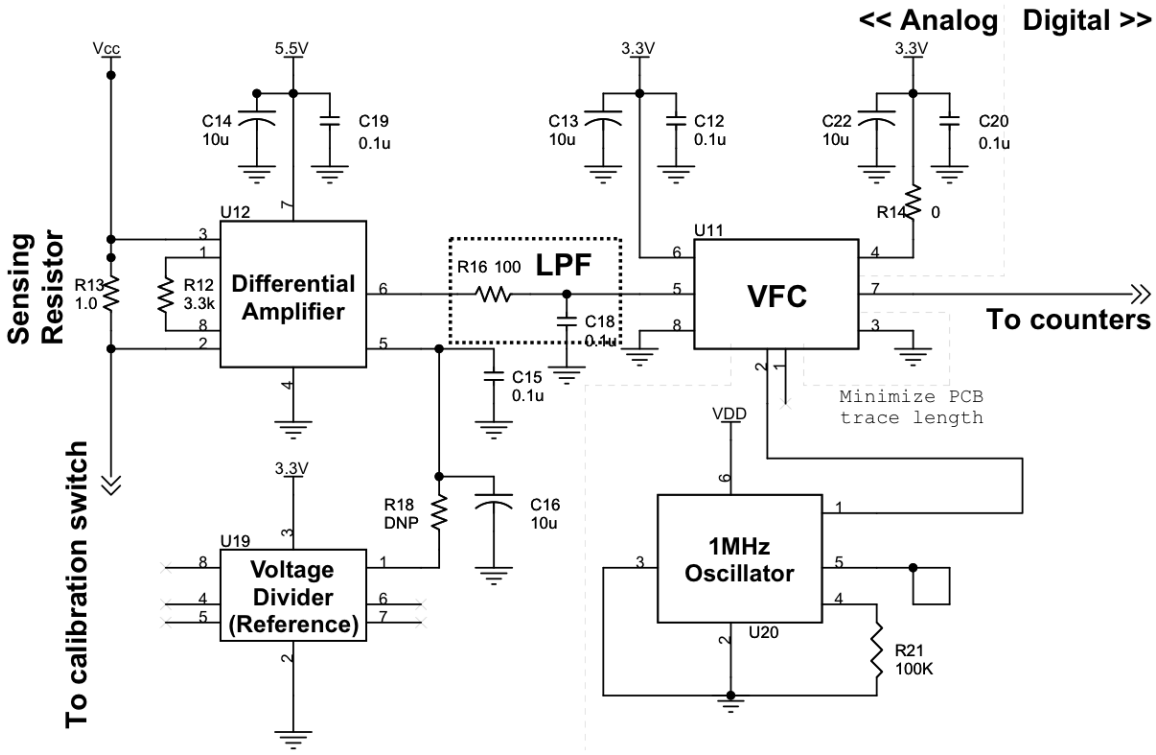


Figure 2.7. A simplified representation of the analog portion of schematic.

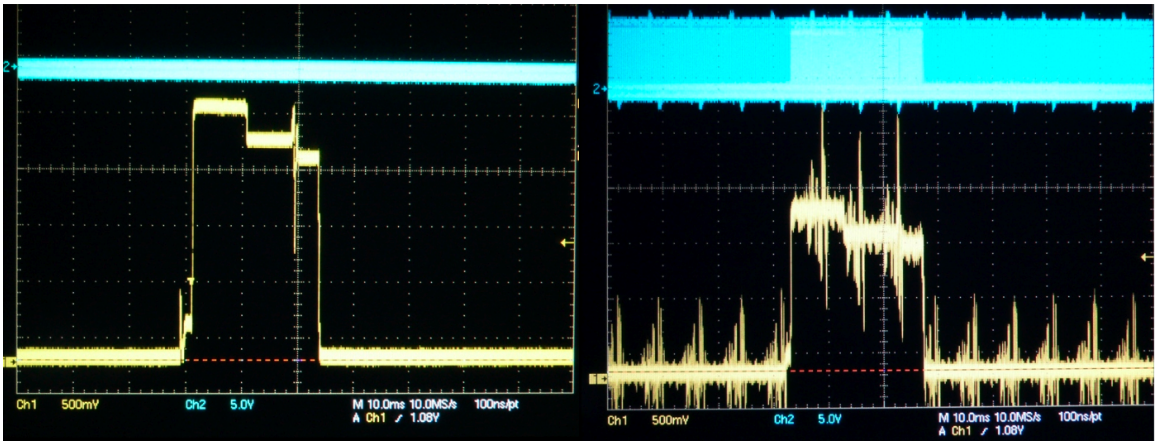


Figure 2.8. Signal is clean before VFC (left). Noise (right) introduced by the VFC oscillator requires analog lowpass filtering, ground guard rings, separate analog and digital ground planes, and decoupling capacitors to contain.

under the VFC. The guard ring slots can be used to attach separate metal shields to the analog and digital portions of the circuit, although we have not yet needed to do so.

Lowpass Filtering

As previously discussed in Section 2.1.2, energy content is present across the entire spectrum. To avoid false readings due to high frequency components in our signal, we cutoff the frequency content at the point where the energy content drops significantly such that it will not adversely affect our energy readings. In our system, this point occurs at 20kHz as seen in Figure 2.3. We place a single LPF between the differential amplifier and the VFC to achieve this purpose, as shown in Figure 2.7.

2.4.3 Digitization

Digitization is the stage that converts the analog signal to the digital domain, allowing digital signal processing (DSP) to be applied and eventually interfaced with a digital system (MCU). This is one of the central stages in most metering systems and is where some of the original signal information is lost due to the finite resolution of digital systems.

Analog-to-digital converters (ADC) are the most commonly used digitization device. ADC takes an input voltage and outputs a digital signal with a specified *bit* resolution. For example, a 12-bits ADC means that for an input range of 0-40mV, the minimum voltage level that can be captured is $\frac{40mV}{2^{12}} = 9.8\mu V$. Furthermore, 12-bits means there are 2^{12} discrete steps over the input range and any value between the steps will need to be rounded.

Our dynamic range requires at least 14-bits of resolution (Section 2.1.1). We considered several different designs:

- Internal ADC of the MCU is inadequate because they are usually only 12-bits. Furthermore, it incurs significant MCU overhead.
- 16-bit ADCs are slightly more expensive but not uncommon. However, one of our goals is to internally integrate power to obtain energy. If our power measurement

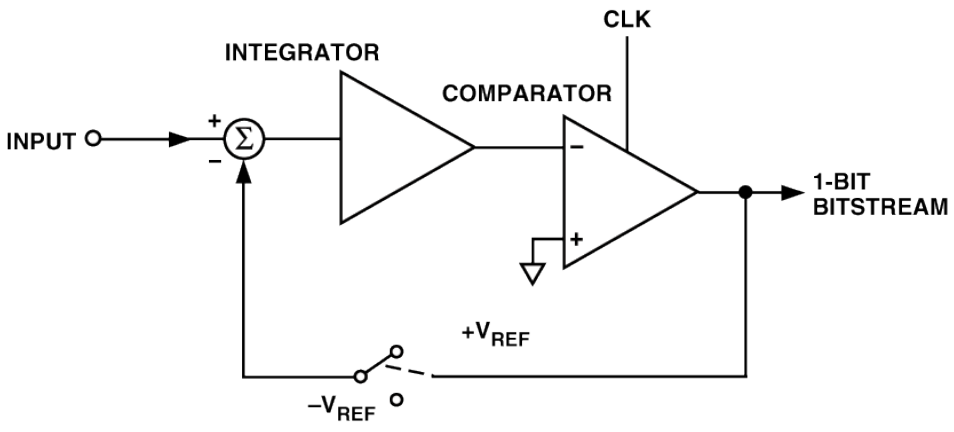


Figure 2.9. The architecture of a VFC.

(output of the ADC) is already in a pure digital format (e.g. parallel bus, I2C, SPI), we will need a compatible device to integrate the digital signal. This implies either a DSP or MCU. A separate DSP or MCU chip adds unnecessary complexity and cost. On the other hand, using the same mote as the integrating MCU would incur significant CPU and power overhead.

- IC chips that integrates 16-bit ADC with counters are essentially energy monitoring ICs. However, as discussed in Section 2.2, we have not found a single IC that satisfies all of our requirements.

- Voltage-to-frequency converter (VFC) is a low cost analog digitization device in the sense that the output is a simple digital-compatible pulse train instead of a full fledged digital bus as in the case of ADC. We further investigate VFC below.

A VFC operates by integrating the input voltage and feeding the output of the integrator to a comparator as seen in Figure 2.9. As soon as the charge accumulated in the integrator exceeds the reference voltage (Vref), the comparator outputs a pulse, which also acts as negative feedback to "balance" the charge inside the integrator. The net effect is a pulse train whose frequency is proportional to the input voltage. It is different from ADC because any arbitrarily small voltage can be captured via the analog integrator. The

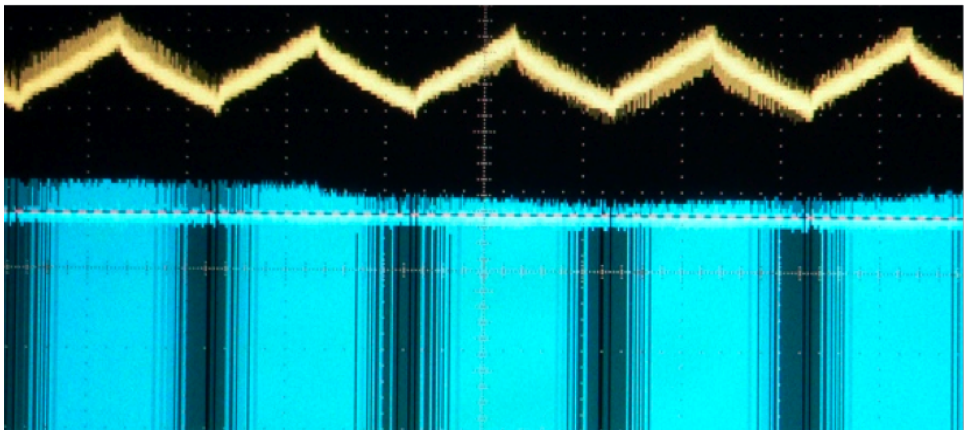


Figure 2.10. The VFC transfer function shows output frequency is proportional to input voltage.

transfer function for the VFC we chose is:

$$f_{OUT} = 0.1 f_{CLKIN} + 0.8(V_{IN}/V_{REF}) f_{CLKIN} \tag{2.1}$$

where f_{CLKIN} is the 1MHz clock input and V_{ref} is 3.3V. A more graphical representation is shown in Figure 2.10.

While the resolution of VFC is infinite in theory since it's analog, the actual fidelity of the signal is limited by the internal noise. The VFC we are using has a maximum peak-to-peak noise of $100\mu V$. Assuming the gain of our amplifier is around 80 and our lowest input voltage is $2\mu V$, the smallest input to the VFC is then $80 \times 2\mu V = 160\mu V$, which is barely above the noise. While this places stringent noise filtering requirement on our design, the actual signal-to-noise ratio (SNR) is not bad since the specified $100\mu V$ is the maximum peak-to-peak noise (as opposed to RMS noise).

Our choice of using VFC is also due its easily-integratable output. As mentioned before, we need to sum the output of the VFC to obtain energy and we want minimal perturbation of the DUT. ADC usually require a DSP or MCU at the output to "interpret" the digital output. If the same mote is used to read ADC's output, it will incur significant overhead because power readings need to be read at a fairly high frequency. To reduce complexity, perturbation, and cost, we favor VFC's elegant pulse train output, which can be easily accumulated using simple counters (see Section 2.4.4 for details).

2.4.4 Energy Output

The last stage is energy accumulation and digital output. To provide the user with a direct energy value, we will need to integrate the power readings from the previous stage:

$$E(t) = \int_{t0}^{t} P(\tau) d\tau$$

If the previous stage uses an ADC, we will need a DSP or MCU, which either incurs significant complexity/cost or perturbation, as explained in the previous section.

Fortunately, we can use simple counters to sum the pulse train output from the previous stage. Every rising or falling edge of a pulse in the pulse train increments the counter by one. Higher power consumption leads to more frequent pulses, which in turn leads to faster counter increments. The difference in counter values between t_0 and t_1 represents the energy consumption during time $t_1 - t_0$. However, counters have a finite range, represented by its bits. To prevent counters from overflowing, the MCU needs to read and reset the counters periodically. This presents overhead, but it can be minimized by using a large counter that overflows infrequently. We choose to use a 32-bit counter. From equation 2.1, we find that the VFC has a maximum output frequency of 0.9MHz (assuming VFC is maximum when $V_{IN} = V_{REF}$ and $f_{CLKIN} = 1MHz$). This implies that the counter will overflow roughly once every hour (see Section 2.3.3), which is infrequent.

Finally, for the user of SPOT (e.g. the mote) to read the counter values, we need a compatible digital output format. Parallel output is not possible because 32 bits require too many pins. Serial output such as I2C or SPI is desired. Since our data rate is low, I2C is more than sufficient.

Additionally, to relieve the MCU of time-keeping responsibilities and to precisely measure elapsed time, we included another 32-bit counter dedicated to time-keeping and is triggered by the same oscillator that drives the VFC. Please refer to Section 2.3.3 for preliminary descriptions. For this scheme to work, we need some way of reading the two counter chips at the exact same time. Fortunately, one valid command to the counters that we use is "capture", which essentially takes a snapshot of the counter values and stores them in registers. Furthermore, because the addresses for the counter chips can be dynamically

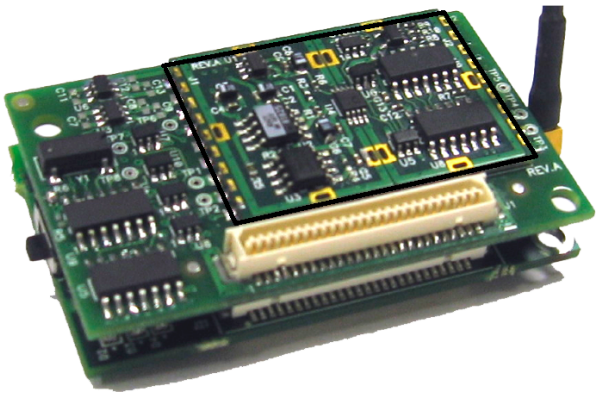


Figure 2.11. The SPOT module attached to its MicaZ carrier board.

changed, we can wire the I2C addresses for the two chips to be the same during a "capture" command, then re-wire the addresses to be different and read back the captured values individually.

2.5 Carrier Board

SPOT, as described in this paper, is a single-sided, leadless chip carrier (LCC) that can be treated as a modular circuit component and directly soldered to a printed circuit board. To be useful, the module must intercept the power line between the power supply and the node under test. We have built a Mica2/MicaZ carrier board that incorporates the SPOT module, all needed power supplies, a real-time clock, extra counters, and power line intercept, as shown in Figure 2.11. A Telos [1] version of the carrier board is under development. The testbed described in Section 4.1 is instrumented with these modules.

Chapter 3

System

In this chapter we present the calibration process, driver architecture, and evaluation of SPOT. More specifically, we focus on the system issues related to obtaining accurate power and energy measurements in the application layer. We further ground our system by using SPOT to profile simple sensornet applications and interpret their results.

We begin this chapter by evaluating SPOT using a series of micro-benchmarks, in accordance to the original requirements we outlined for SPOT. After that, we present a TinyOS-2 [16] implementation of our driver, responsible for communicating with the I2C interface of SPOT, and translating raw counts into standard energy units that can be easily integrated in applications. An important function of the driver is to calibrate SPOT, both at boot time to zero out device-specific variances and to apply a predetermined calibration function. We describe this process in a separate section. In the following section, we measure the power and energy profile of an application based on Blink using the SPOT system, and macro-benchmark its results.

3.1 Single Node Evaluation

In this section, we evaluate SPOT's accuracy as a tool for metering energy and power for a typical sensornet workload. In particular, we evaluate SPOT's dynamic range, resolution,

and stability, which represent the most challenging requirements for an embedded power meter.

3.1.1 Dynamic Range and Resolution

A principal requirement of our system is a dynamic range exceeding 10000:1 with current resolution of at least $2\mu A$. To evaluate SPOT's resolution, we loaded a 3.3V power source with resistors of different value in the $M\Omega$ range to create currents of 1.08, 2.07, and $3.48\mu A$. In addition, we loaded the power source with a TelosB mote in sleep mode, which draws $9.09\mu A$, to provide an additional data point.

Table 3.1. Snapshots of SPOT's uncalibrated, free-running Time and Energy counters taken approximately every 40 ms (first two columns).

Time Count	Energy Count	Counts/Sec
397369588	2535884271	161804.7903
397424011	2535893077	161806.5891
397453917	2535897916	161806.9953
397489646	2535903697	161801.3378

For each data point, we take 600 consecutive readings from SPOT. An example of the uncalibrated readings that SPOT output is shown in Table 3.1 in the first two columns. The first column lists counter values from the Time Counter and the second column lists counter values from the Energy Counter. The elapsed time between readings is the sum of the I2C access time (I2C is running at 100kHz) and the radio packet transmit time (one packet is transmitted for each reading).

For simplicity, we will refer to values in the table using the initials of the column heading plus the row number in subscript (starting at 1). For example, the second Energy Count is designated as EC_2. The uncalibrated data shown in Table 3.1 can be used in several ways. For example, we can determine the energy the mote consumed between the 1st reading and the 4th reading simply by computing $EC_4 - EC_1$; this energy corresponds to an elapsed time of $(TC_4 - TC1)/10^6 = 0.12$ seconds, since the Time Counter increments

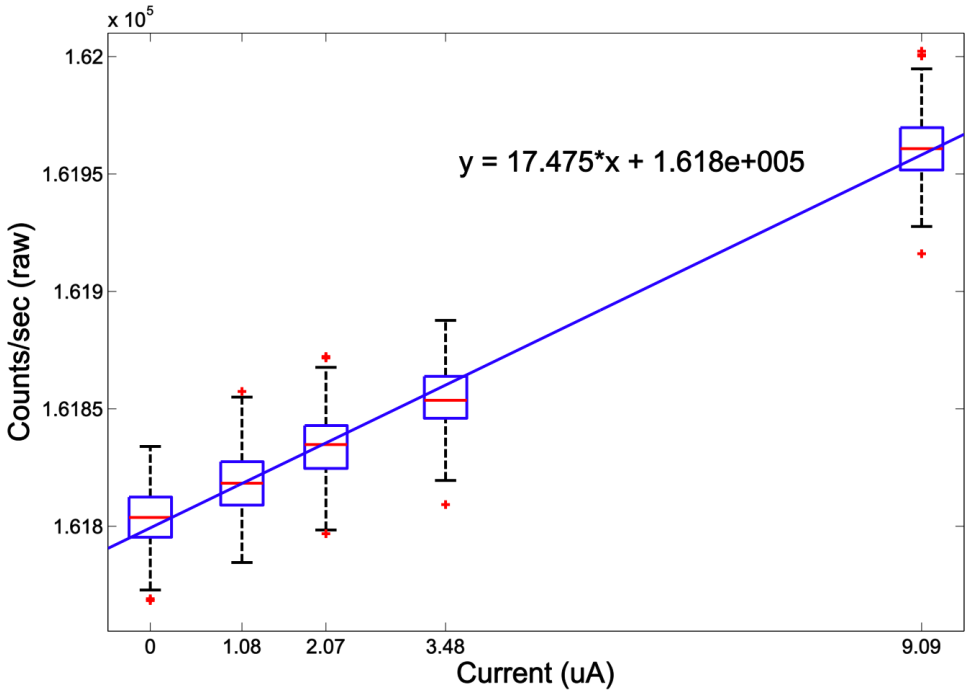


Figure 3.1. SPOT current resolution. SPOT can resolve currents at the microamp level.

at 1MHz. Since power is the derivative of energy, we can estimate the average power over an interval by simply taking the difference of the energy counts. The third column shows the power in units of counts/second. For example, C/S_2 is obtained by computing $(EC_2 - EC_1)/(TC_2 - TC_1) \times 10^6$. This value is linearly proportional to power and current, since voltage is constant.

Power values (counts/sec) for each current are plotted in Figure 3.1 in boxplots to show their distributions. The power for $0\mu A$ current is not zero due to the amplifier non-zero offset voltage and the VFC minimum frequency of 0.1MHz.

Notice that while values between μA boundaries do overlap, the variance is limited and the medians values sit at regular intervals from each other. This suggests that SPOT is able to resolve $2\mu A$ or smaller currents, but that it is necessary to take multiple samples. In practice, the counter reading rate is not likely to be low, which effectively averages the readings over much longer runs.

A simple linear regression of the five medians shown in Figure 3.1 generates a useful

calibration curve for SPOT. Despite being far from optimal, since this curve is generated using data for points between 0 and $9.09\mu A$ rather than the full range extending to 45mA, we show that this curve is still useful in the next section.

Because our Vcc is 3.3V and our amplifier has a gain of around 66, we can tolerate a maximum input current of 45mA, assuming the amplifier offset does not exceed 0.3V. At the other extreme, Figure 3.1 shows that we can resolve currents at $1\mu A$ or even less. This means that SPOT has a dynamic range exceeding $\frac{45mA}{1\mu A} = 45000 : 1$, which surpasses our dynamic range requirement.

3.1.2 Long-Term Tracking Accuracy

Because motes spend the majority of their time sleeping, it is important to evaluate SPOT's accuracy in monitoring a mote's energy consumption during its sleep state. This is more difficult than monitoring a mote in active state because the current to be monitored is four orders of magnitude smaller.

The counter readings obtained from SPOT were calibrated using the equation found in Section 3.1.1 and compared with a reference curve obtained using an professional-grade current meter, as shown in Figure 3.2. The mote under observation, a TelosB [1], is drawing approximately $9\mu A$ of current.[1]

As seen in Figure 3.2, SPOT is able to closely track energy usage, but with a small drift. After 6 minutes, the error is less than 0.1mJ or 3%. The absolute error accumulates over time but the relative error should stay constant, which may be acceptable for most applications. If more accurate calibration is required, additional calibration points will be needed.

3.1.3 Energy and Power Tracking

Most sensornet applications are duty-cycled, waking up occasionally to take a sensor sample or send a message, and sleeping the remainder of the time. To evaluate SPOT's

[1] $9\mu A$ represents a typical sleep state current for TelosB powered at 3.3V voltage supply.

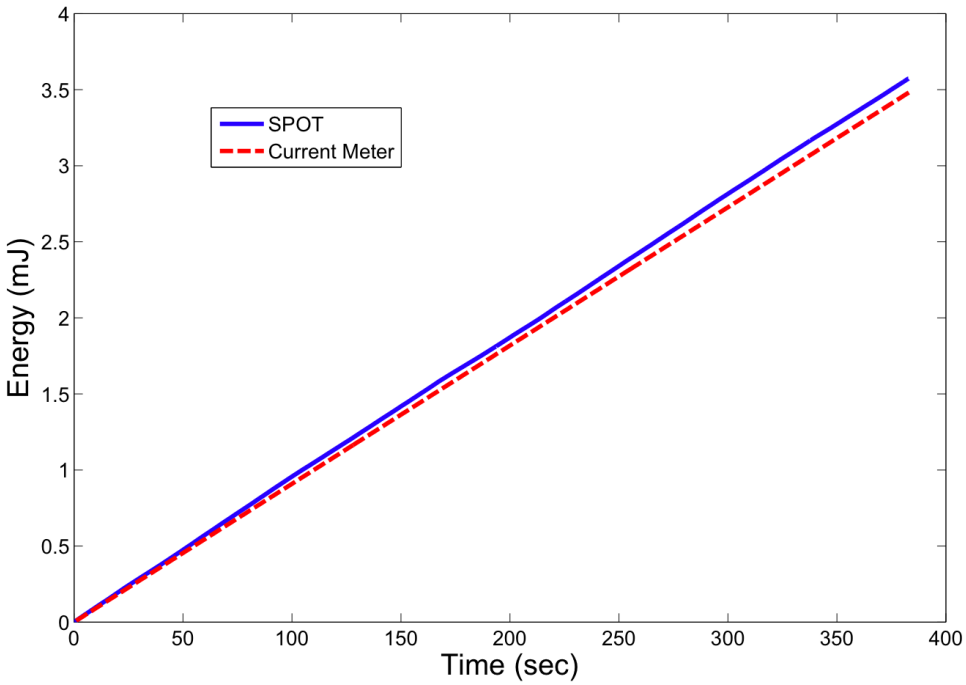


Figure 3.2. Energy metering of $9.09\mu A$ load over a period of time exceeding 7 minutes. The accumulated error is 0.1mJ or 3% of the actual energy usage.

tracking accuracy in monitoring motes with relatively low duty-cycles, we now consider a typical workload. The mote under observation is a TelosB mote running at slightly higher than 2Hz duty-cycle. The reference curve is generated by integrating the measured power signals collected using a high-end digital storage oscilloscope. Because the oscilloscope does not have enough dynamic range to resolve active state power and sleep power at the same time, we will focus on only the active state power in this experiment, since the previous section demonstrated SPOT's ability to track sleep state power.

The top graph in Figure 3.3 shows the energy monitored by SPOT and the oscilloscope. The flat portions of the curve represent energy consumed by the mote during sleep while the sharp rises between flat steps represent energy consumed during active cycles. The change in energy during active cycles is approximately 0.4mJ while there is no observable energy change during sleep states. An interesting observation is that energy consumed during different active cycles are all slightly different. This is likely due to random backoff in the medium access control layer, which causes the variations in active cycle time. This

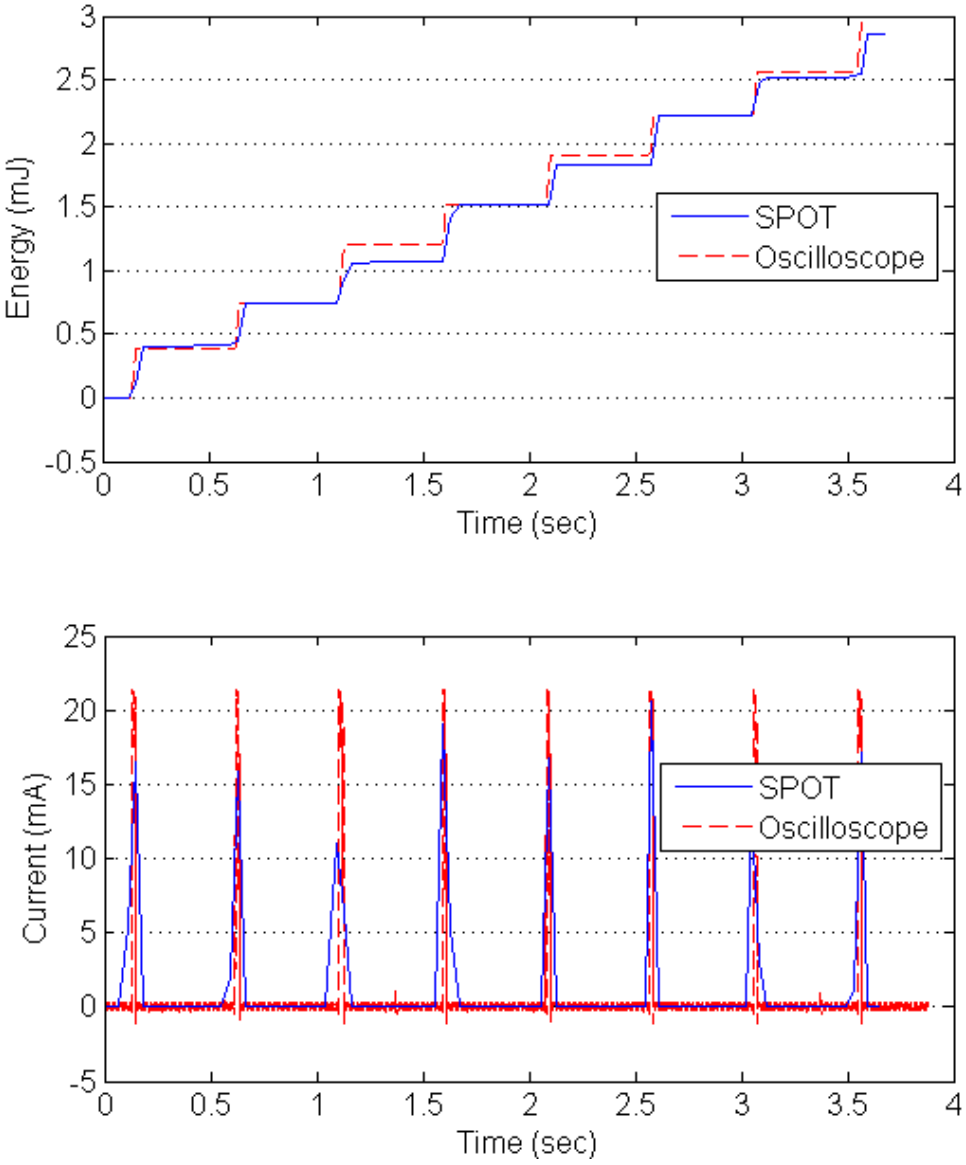


Figure 3.3. Energy and power tracking of duty-cycling TelosB mote using SPOT compared with a digital storage oscilloscope. The flat portions of the top graph represent energy consumed by the mote during sleep while the sharp rises between flat steps represent energy consumed during active cycles. The bottom graph shows the power draw recorded by SPOT by differencing successive energy usage readings.

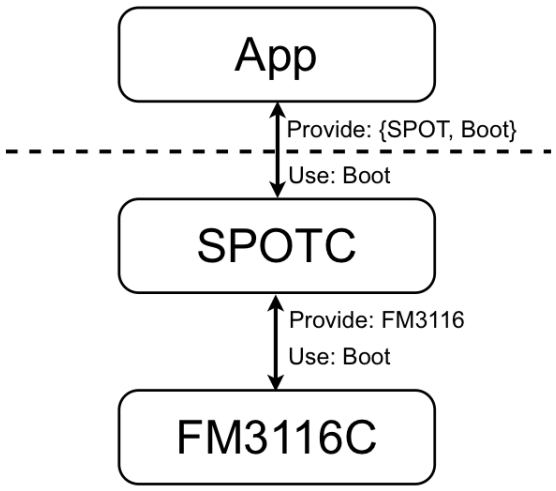


Figure 3.4. The multi-layered driver architecture abstracts complexities from the application and allows code-re-usability.

also explains why SPOT does not match the oscilloscope more closely, since the SPOT and oscilloscope measurements are taken at different times.

The bottom graph of Figure 3.3 shows the power draw recorded by SPOT.[2] Note that the observed resolution is quite low. This is because we wait for the radio on every sample. However, because SPOT samples the signal at 1MHz internally, even-though we are not reading the counters as fast, the energy measured is still quite accurate.

3.2 Driver and API

We implemented the driver for SPOT in TinyOS-2, an event-driven operation system designed for sensor networks. We provide a high-level application programming interface to SPOT to make using it simple for TinyOS developers, shown in Section 3.2.1. We use a multi-layered driver architecture to manage complexities and to allow code-reusability, similar to the hardware abstraction architecture proposed by Handziski et. al in [17]. At the bottom layer, *FM3116* is responsible for communicating with the FM3116 CPU companion chip on SPOT to accumulate raw counts. Above it, *SPOT* is responsible for abstracting

[2]The terms "power" and "current" are used interchangeably because the voltage is constant and therefore power is directly proportional to current.

the functions of FM3116, initializing SPOT, applying calibration function, manage ports on the MicaZ [18] carrier board, and ultimately provide the upper layer application with a simple API, as shown in Figure 3.4.

3.2.1 Application Programming Interface

The application programming interface provides the application programmer with a set of simple commands. To read the energy value from SPOT, simply invoke the following split-phase command:

```
command error_t sample()
```

This command will invoke a series of I2C transfers on the SPI bus of the MicaZ mote and perform the following actions:

```
Read both energy and time counters at the same time
1. Configure addresses to be the same
2. Issue I2C command to capture counter values
3. Configure addresses to be different
4. Read from time counter
5. Read from energy counter
Apply calibration function
```

At the end of these events, SPOT driver will signal the *sampleDone* event with the energy and timer values:

```
event void sampleDone(uint32_t time, uint32_t energy)
```

The speed of I2C bus on the MicaZ is 100kHz, this leads to about 2.5ms to perform each energy readings.

The application can directly use the energy value from the return event as the current energy of the mote; alternatively, the application can sample SPOT at different times and obtain the energy consumed during that interval using simple subtraction. Furthermore, by dividing the timebase from the energy values, one can obtain the power profile of the application. We will demonstrate this in Section 3.4.

Due to the high fidelity requirements of SPOT, even a small variation in the sizing of discrete components, such as resistors and capacitors, would cause a noticeable change in the measurement. To mitigate this effect, calibration is required (described in more detail in Section 3.3), in addition a short initialization process. But since this process needs to take place before all other application code is run, we have decided to include initialization and calibration in the *Boot* event used by the SPOT driver, while providing a separate *Boot* interface for the actual application code to use. This essentially bootstraps the application after the initialization and calibration of SPOT.

3.2.2 Driver

As shown in Figure 3.4, SPOT driver consists of two modules - FM3116M implements the driver for the CPU companions (used as counters with I2C output) while SPOTM implements the SPOT API.

FM3116 module implements the FM3116 interface which provides a subset of the functions provided by the FM3116 CPU companion chip:

```
command error_t init();
event void initDone();
// Take a snapshot of the counters
command error_t takeSnapshot();
event void snapshotDone();
// Return counter value from earlier snapshot
command error_t readCounter_Only();
event void readOnly_Done(uint32_t cnt);
// Differs from readCounter_Only in that
// readCounter will RE-TAKE SNAPSHOT
command error_t readCounter();
event void readDone(uint32_t cnt);
command error_t CAL();
event void calDone();
command error_t writeReg(uint8_t addr, uint8_t val);
event void writeReg_Done();
command error_t readReg(uint8_t addr);
event void readReg_Done(uint8_t val);
```

FM3116M uses the *Resource* component to arbitrate for access to MicaZ's SPI bus, since the SPI bus might be shared by other users, such as the upper-layer application itself.

The FM3116 driver essentially implements a number of split-phase state machines, one of which is responsible for reading the counter register from the chip. This state machine is, in turn, used by an over-arching state machine that invokes the smaller state machine twice to read both the counter for energy and the counter for the time-base.

Upon receiving a *readCounter()* command, the SPOT driver must first request access to the I2C bus. Upon a successful grant, an I2C command is issued to the FM3116 chip telling it to latch the current energy and timer content into two registers; upon its completion, another resource request is made for the I2C bus; when granted, the SPOT driver will issue two separate I2C read commands at the addresses of those two registers. If they are successful, two 32-bit values for energy and timebase will be delivered to the upper layer driver (SPOTM) via the *readDone()* event.

SPOT module implements the SPOT interface which provides the above-mentioned API. *SPOTM* uses the *FM3116* interface to communicate with the FM3116 chip to obtain the energy and timebase values via I2C. A calibration function is applied to the result from FM3116 to provide the application with standard metric energy values, as described in Section 3.3.

In addition, *SPOTM* implements (provides) the Boot interface to bootstrap the boot-time calibration and initialization process. This is also implemented by a state machine that performs the following actions:

```
1. Short the measurement pins to SPOT, creating a 0V reference input.
2. Wait for 5 seconds for the internal capacitor of the amplifer to discharge.
3. Record the energy and timebase.
4. Wait for 1 second.
5. Record the energy and timebase again to obtain the power for 0V input.
6. Return measurement inputs to normal.
7. Wait for 5 seconds for the internal capactior to charge again.
8. Signal Boot.booted event to trigger actual application to run.
```

At the end of the bootstraping process, two sets of $\{energy, timebase\}$ are recorded inside the driver and can be read out during the application's Boot.booted() event.

Because SPOT is mounted on the MicaZ carrier board which integrates other feathers, *SPOTM* has to configure any connected ports to the microprocessor in the correct config-

urations. Lastly, while performing a SPOT measurement requires waking up the processor from sleep, the driver is designed to return the processor to the previous state in order to minimize perturbation of the actual application.

By implementing calibration in a separate boot sequence, the SPOT driver can be easily integrated into applications, requiring a simple "sample()" command followed by a "sampleDone()" event. The overhead of this driver is essentially the cost of I2C transfers, which is around 2.5ms per sample. For applications which are only concerned with energy consumptions over relatively large timescales (seconds), this overhead is negligible. However, for applications which requires high frequency power profiles (on the order of milli-seconds), the overhead is significant.

3.3 Calibration

Similar to most real sensors, SPOT requires a calibration function to convert raw *counts* of the energy and timebase counters to standard energy units. As shown in Figure 3.5, SPOT is a linear time-invariant system, with a transfer function composed of sums and products. The overall transfer function of SPOT is linear with only two terms - a gain and an offset.

Instead of trying to obtain the coefficients at each stage, we choose to calibrate SPOT as a whole system to avoid inducing errors in the intermediate stages. Traditionally, one would excite the system with a set of known inputs and observe the corresponding outputs to compute the two coefficients. In the case of SPOT, the inputs are predetermined set of workloads that generate known power consumptions. Unlike the method used in Section 3.1, we cannot directly apply a current across the input pins of SPOT since we want to include the path between the power source of the mote to the input pins of SPOT in the calibration curve as well.

To generate the set of known workloads, we program the mote with a set of different programs that varied in hardware resource usages, such as turning on and off LEDs and radios. To obtain the true power consumption of these workloads, we had to use an oscilloscope. This is because the output of SPOT will include both the nominal power

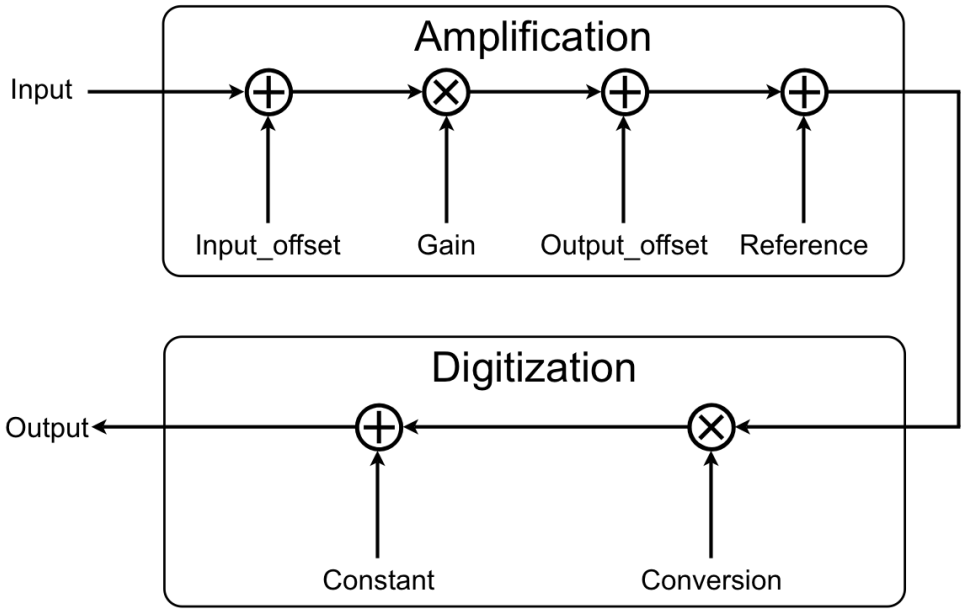


Figure 3.5. Simplified SPOT system block diagram.

consumption generated by the workload *and* the energy used by itself to sample the inputs (i.e. energy for I2C transfers). We need to capture this energy when establishing our true power consumption. And indeed, we can readily observe a small spike in current on the oscilloscope when SPOT tries to sample the input. We integrate power observed on the oscilloscope and compare it to the output of SPOT to calibrate SPOT.

Using least-mean-squared method, we obtain the gain to be 11.628 and offset to be 497220.

However, due to variances in devices, this gain and offset will vary slightly across devices. Figure 3.6.A shows that the energy and power across different motes for the same workloads are slightly different. To mitigate this effect, we perform boot-time calibration by zeroing out the input for each mote and measure the output, to obtain a device-specific offset. This process is described in more detail in Section 3.2.2. We show that this boot-time calibration is able to reduce the error by more than 26%, as shown in Figure 3.6.B. We do not perform device-specific calibration for the gain coefficient at this time, which explains the small variance that still exists post-calibration. However, we plan to add boot-time calibration

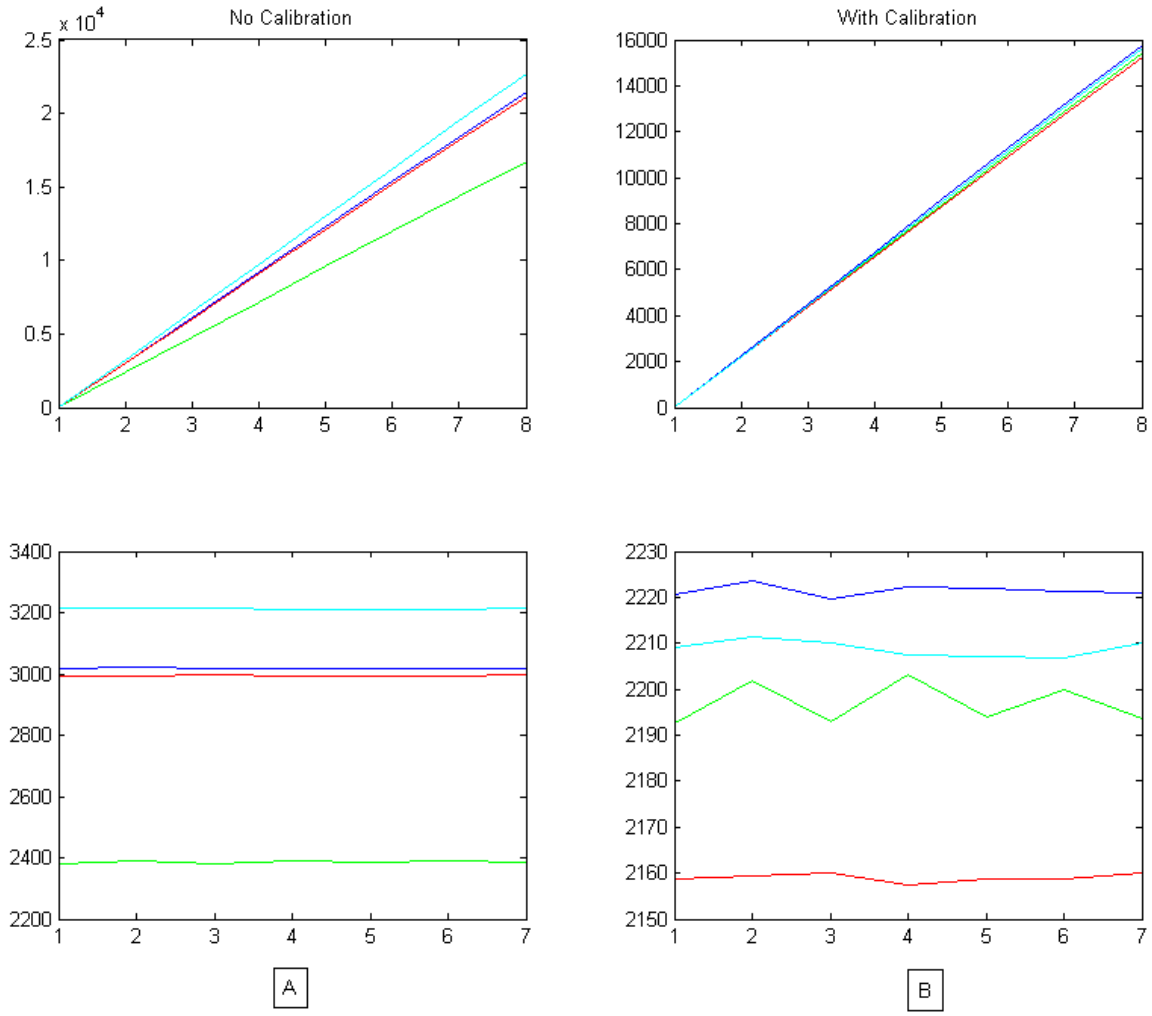


Figure 3.6. The energy and power measurement from SPOT (running on several different nodes) for a constant workload before (A) and after (B) per-mote calibration.

for the gain coefficient in the near future by measuring a set of predetermined workloads and compare with their (predetermined) true powers.

3.4 Energy-Aware Application Example

To demonstrate how easily it is to enable power monitoring using SPOT, we use SPOT to meter a simple application that blinks periodically.

We start by defining the interval at which we wish to sample the energy readings and wire up the SPOT interface:

```
#define SAMPLES 10
#define INTERVAL 1024

module BlinkSPOTM {
        ...
        uses interface Timer<TMilli> as Timer;
        uses interface SPOT;
}
```

In its implementation, we create two arrays to store the timebase and energy values. Inside the Boot.booted() event, we read out the calibration values that SPOT measured during bootstrap. We also start the application and the timer to sample SPOT at the end of booted():

```
implementation {
        ...
        uint32_t base[SAMPLES];
        uint32_t measure[SAMPLES];
        ...
        event void Boot.booted() {
                // get the calibration data out
                base[0] = call SPOT.getCalTime1();
                base[1] = call SPOT.getCalTime2();
                measure[0] = call SPOT.getCalEnergy1();
                measure[1] = call SPOT.getCalEnergy2();
                // start blinking
                call Timer.startOneShot(512);
                // start SPOT sampling
                call SPOTTimer.startPeriodic(INTERVAL);
        }
```

When it's time to sample SPOT, simply call *SPOT.sample()*:

```
        event void SPOTTimer.fired() {
                if (index < (SAMPLES)) {
                        call SPOT.sample();
                } else {
                        index2 = 0;
                        // finished sampling, send date out via radio
                        call AMControl.start();
```

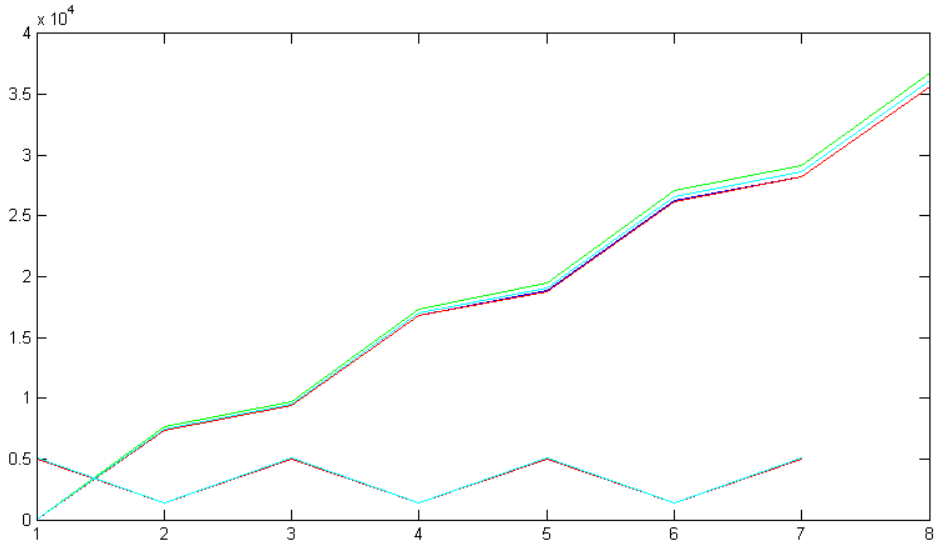


Figure 3.7. The energy and power measurement from SPOT (running simultaneously on several nodes) for a dummy application that turns on and off three LEDs at 2sec interval.

```
        }
}
```

When each SPOT reading is finished, the driver will trigger *SPOT.sampleDone()* with the timebase and energy. The application can then record them:

```
event void SPOT.sampleDone(uint32_t time, uint32_t energy) {
        base[index] = time;
        measure[index] = energy;
        index = index + 1;
}
```

The application can of course make decisions using these energy values at run-time, as proposed in more detail in Chapter 5. In our dummy application, we simply store them and send them to a base-station via radio. These energy values are plotted in Figure 3.7. The monotonically increasing line is the energy consumed by the mote while the oscillating line is the power (by differentiating the energy), normalized to be shown on the same scale as energy. As we can see, the power and energy reflects the energy used for powering the LEDs.

Chapter 4

Network

Although the SPOT system itself represents a unique, mixed-signal, hardware-software design, its true potential lies in its ability to easily scale up to large networks. A network of motes equipped with SPOT will enable researches that are previously not possible.

Recall that state-of-the-art simulators like PowerTOSSIM base their models on micro-benchmarks of a single node taken in a particular environment. An instrumented testbed will allow us to answer a number of questions about the generality of these models: How representative is the node that was instrumented to calibrate PowerTOSSIM's model? How does interaction with the physical environment shape energy usage? How do temperature variations affect leakage currents?

Beyond the simple verification of mote power models, a network of SPOT-enabled motes can be used to macro-benchmark the energy-efficiency claims by existing network protocols as well as implementations of similar or identical protocols from different TinyOS distributions.

A SPOT-enabled testbed could also aid in the development of energy-aware applications by allowing application writers to visualize dynamic energy consumptions during development time.

In this section, we will describe the testbed setup as well as data acquisition and parsing

of SPOT-generated data. We will also use CTP [19] as an example to demonstrate SPOT's ability as a tool for investigating network issues and as an aid for designing network protocols.

4.1 Testbed Setup

We have built a testbed consisting of 78 MicaZ motes, half of which are SPOT-enabled, and are all connected via Ethernet back-channel. These motes are deployed on the 4th floor of the Computer Science building in UC Berkeley, as located in Figure 4.1. Every mote is connected to a Ethernet back-channel via MIB600 boards which essentially convert them into IP devices, allowing them to be programmed remotely using IP addressing. Data sent by motes via UART are also fed to the Ethernet back-channel.

For SPOT-enabled motes, the SPOT board is installed between the MicaZ mote and the MIB600 board, allowing SPOT to intercept and measure power.

There is currently no reservation system. One can simply upload the binary files to the server and invoke a script that programs a customized set of motes in parallel. To receive data from motes, another script can be used to tunnel all the UART data to the user's standard output.

4.2 Data Acquisition and Parsing

One typical usage of SPOT is to measure energy and power consumption of the application at the network level and analyze them later. To collect SPOT measurements from our testbed is a straight-forward process.

There are multiple ways to acquire and parse data. In our example, we simply log the raw UART traces from all motes into a single file, which looks similar to:

```
...
1 12207 00 FF FF 00 00 0A 00 0D 00 00 2B 46 50 2E F4 7B 75 A3
1 12208 00 FF FF 00 00 0A 00 0D 00 00 AB C6 5F 2E FE 36 7D A3
```

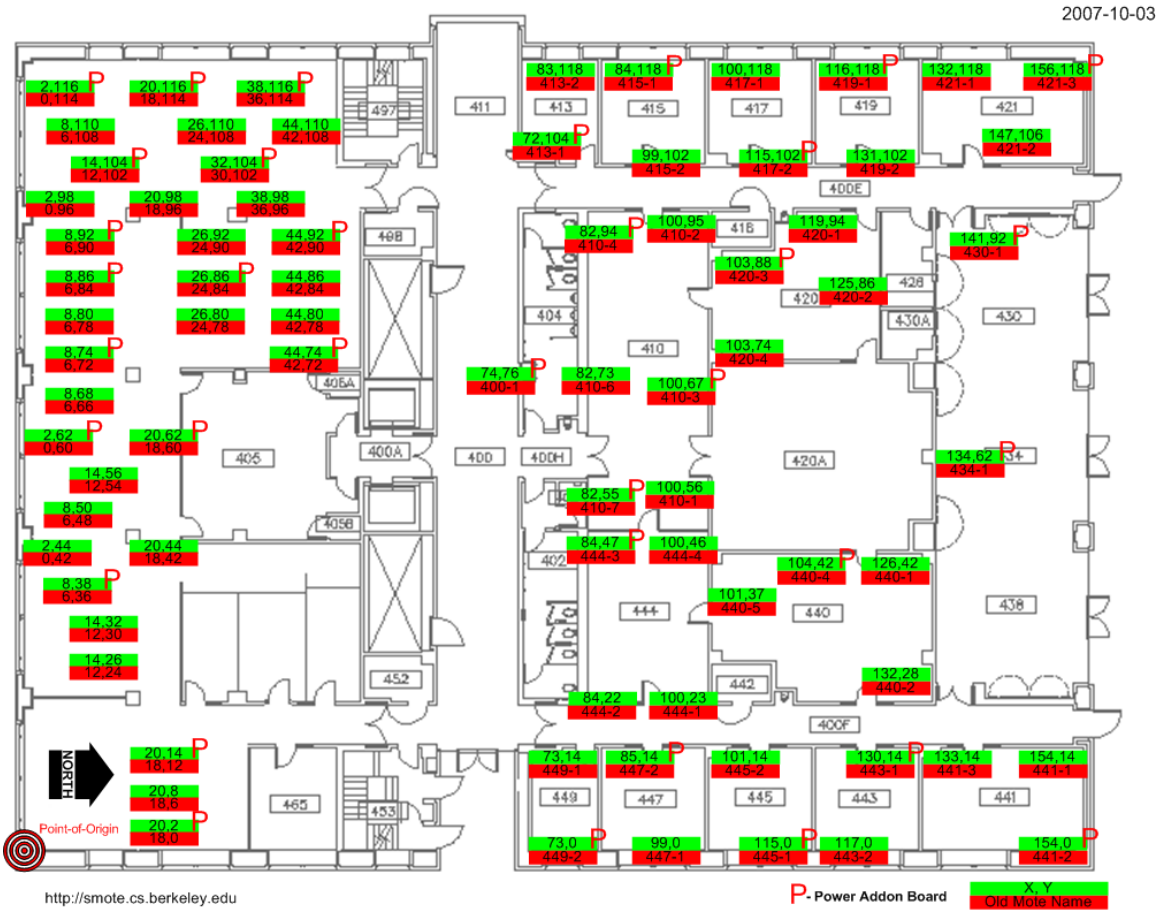


Figure 4.1. 78 MicaZ motes are deployed in the 4th floor of Soda Hall at Berkeley. Roughly half of them are SPOT-enabled, as indicated by letter P.

```
1 12208 00 FF FF 00 00 0A 00 0D 00 00 05 39 BC 2E 90 AF AD A3
4 12216 00 FF FF 00 00 0A 00 0D 00 00 CF 72 82 78 F9 5A 9E 09
4 12216 00 FF FF 00 00 0A 00 0D 00 00 6C DE 91 78 1F 1F A6 09
3 12221 00 FF FF 00 00 0A 00 0D 00 00 82 85 76 E1 90 9B 2F 77
4 12225 00 FF FF 00 00 0A 00 0D 00 00 DC D6 ED 78 CC C7 D6 09
...
```

And since we know the header format for TinyOS 2 packets and we constructed the payload ourselves, we can easily decode the data using a simple script. An example of the decoded SPOT measurements are shown below.

```
...
1 12207 777012779 2742385652
1 12208 778028715 2742892286
1 12208 784087301 2746068880
4 12216 2021814991 161372921
4 12216 2022825580 161881887
3 12221 3782641026 1999608720
4 12225 2028852956 165070796
...
```

The first column is the node ID; second column is the timebase; third column is the energy counts; and fourth column is the power.

This data can then be used in other analysis and/or visualization tools to correlate to other data traces from the application.

4.3 An Example: CTP

Many existing sensornet MAC and network protocols try to optimize for energy usage. However, none have been able to empirically evaluate their claims. One of SPOT's goals is to measure power and energy empirically, in-situ and at scale. In this section, we use SPOT to measure the energy consumption of CTP, the default routing protocol in TinyOS version 2, in order to demonstrate SPOT's ability to help visualizing the energy efficiency of network protocols.

In this application, every node except the root is sending periodically to the base-station at a rate of 0.5Hz. Because motes span the entire 4th floor of Soda Hall in UC Berkeley,

a large subset will have to reach the base-station via other nodes, forming a multi-hop network. In this experiment, we used the default radio stack of TinyOS 2. The MAC layer used in the default stack is not low power, as a results, the radio is not duty-cycled and will stay on the entire duration. From a power point of view, this will make it harder to distinguish which mote is performing more radio activities since the baseline power is already very high. Resolving the different amount of routing activities without duty-cycling will be an interesting challenge for SPOT.

We programmed 18 nodes with this application (node ID is mapped to geographical coordinates as shown below). Mote 1 is the root / receiver.

```
mote 1 ms-4-103-88
mote 2 ms-4-100-67
mote 3 ms-4-73-0
mote 4 ms-4-154-0
mote 5 ms-4-141-92
mote 6 ms-4-20-14
mote 7 ms-4-38-116
mote 8 ms-4-8-92
mote 9 ms-4-44-92
mote 10 ms-4-20-62
mote 11 ms-4-2-62
mote 12 ms-4-8-86
mote 13 ms-4-8-38
mote 14 ms-4-14-104
mote 15 ms-4-20-116
mote 16 ms-4-26-86
mote 17 ms-4-44-74
mote 18 ms-4-8-74
```

Using the approaches described in Chapter 3, we collected the energy and power measurements as shown in Figure 4.2.

We can make a few immediate observations simply by studying this graph. First of all, we can see that without a duty cyclable MAC (e.g. low-power-listening MAC), there are little differences between sending, receiving, and not active at all. Even-though we are sending at a rate of 1 packet per 2 seconds, we cannot really see significant change in power over time. However, the power traces do exhibit periodicity at a frequency of around 2Hz, which is four times the rate of the application sending speed. More interestingly,

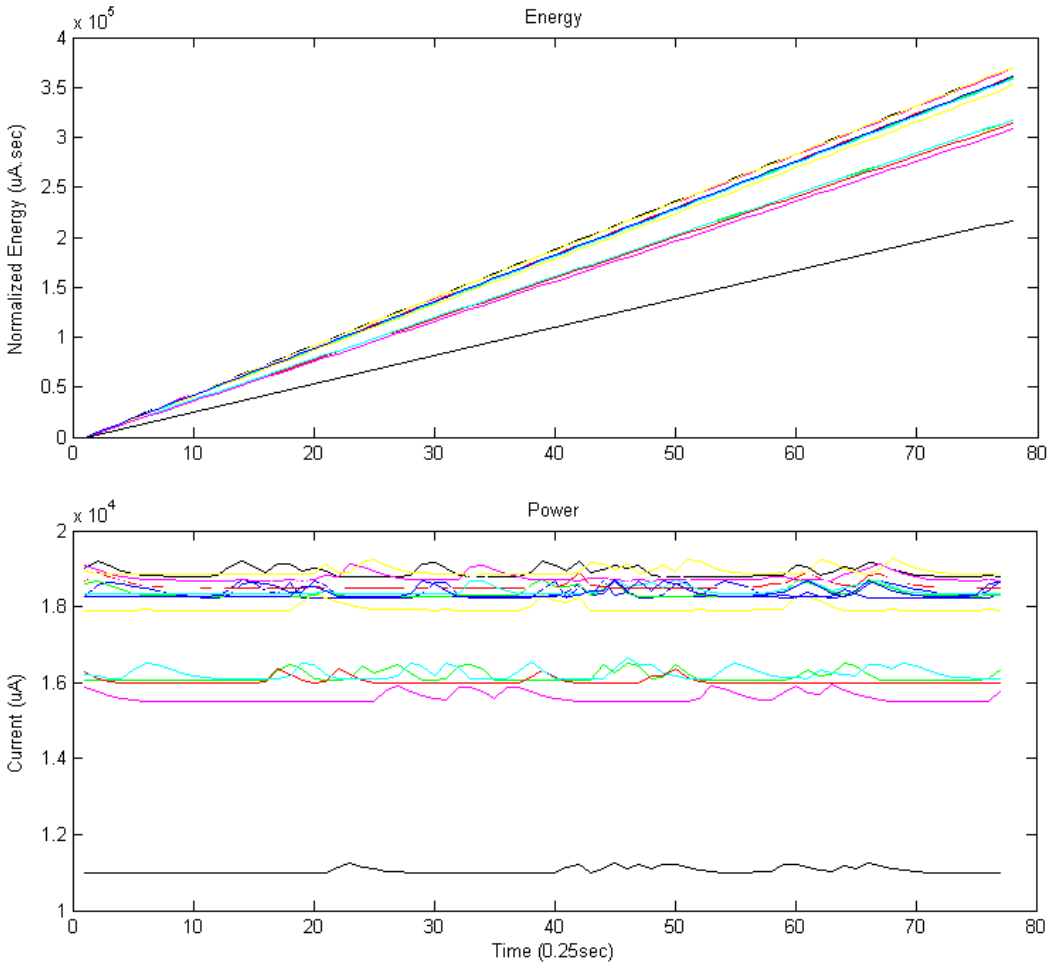


Figure 4.2. Energy and power measurements from 18 motes running simple collection application based on CTP (non-LPL)

the periodic behavior for many motes are matched in phase, which can be explained as a packet been routed up the routing tree, crossing several routing motes in the process. Observations of variations across motes are also very interesting. The power and energy for these 18 motes are largely bi-modal, with two possible outliers. One possible interpretation for the bi-modality is that intermediate motes have the extra burden of routing the packets from their children, thus consuming more energy and falls in the higher energy mode. This interpretation agrees to our general assumption that the power of each node in a routing tree should correlate to its depth in the tree. This hypothesis also suggests that we should be able to separate radio traffic generated by local sensing from traffic generated by routing. A different interpretation for the bi-modality could be that the temperature gradient across the floor influences the power consumed and separates the motes into two power groups. However, without application dependent debugging data, we cannot pinpoint the exact cause. This also suggests that in order to efficiently utilize SPOT's measurement, the user should correlate application specific events (often in the form of debug messages) with SPOT's measurements.

In our example, we did not to collect CTP's debug messages due to time constraints. But we will perform this study in future work, including enabling LPL in CTP.

Chapter 5

Energy Management

SPOT is a tool that measures energy and power, in-situ and at scale. So far, we have explored its application as a magnifier that allows us to see inside a network of motes and observe its energy dynamics. However, because SPOT provides energy measurements directly to the application running on the mote, SPOT enables the application to immediately use this information in making run-time decisions based on energy and power. This closes the feedback loop between energy sensing and actuation, and opens the door to many new possibilities.

One important application that SPOT enables is the run-time management of energy. In settings where battery replacement or recharging is difficult, it is important that sensornets have long and predictable lifetimes. We thus expect energy management to play an increasingly important role in meeting user requirements. Today, system developers seek a balance between network lifetime and performance, but recent history shows that unexpected and dynamic environmental conditions often scuttle expected energy budgets.

For example, many nodes in the Great Duck Island deployment were conjectured to have died prematurely because unexpected overhearing of traffic caused radios to become operational for longer than originally predicted [20]. This pattern was repeated in the Redwoods deployment, but for a supposedly different reason: some nodes seemingly died prematurely because they became disconnected from the wireless network and depleted their

batteries trying to reconnect [21]. Even systems augmented with energy harvesting are still susceptible to this type of problem. In the Trio Testbed, seasonal and daily variations in solar power, the angle of inclination of the solar cell, the effect of dirt and bird droppings on the cells, and the inefficiencies in power storage and transfer resulted in node duty cycles ranging from 20% to 100% [22].

The issues with these deployments arise from mistaken assumptions, unforeseen difficulties, and unpredictable environmental dynamics. Solutions to these issues take two extreme approaches. At one extreme, some have proposed run-time adaptation to meet lifetime requirements [23] or energy availability [24], [25]. While promising, these efforts have addressed rather coarse-grained, high-level adaptation – for example, by adjusting sampling rates or varying the system-wide duty cycle – but they remain silent on prioritizing energy usage in a fine-grained and flexible manner. At the other extreme, low-level energy management mechanisms that give direct control over the hardware to multiple entities (e.g. network protocols) can be tedious to implement and difficult to debug because of the lack of any isolation. Arbitration can address the isolation problem, but it does not enable run-time adaptation to varying workloads [26].

We believe that using an energy management architecture would alleviate or even prevent these types of problems. Section 5.2 examines in greater depth how applications might benefit from this architecture.

To provide a basis for energy management, we suggest an architecture that allows sensornet application writers to treat energy as a fundamental design primitive. Building systems that manage energy as a critical resource is not a new concept; research in a number of areas harnesses this idea. In fact, our architecture incorporates many of these concepts, including classifying energy as a first-class OS resource [27], prioritizing resource requests [28], accounting for fine-grained energy consumers [29], allocating resources based on dynamic constraints [30], and providing quality-of-service (QoS) guarantees by using feedback [31]. In addition, we adopt a three component decomposition that is common for architectures managing scarce shared resources, seen in Figure 5.1: (1) a *policy* interface for user input, (2) a mechanism to *monitor and control* system and component resource

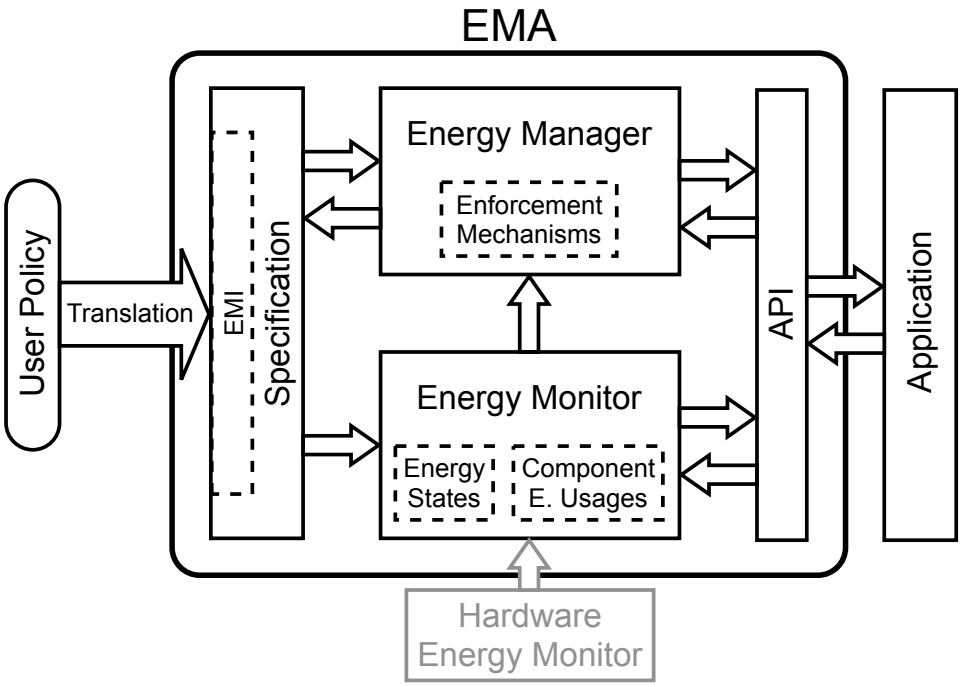


Figure 5.1. Energy Management Architecture

usage, and (3) a *management* module for enforcing policy directives. Section 5.1 describes the specific roles of these components in our architecture.

By employing and adapting concepts from traditional OS, networking, and architecture literature, we envision an energy management architecture (EMA) that allows sensornet applications to accurately view and efficiently manage the energy budget of a node. The primary contributions of this work beyond existing architectural efforts are facilities for: (1) individual accountability for management of computational units relevant to sensornets, (2) priority of policy directives to enable graceful degradation and predictable operation in a dynamic environment, and (3) expression of network-level policy that can be used by individual nodes for local optimization and shared among nodes to adjust the behavior of the network as a whole. In its entirety, the EMA promotes prioritized enforcement of policy during run-time operation as well as enables improved system behavior visualization for debugging during application development.

```
1. Network lifetime of at least one year.
2. Sample all sensors at 1Hz.
3. a. Communicate readings on multihop tree.
   b. Store readings locally.
4. Maximize sampling rate.
```

5.1 Architecture

Our vision of an energy management architecture consists of three basic components, reminiscent of classic OS resource management architectures: a specification component, an energy monitor, and an energy manager, as seen in Figure 5.1. The specification component provides a set of interfaces to the programming paradigm and communicates the energy policy to the monitor and the manager. The monitoring component observes granular component energy usages via a set of interfaces to the application. It also monitors overall system energy such as remaining battery energy and incoming harvested energy. The management component uses energy information from the monitoring component to enforce user policies conveyed by the specification component and performs admission control on activities via interfaces to the application similar to the interfaces used by the monitoring component. While some of these ideas have been well-studied in other areas of computer science, they have not been applied to sensornets effectively due to unique resource challenges. We highlight some of these challenges and offer possible solutions.

5.1.1 Specification of User Policy

Allowing a user to specify a set of directives that are dynamically checked and automatically satisfied at run-time by the system is a powerful idea that exists in many subareas of OS and networking. Applying this principle, we believe that empowering the user of a sensornet with the mechanism to directly specify energy-related requirements is very useful. Following is an example:

Expressive Interface

The specification component in our EMA provides an interface (herein, EMI) to the upper layer programming model. The programming model is responsible for translating user policy, usually written in a high-level language, through use of the EMI. The expressiveness of EMI plays a significant role in both the complexity of EMA and the amount of responsibility required of the programming model.

There are several benefits of having an expressive EMI. A rich interface will allow EMA to support a range of programming models and user policies. If a programming model is relatively simple, it can offload user policy to EMA without performing complicated translation; if a programming model is complex or if it performs some measures of energy management internally, it should still be able to use the EMI effectively while maintaining tight control over resources.

Furthermore, users may desire to specify network-level directives such as total network lifetime or uniform sampling rate. Therefore, EMI should be expressive enough for the user to specify a broad range of network behaviors. Such requirements may be satisfied in a centralized fashion where a node is selected to translate the network-level policy into node-level commands and issue them to the rest of the network, or in a distributed fashion where individual nodes translate the policy and coordinate with other nodes. We favor the distributed approach because, by preserving the high-level policy to the node level, individual nodes have more flexibility and authority to adjust to the locally optimal operating point while still retaining the ability to coordinate with other nodes to reach a global optimum. This is particularly important in a dynamic environment where energy is changing and non-uniform.

However, the expressiveness of EMI needs to be bounded and tractable. For example, we could limit the directives to contain only one optimization clause (maximize / minimize) while the rest of the clauses must be binary predicates (true / false). This, when coupled with strict prioritization, essentially reduces the algorithm complexity to linear time with one variable for optimization.

Priority

User policy is composed of a set of directives, all constrained by a single shared resource – energy. Furthermore, availability of this energy is often unpredictable due to non-uniform communication cost and environmental fluctuations (e.g. energy harvesting). We need some way of coping with the inevitable tension between directives. One solution is to associate directives with priorities, such that the system can degrade in a predictable fashion. This mechanism allows EMA to handle situations when a subset of the requirements cannot be met, which is highly likely.

For example, using the policy shown earlier, the line-item numbers could correspond to the priorities of directives while the sub-numbering (e.g. 3.a, 3.b) indicates to "do one of these," with the higher ones having priority (i.e. a before b). Using this policy, EMA would first guarantee a lifetime of one year, then a minimum sampling rate of 1 Hz, followed by either sending readings up the tree or storing them locally, depending on whether there is enough energy for sending. If there is more energy, the sampling rate is increased. If the available energy is insufficient to meet all of the directives, the system degrades in the reverse pattern: first lower sampling rate until it reaches 1 Hz; when there is not enough to satisfy 1, 2, and 3.a, do 1, 2, and 3.b.; and so on.

Exception Handler

It is often important for the user to have visibility of network health, especially when the network begins to degrade. For example, it may be important to reserve enough energy to send a packet through the network to the user informing her each time a directive in the policy can no longer be met. We incorporate this idea by introducing an exception-handling mechanism in EMA to include the user in the feedback loop about its status with respect to the policy. If a constraint can no longer be met, the node can invoke a predefined exception handler. Separate handlers can be defined for each directive, which will be invoked in ascending priority order (low to high). For example, a reasonable exception handler might send a status packet to the base station. The user can then make use of this information

by changing the policy or noting it for later use (when analyzing collected data). The exception handler could also be more elaborate, such as requesting node neighbors to take on increased responsibility or issue a new policy. A general exception-handling mechanism gives the user flexibility to control exactly what happens when the network degrades.

5.1.2 Energy Monitoring

An accurate runtime view of the energy states is necessary for the EMA management component and can be very useful to applications, routing protocols, MAC protocols, or any other components performing energy-related optimizations. Similar to traditional OSes, presenting energy information at the OS-level usually provides reusability, interoperability, and efficiency. We divide energy information into two parts: component energy usage profiles and system energy levels.

Component Energy Usage Profiles

Conceptually different activities, such as temperature sensing, external storage usage, and radio usage, should be *individually accountable*. This provides the system with a fine-grained view of energy usage and allows for arbitration and admission control of activities based on user policy. The separation of activities ideally correlates with the guidelines in the user policy and therefore provides the energy management component with a basic unit for control.

But unlike in a traditional OS, there is often no clear unit of accounting for energy, such as processes or threads, in sensornets. This is further complicated by cases when one component (e.g. the radio) is consuming energy on behalf of another component (e.g. a light sensor), in which case the energy used by the radio should be logically accounted for as light sensing. There have been several research efforts that attempt to provide some notion of grouping. They may be adapted to providing energy accounting in different ways, varying on flexibility, code reusability, and complexity. The suitable choice would depend on the application and programming paradigm. For example, we may be able to group activities

by task and manage the tasks through an energy-aware task scheduler. This grouping is naturally supported by task-based systems such as TinyOS [32] or Tenet [33]. However, in the case of TinyOS, even though it uses task IDs, it does not currently support thread context and therefore it cannot easily differentiate multiple invocations of the same task by different activities. The Tenet task library presents functional blocks at a relatively high level and could be grouped by activity fairly naturally. However, some tasks that are used by multiple activities will require extra attention for differentiation. TinyOS 2 (T2) [16] defines a hardware abstraction layer (HAL) that allows grouping by hardware. However, the granularity of HAL is low-level and may not easily correspond to activities in the energy policy. Similarly, the resource component [26] in T2 provides a fixed grouping, albeit at a higher level than HAL. A more flexible approach is to allow arbitrary grouping by the application via a set of APIs. This approach allows the application to decide what should be logically grouped into an activity. However, this will require rewriting existing applications to use the API.

System Energy Levels

System energy levels refer to the actual energy in the system, such as the remaining battery energy as measured by a hardware monitoring facility (such as *SPOT*) or through software estimations. For example, the current energy level can be calculated in software by assuming an initial energy and subtracting component energy usages. However, software methods usually assume a set of static component energy profiles that may not be accurate, do not account for all possible energy consumers, and accumulate error over time. Hardware monitoring facilities, if available, provide much more accurate energy measurements.

5.1.3 Energy Management

Similar to traditional QoS in networking, the energy manager uses some form of admission control to regulate component accesses to energy resources, providing QoS based on user policy. However, unlike traditional energy management systems, the scope of our

manager extends beyond making local decisions, as there may exist network-wide polices that require some sort of coordination between nodes. In sensornets, energy management deals with both local and global resource management.

Locally, we can use system energy levels and component usage profiles from the energy monitor to make decisions based on user policy. For example, an average consumption rate can be calculated to estimate the remaining lifetime. This estimate, coupled with real-time component energy usage, allows the energy manager to perform per-activity admission control. Globally, we can manage resource usage by sharing system energy levels amongst nodes. This is useful if, for example, the policy requires uniform sampling, as a network can only sample as quickly as the node with the lowest sampling frequency. EMA may also borrow management strategies from networking, such as rate control protocols used for the Internet. This allows fairness to be considered among competing energy-consuming activities.

Distributed coordination raises questions about the architecture. It is unclear where the networking component, used to send messages among nodes, will reside. It can be part of the manager, but must closely interact with the network layer of the system to send and receive coordination packets. It may also be collected into a library that is imported into the image at compile-time if a statement in the user policy requires network-wide coordination. Perhaps it is best suited as an application-specific implementation that is facilitated by providing the necessary interface to the energy states. Since distributed coordination is not always necessary, integrating it into the architecture monopolizes valuable resources. We will concentrate on the advantages of either making this component a compile-time library or providing hooks for implementation in the application layer.

We also consider ways to evaluate user policy in a design-time "sanity check." This may be done by collecting information from the network and comparing the active state of the nodes in the network with a known knowledge base or model to see if the provided user policy is feasible. This could be helpful to the end user in designing policy and ensuring, within a reasonable performance bound, that user requirements can be enforced.

5.2 Examples

This section presents two examples of how specific sensornet deployments could benefit from using EMA. We show that each application is representative of a broad class of applications and that EMA is flexible enough to meet varied requirements while providing specific advantages to application developers. Additionally, by providing EMA with multiple degrees of freedom, such as variable sampling rates, reporting rates, and the ability to use low-consumption storage rather than the radio, the application developer can increase the probability of successful deployments.

5.2.1 Redwoods

In [21], the authors describe a deployment that sought to examine the microclimates surrounding Redwoods trees through measurements of temperature, relative humidity, and incident and light intensity, taken every five minutes. As is common for most monitoring applications, this sampling period was selected through an iterative evaluation between environmental and computer scientists based on the characteristics of the phenomena under observation, the battery size, and the *estimated* energy consumption of each sense, log, and send operation. Readings were logged and reported for a total of 44 days, though some nodes were not functional for the entire duration due to depleted batteries. The cause of this unexpected energy depletion was a software bug that kept nodes in an energy-consuming "listening" state if a communications opportunity (every 5 minutes) was missed. The energy monitoring component could have been used during the application development process to provide per component energy usage statistics. Upon visualizing this unforeseen energy consumption, the designers might have tracked down this bug prior to deployment, averting energy waste and the resulting exhausted nodes and missed data gathering opportunities.

If, for example, energy usage were not tested in advance, EMA could still circumvent the faulty behavior. Below is an example set of user policies for this application.

The notion of priority in EMA user policy permits *graceful degradation* of these requirements. During the course of the experiment, if the energy monitor calculates that there

```
1. Network lifetime of at least 1 month.
2. Sample all sensors every 5 minutes and
   a. Send readings to base station.
   b. Log readings to local storage.
```

will be insufficient energy to survive for the rest of the desired lifetime (in this case, one month) while sampling and sending (item 2.a.), the energy manager will abandon item 2.a. and instead enforce items 1 and 2.b. (i.e. sample and store the readings). In the event that there is not enough energy to meet either of these policies, the energy manager will use the remaining energy to perform the exception handler specified by the user. In this way, priority protects a system that is unable to meet all the goals of the user by at least satisfying the most important goals. We argue that using EMA for this application would either allow this fault to be caught in the development process or prevent it from occurring during runtime operation, increasing overall data yield of this application substantially.

These benefits are broadly applicable to a large class of environmental monitoring applications where specifications are strict and static (e.g. sample every 5 minutes) such as [35]–[38]. Essentially, EMA facilitates expressing priority in user policy and handling the tradeoffs that inevitably appear in the face of dynamic environmental conditions.

5.2.2 Arctic Observations

Dr. David Carlson, in his SenSys 2006 keynote speech about the International Polar Year (IPY), urged the sensornet community to develop applications for Arctic observations. He argued that sensornets are a solution to the chief requirement of these applications: broad observation over timescales of an entire season or longer with minimal device maintenance. This requirement arises from the inaccessibility of the environment, often either too remote, expensive, or harsh to visit frequently, and the observation of phenomena with relatively long durations. In these applications, network lifetime is the overriding constraint while, perhaps, sampling and reporting rates may be relaxed.

Consider an application to monitor the melting of glaciers similar to [39]. With a

minimum lifetime encoded as the first requirement, reporting and sampling rates can be modulated in order to achieve the foremost requirement of lifetime, as is shown below.

```
1. Network lifetime must be at least 1 year.
2. Sample all sensors at least every 4 hours and
   a. Send readings to basestation once a day.
   b. Log readings to local storage.
```

Network lifetime as the overriding constraint characterizes a class of applications where challenging environments and isolated networks (i.e. no uplink) prevent predictable communication and sensing patterns. In addition to enabling this class of applications, EMA facilitates the class of lifetime-centric deployments in which the phenomena dictates the duration of the network, such as the life of a rat, the pollination cycle of a flower, or the extent of a storm season. Furthermore, by empowering EMA with another degree of freedom, in this case a variable sampling rate, the energy manager component is able to make informed decisions regarding the tradeoffs presented by the user policy and the environmental conditions.

5.3 Proof-of-Concept Exercise

In this exercise we show some potential benefits of an OS-level energy management service. This is purely a proof-of-concept exercise written in TinyOS. We want to demonstrate that by accounting for individual energy-consuming activities and performing run-time optimizations based on prioritized user policy, even a trivial implementation can provide some amount of assurance and predictability in addition to better resource utilization in a dynamic environment.

We use a simple *request/grant* API and provide energy management as a generic TinyOS component (herein EM). We use a uniformly-weighted moving average to estimate current power for a particular activity, hold requests in a 1-deep queue, and grant only when there is enough energy to satisfy both the requested activity as well as all other activities with higher priorities. The application has the flexibility to specify which energy account to debit

```
1. Network Lifetime >= 100 seconds.
2. Allow all radio traffic.
3. Sense as fast as possible.
```

and by how much using the *request(account, energy_type_string)* command and predefined types. Additionally, the application has the flexibility to group tasks as it desires for the *grant()* event.

We make the following assumptions: (1) Each mote has an initial energy of 1100 units; (2) Sensing costs 10 energy units per sample; and (3) Communication costs 1 energy unit per packet.

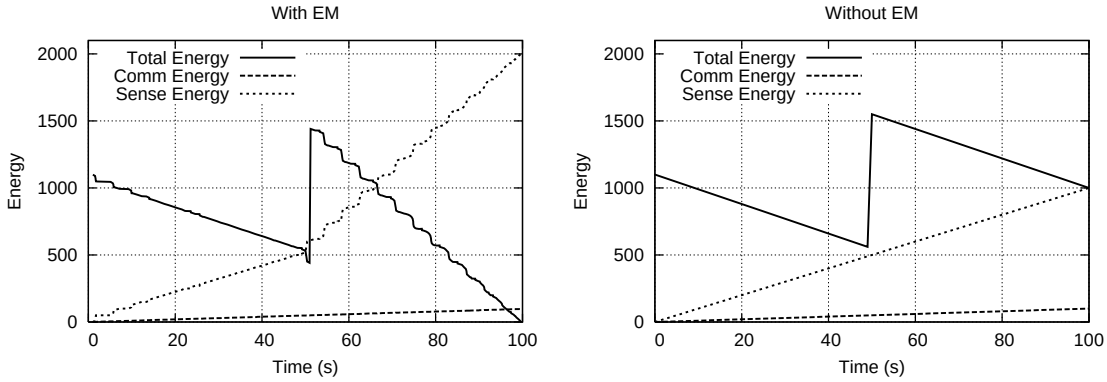
The user policy below is directly encoded in a header file:

Three motes are subjected to different (simulated) environmental conditions and their energy states are periodically reported by radio. The communication rate is initially set to 1 packet per second.

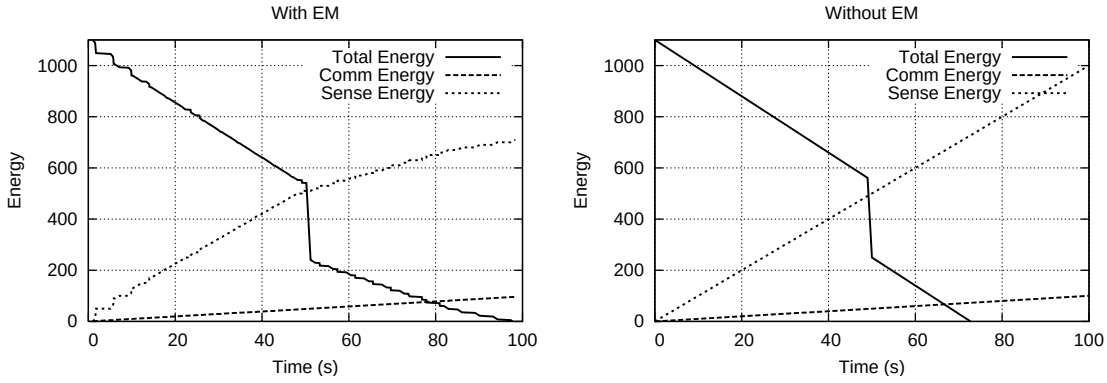
As seen in Figure 5.2(a), when system energy is increased at t=50 (to simulate energy harvesting), EM automatically increases the rate of grants for sensing, achieving better energy utilization. Conversely, in Figure 5.2(b), EM reduces the sensing rate when detecting a drop in overall energy at t=50, whereas the mote without EM dies at around t=70. Figure 5.2(c) shows a plausible scenario where the communication rate increases in the middle of the experiment, possibly due to environmentally-induced retransmissions or increased forwarding responsibility. With EM, the mote is able to reduce sensing and satisfy the lifetime guideline, whereas without EM, the mote dies at t=80. For all EM-enabled motes, the lifetime is exactly 100 seconds with full energy utilization, while the sensing rate is adapted to comply with user-defined policy.

5.4 Implications

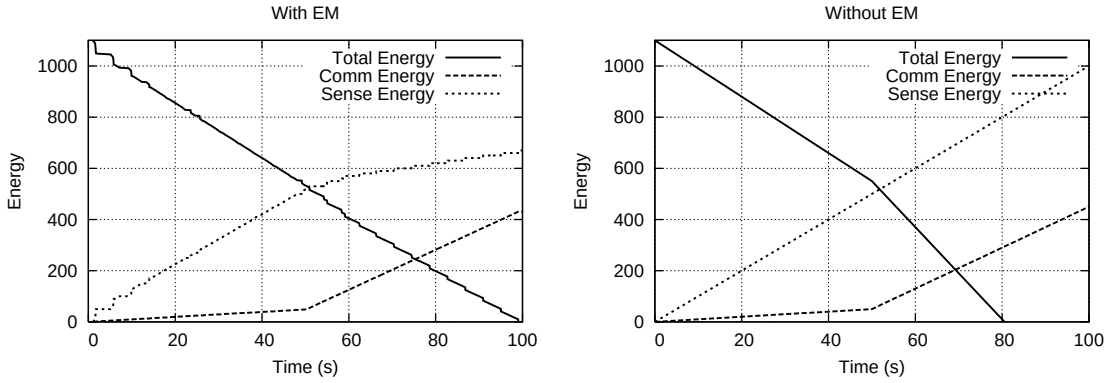
The push for an overall sensornet architecture is not a new one. One particular architecture has been outlined in [40], and a series of architectural components have been



(a) 1000 units of energy are injected at time t=50, i.e. energy is harvested.



(b) 300 units of energy are subtracted at time t=50, i.e. energy storage malfunction.



(c) Communication rate increases to 8 Hz at time t=50, i.e. increased transmissions due to interference or more forwarding responsibility.

Figure 5.2. Behavior of motes under simulated environments. Left columns shows data collected from three motes written using EM while the right column shows simulated data without EM

proposed in [33], [41], [42]. Additionally, a host of industrial standards/specifications are being developed for either sensornets in particular or low-power wireless devices in general.

One of the basic design principles for EMA is to remain as agnostic as possible to the overall sensornet framework within which it will be situated. As such, rather than create an energy management service that is deeply intertwined throughout the system stack, a cleaner approach is to create a standalone component that only integrates through clearly-defined interfaces and is positioned parallel to the rest of the stack. The important aspect of these interfaces is that they are simple and powerful, yet general enough to be used by a variety of services.

Based on preliminary analysis, we believe our architecture will effectively complement a host of existing architectures and programming paradigms, such as [33], [41]–[44]. Although one of our design assumptions is the use of TinyOS, the architecture is easily portable to additional operating systems. We provide a concrete example as a single point in the design space.

Our example system is composed of a modular structure, namely SP [41], [45] and NLA [42], as well as DSN [43], a declarative programming paradigm. EMA sits parallel to the rest of the system. SP serves as the narrow waist, decoupling the network layer from the link layer. NLA is a modular network layer framework that decomposes network protocols into a routing topology, a routing engine, a forwarding engine, and an output queue. NLA sits directly on top of SP and exports a narrow send/receive interface to upper layers. DSN models the sensor node as a query processor, allowing users to declaratively specify programs using rules and predicates. The declarative high-level nature of the user interface of DSN provides a suitable mechanism for the interpretation of user policy, which will consequently by translated into the EMI. DSN also provides the ability to specify these requirements both during compile-time as well as dynamically during runtime, fully utilizing the dynamic capabilities of EMA. Based on these requirements, the energy manager will determine the admission control policy for the system. EMA may use NLA to communicate nodal energy levels between neighbors and coordinate collective actions for satisfying network level directives.

Chapter 6

Conclusion

We presented the hardware design, software driver, calibration process, and single & network-level evaluation of SPOT – an accurate and sensitive meter for monitoring wireless sensor network power profiles and energy usage patterns at scale. We showed that SPOT is able to meet or exceed challenging requirements unique to wireless sensor networks and other low power systems, as summarized in Table 6.1.

Metric	Requirement	SPOT
Dynamic Range	$> 10000 : 1$	$45000 : 1$
Resolution	$< 2\mu A$	$< 1\mu A$
Sampling Rate	> 20kHz	Internally at 1MHz
		Output at I2C speed
Perturbation	Minimal	1Ω additional load to DUT
		Energy measurement via I2C
		At least one read per hour
Integration	Easy	1.35"x1" all-in-one
Cost	$< \$25$	Off-the-shelf ICs

Table 6.1. SPOT satisfies the power and energy metering needs of sensornet nodes *in situ* and *at scale*.

We also showed that SPOT can be easily integrated into existing applications via a simple API and demonstrated that SPOT is able to visualize the energy and power consumption of real applications. SPOT employs an simple architecture that uses a pure analog sampling

front end to provide a large dynamic range, and a dual counter energy accumulation stage to provide high temporal resolution. These features are useful for profiling a range of systems – like pagers, PDAs, and cellphones – which exhibit highly bimodal or widely varying power profiles. SPOT will enable heretofore impossible empirical evaluations of low power designs at scale, and it will enable a new class of sensornet research in which applications can integrate the dynamic power profile of a system into the application logic.

We also presented our vision of an energy management architecture that provides energy as a primitive by implementing services for both energy monitoring and management at the OS-level. An integral part of EMA is the energy monitoring component. While software emulation provides an approximation, SPOT is able to provide much more detailed and accurate energy and power profiles with little software overhead. An energy management architecture will aid in future designs of energy-aware application and enable a class of energy-centric applications where lifetime and performance requirements can be specified and controlled under a uniform framework. The monitoring component exposes two previously unavailable streams of information: system energy level and component energy usage. The application can utilize this information in many ways, including energy adaptation, fine-grained component level performance adjustment, intelligent duty cycling, evaluation of low-power designs, and graceful degradation. The specification component empowers the user and programmer with the ability to directly express energy-related policies such as lifetime and sensing rates. The management component enables the system to enforce this policy. This consolidates many variations of energy management under a unified framework and fuses previous attempts in expressive energy monitoring granularity. An energy management architecture not only benefits applications during run-time, but also helps programmers during the debugging phase. EMA presents increased visibility into the network allowing effective debugging of potential problems.

References

[1] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," *IEEE SPOTS*, 2005.

[2] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a wireless sensor network platform for detecting rare, random, and ephemeral events," in *IPSN/SPOTS*, Apr. 2005.

[3] J. Hsu, J. Friedman, V. Raghunathan, A. Kansal, and M. Srivastava, "Heliomote: Enabling self-sustained wireless sensor networks through solar energy harvesting," in *ISLPED*, Aug. 2005.

[4] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOM*, Jun. 2002.

[5] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys*, Nov. 2004.

[6] O. Saukh, P. Marron, A. Lachenmann, M. Gauger, D. Minder, and K. Rothermel, "Generic routing metric and policies for wsns," in *EWSN*, Feb. 2006.

[7] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari, "Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks," in *Conference On Embedded Networked Sensor Systems*, Nov. 2004.

[8] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin, "Topology control protocols to conserve energy in wireless ad hoc networks," in *Technical Report 6*, Apr. 2003.

[9] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in zebranet," *ACM Sensys*, 2004.

[10] T. He, S. Krishnamurthy, J. A. Stankovic, T. F. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh, "An energy-efficient surveillance system using wireless sensor networks," in *MobiSys*, Jun. 2004.

[11] S. Gurun and C. Krintz, "Full system energy estimation for sensor network gateways," in *Technical Report*, Oct. 2006.

[12] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, , and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *SenSys*, Oct. 2004.

[13] Maxim-IC, "Ds2438 smart battery monitor," http://datasheets.maxim-ic.com/en/ds/DS2438.pdf, Jul. 2005.

[14] Analog Devices, "Single phase multifunction energy metering ic with di/dt input," http://www.analog.com/UploadedFiles/Data_Sheets/ADE7753.pdf, Aug. 2003.

[15] Microchip, "Energy metering ic," http://ww1.microchip.com/downloads/en/DeviceDoc/21948c.pdf, Aug. 2005.

[16] P. Levis, D. Gay, V. Handziski, J.-H. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, and A. Wolisz, "T2: A second generation os for embedded sensor networks," *Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universitat Berlin*, 2005.

[17] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, and D. Culler, "Flexible hardware abstraction for wireless sensor networks," *EWSN*, 2005.

[18] "Micaz," http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

[19] Rodrigo Fonseca and Omprakash Gnawali and Kyle Jamieson and Sukun Kim and Philip Levis and Alec Woo, "The collection tree protocol (ctp)," http://www.tinyos.net/tinyos-2.x/doc/txt/tep123.txt, Aug. 2006.

[20] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," *ACM Sensys*, 2004.

[21] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," *ACM Sensys*, 2005.

[22] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, and D. Culler, "Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments," *IEEE SPOTS*, 2006.

[23] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: An acqusitional query processing system for sensor networks," *ACM TODS*, 2005.

[24] A. Kansal, D. Potter, and M. B. Srivastava, "Performance aware tasking for environmentally powered sensor networks," *ACM Sigmetrics*, 2004.

[25] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," *IEEE SPOTS*, 2005.

[26] K. Klues, V. Handziski, D. Culler, D. Gay, P. Levis, C. Lu, and A. Wolisz, "Dynamic resource management in a static network operating system," *In submission.*

[27] H. Zeng, C. S. Ellis, and A. R. Lebeck, "Experiences in managing energy with ecosystem," *IEEE PerCom*, 2005.

[28] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," *USENIX OSDI*, 1999.

[29] R. Neugebauer and D. McAuley, "Energy is just another resource: Energy accounting and energy pricing in the nemesis os," *USENIX HotOS*, 2001.

[30] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," *ACM SOSP*, 1999.

[31] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE JSAC*, 1996.

[32] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for wireless sensor networks," *Ambient Intelligence*, 2005.

[33] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler, "The tenet architecture for tiered sensor networks," *ACM Sensys*, 2006.

[34] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," *USENIX OSDI*, 2006.

[35] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," *ACM Sensys*, 2004.

[36] N. Ramanathan, L. Balzano, M. Burt, D. Estrin, T. Harmon, C. Harvey, J. Jay, E. Kohler, S. Rothenberg, and M. Srivastava, "Rapid deployment with confidence: Calibration and fault detection in environmental sensor networks," *CENS Tech Report #62*, 2006.

[37] R. Musaloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray, "Life under your feet: Wireless sensors in soil ecology," *EmNets*, 2006.

[38] K. Martinez, P. Padhy, A. Elsaify, G. Zou, A. Riddoch, J. K. Hart, and H. L. R. Ong, "Deploying a sensor network in an extreme environment," *IEEE SUTC*, 2006.

[39] D. Culler, P. Dutta, C. T. Eee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, , and J. Zhao, "Towards a sensor network architecture: Lowering the waistline," *USENIX HotOS*, 2005.

[40] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A unifying link abstraction for wireless sensor networks," *ACM Sensys*, 2005.

[41] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica, "A modular network layer for sensornets," *USENIX OSDI*, 2006.

[42] D. Chu, A. Tavakoli, L. Popa, and J. Hellerstein, "Entirely declarative sensor network systems," *ACM VLDB*, 2006.

[43] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood, "Envirotrack: Towards an environmental computing paradigm for distributed sensor networks," *IEEE ICDCS*, 2004.

[44] A. Tavakoli, J. Taneja, P. Dutta, D. Culler, S. Shenker, and I. Stoica, "Evaluation and enhancement of a unifying link abstraction for sensornets," *In submission.*