TECHNICAL REPORT

Experiments with a Solar-powered Sun SPOT

Vipul Gupta

> *Sun Microsystems Laboratories*

Experiments with a Solar-powered Sun SPOT

Vipul Gupta

SMLI TR-2009-178 January 2009

Abstract:

Sun SPOTs are small, battery-powered, wireless embedded devices that can autonomically sense and respond to their environment. These devices have the potential to revolutionize a broad spectrum of applications - environmental monitoring, asset tracking, proactive health care, intelligent agriculture, military surveillance, *etc*. Many of these require the device to run for long periods (months) using a combination of duty cycling and renewable energy sources (*e.g.*, solar panels). This note describes lessons learned while collecting data from a solar-powered SPOT for a period of nearly four weeks.

Sun Labs
16 Network Circle
Menlo Park, CA 94025

email address:
vipul.gupta@sun.com

For information regarding the Sun Microsystems Laboratories Technical Report Series, contact Mary Holzer or Nancy Snyder, Editors-in-Chief <Sun-Labs-techrep-request@sun.com>.All technical reports are available online on our website, http://research.sun.com/techrep/.

Experiments with a Solar-powered Sun SPOT

Vipul Gupta
Sun Microsystems Laboratories
16 Network Circle, UMPK16-160
Menlo Park, CA 94025
vipul.gupta@sun.com

Abstract

Sun SPOTs are small, battery-powered, wireless embedded devices that can autonomically sense and respond to their environment. These devices have the potential to revolutionize a broad spectrum of applications – environmental monitoring, asset tracking, proactive healthcare, intelligent agriculture, military surveillance, *etc.* Many of these require the device to run for long periods (months) using a combination of duty cycling and renewable energy sources (*e.g.*, solar panels). This note describes lessons learnt while collecting data from a solar-powered SPOT for a period of nearly four weeks.

1 Introduction

The Sun™ Small Programmable Object Technology (SPOT) [1] is a research platform designed at Sun Labs to explore the next frontier of network computing. Each Sun SPOT device is smaller than a deck of cards, has a 32-bit ARM processor, a rechargeable battery, an IEEE 802.15.4 radio and is programmed in Java™. These devices can be connected to a wide range of application-specific sensors and actuators such as accelerometers, light, temperature, humidity sensors, GPS units, servos, *etc*.

Potential uses for Sun SPOTs include the traditional wireless sensor applications – asset tracking, environmental monitoring, industrial sensing, security surveillance, proactive healthcare and intelligent agriculture – but their extra capabilities and flexibility make them suitable for many additional applications, *e.g.*, robotics, games. For this article, however, we focus on the typical wireless data collection scenario where the device sleeps most of the time but wakes up periodically to sample and transmits sensor values.[1]

Sun SPOT devices have the ability to automatically enter a power saving state when no active work is scheduled. We elaborate on this feature in Section 2. Our experimental set up is described in Section 3. Section 4 highlights our findings and lessons learnt. These lessons should be useful to anyone designing applications or frameworks dealing with long-term data collection on Sun SPOTs. Section 5 briefly summarizes our conclusion.

2 Automatic Power Saving

Sun SPOTs run a specially designed small-footprint Java virtual machine, called Squawk, which can host multiple applications simultaneously and does not require an underlying operating system. Even the device drivers (*e.g.*, to control the radio) are written in Java so the Squawk virtual machine has complete knowledge of all active threads. Whenever it detects that all threads are idle (*i.e.*, executing Thread.sleep(), blocked on

[1] This notion of controlling the ratio of on-period to total cycle time is called *duty cycling*.

a synchronization primitive or waiting for an interrupt from the hardware), it puts the device into one of two power saving modes:

Shallow sleep – In *shallow sleep*, the CPU clock is stopped but much of the hardware is still powered allowing the device to react quickly to external events like arrival of radio data. The SPOT can resume from shallow sleep without additional latency, *i.e.*, as soon as any thread becomes ready to run, the Sun SPOT wakes up and carries on.

Deep sleep – In *deep sleep*, the CPU, RAM, Flash, external board, and the master system clock are all powered off (RAM contents are preserved by low power standby supply). Because deep sleep involves switching off the power to peripherals that may be active, it is necessary to interact with the device drivers to determine if deep sleep is appropriate. If a deep sleep cannot be performed safely, the scheduler will perform a shallow sleep instead. It takes some time to wake from deep sleep, so the scheduler may choose to shallow sleep if the sleep duration will be too short to make deep sleep worthwhile. The minimum idle time for which deep sleeping is allowed is accessible as sleepManager.getMinimumDeepSleepTime(). The minimum deep sleep time includes a time allowance for reinitializing the hardware on wake up so that user code resumes at the correct time. Any part of the allowance that is not needed is made up using a shallow sleep. A SPOT in deep sleep has a current draw of around 33μA. Connecting an external power source to the Sun SPOT via the USB socket or its V_{ext} pin inhibits deep sleep and shallow sleep is used instead.

Additional details on the sleep mechanism can be found in the *Sun SPOT Developer's Guide* [2]. Decisions about when to use shallow sleep or deep sleep are taken by the Java thread scheduler inside the VM. This is transparent to applications and no special work on the part of the programmer is required to take advantage of it. This automatic power management feature can be viewed as an extension of the automatic memory management that has long been a hallmark of Java and similar programming languages.

3 Experimental Set Up

The hardware for our experimental set up includes a free-range SPOT connected to a solar panel via the USB port and a laptop with an attached basestation. A mySQL database installed on the laptop acts as the data sink. This is shown in Figure 1. Figure 2 shows a close up of the solar-powered SPOT. The solar panel used in our setup [3] is roughly 7 in $\times$ 4.5 in and special circuitry sits between the panel and the USB port on the SPOT to regulate the voltage. Charge generated by the solar panel is stored in capacitors until enough has been accumulated to supply a steady 5V to the USB port. At that point, the charge is transferred to the SPOT. When there isn't enough charge to supply 5V, the circuit makes the solar panel appear disconnected to the SPOT.

The software used is a derivative of the *SensorFramework* code [4] originally developed by Ron Goldman. The framework abstracts away low-level details dealing with scheduling the radio and various sensors and presents the programmer with a high-level API. Creating a data collection application is as simple as defining a *sensor* (what values are reported, how the values are collected, their units *etc.*), specifying sampling and transmission schedules and running a built-in host application that listens for sample values and prints them out. The code running on the solar SPOT defines a *compound sensor* such that a single *sample* includes not only light and temperature readings (from sensors built into the EDemoboard included with the SPOT kit) but also battery health, voltage detected on the USB port (where the Solar panel is connected), total uptime and sundry sleep related statistics. As we show later, these additional parameters are useful in debugging and failure analysis.

We've enhanced the framework with the ability to record sensor readings into a mySQL database and a few other features in response to lessons learnt. We also created a simple servlet that uses an *annotated*

Figure 1: The experimental setup includes a laptop with an attached basestation communicating with a solar-powered Sun SPOT, about 15 feet away, using the 802.15.4 radio. The SPOT is placed outside the window of the office that has the laptop. Sensor data is stored in a mySQL database installed on the laptop.

time line [5] from the set of Google visualization tools to plot different fields in a database table against the corresponding timestamp. The servlet is hosted inside Jetty [6], a small footprint web server that can be configured as a servlet container, and makes it possible to access the plots remotely using any web browser that supports JavaScript^{TM} and Adobe Flash.

It is worth mentioning that the SensorFramework was put together quickly as a prototype to gain real-world experience with long-term data collection. A full fledged data collection framework for Sun SPOTs, called Yggdrasil [7], is currently under development but wasn't ready for use when these experiments were initiated. A primary motivation for this write-up is to ensure that Yggdrasil and similar efforts can benefit from the lessons learnt.

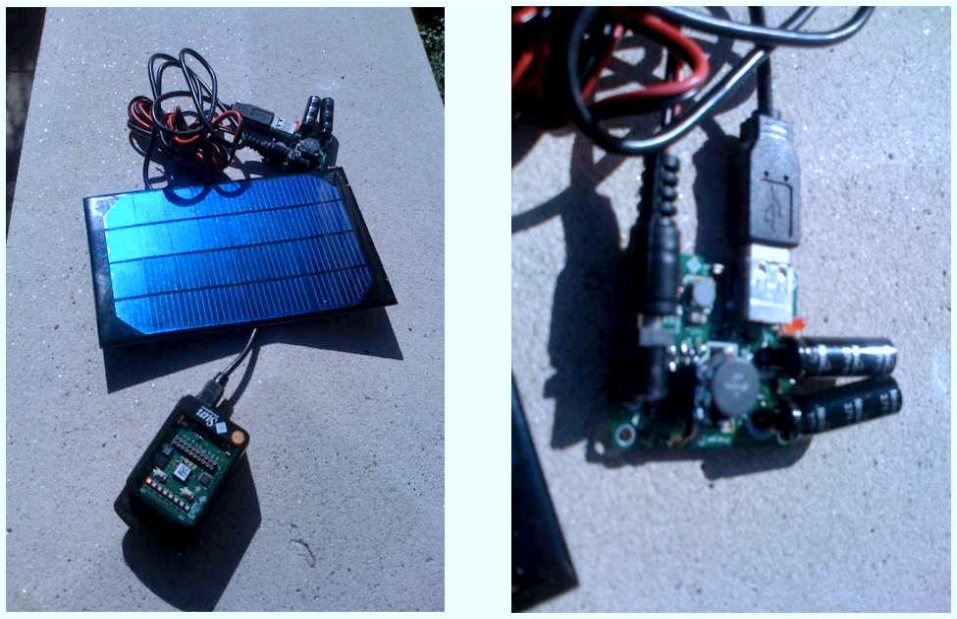


Figure 2: A special charging circuitry sits between the solar panel and the Sun SPOT device.

4 Experimental Results

We ran our experiments for 26 days from September 9 through October 4, 2008. During this time, the solar-powered SPOT collected data samples every 10 minutes starting on the hour and sent them to the sink every 30 minutes. In addition, the SPOT stayed awake for 2 minutes on the hour every hour between 9am and 5pm to receive over-the-air management commands. This sampling schedule was chosen as being representative for a large class of applications based on our interaction with several domain experts.

4.1 Dealing with interruptions in data collection

Figure 3 shows the plots we obtained for light, temperature, USB port voltage, remaining battery level (expressed as a percentage of maximum capacity), battery voltage, number of times deep sleep happened, time spent in deep sleep, and time spent in shallow sleep. These plots have several conspicuous gaps representing periods during which data collection was interrupted. Some of the causes for this disruption were the brittle nature of the original protocol through which the SPOT and sink discover each other, inactivity timeouts in the database connection and the loss of clock on the SPOT due to battery exhaustion. Identifying and fixing these issues turned out to be a significant contribution of our experiments.

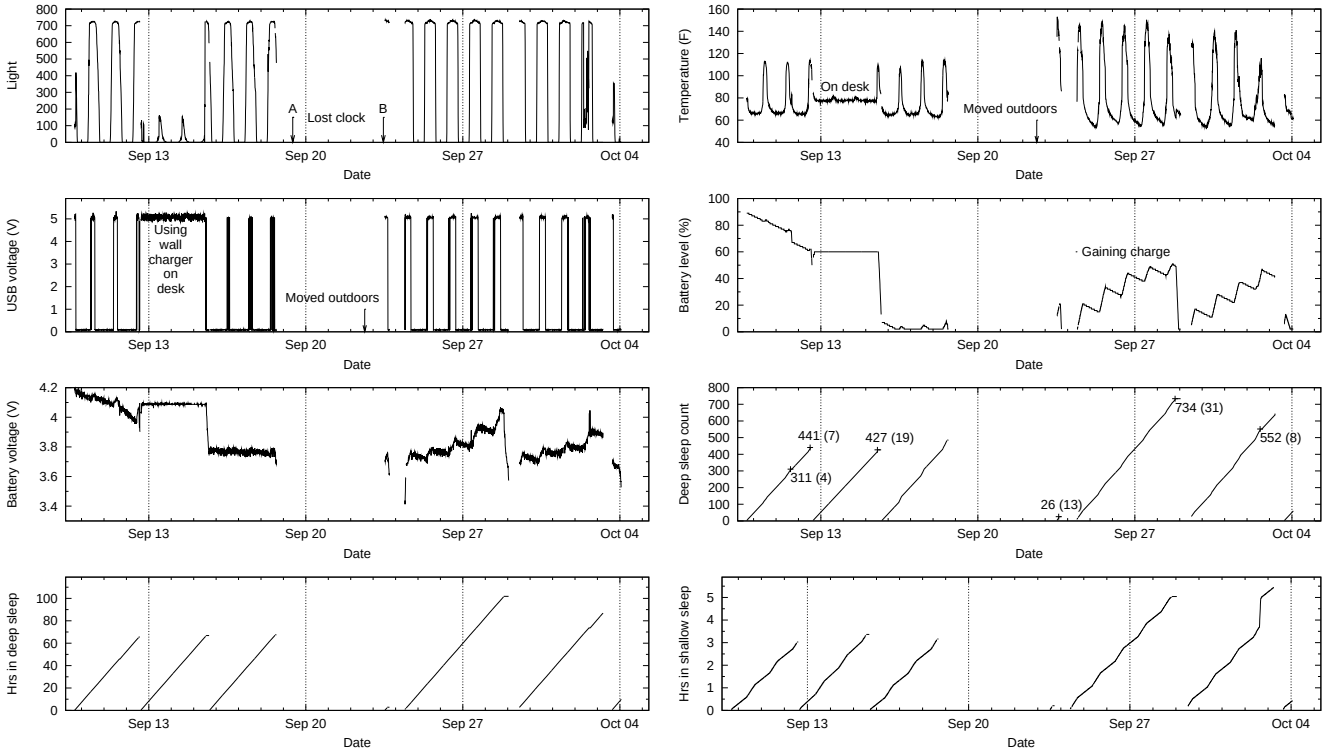


Figure 3: Plots of light, temperature, USB port voltage, battery level, battery voltage, deep sleep count, time spent in deep sleep and time spent in shallow sleep against time. Annotated points in the plot for deep sleep count indicate the value at which deep sleep count stalled and the number of samples, in parenthesis, reporting the stalled count.

With the original SensorFramework code, a SPOT did not collect and send samples after rebooting unless the host application was also restarted. Furthermore, the introduction of a new sink used to cause silent redirection of the samples away from the original sink. We modified the discovery protocol so data collection can resume even if either the part running on the host or that running on the SPOT is restarted. If

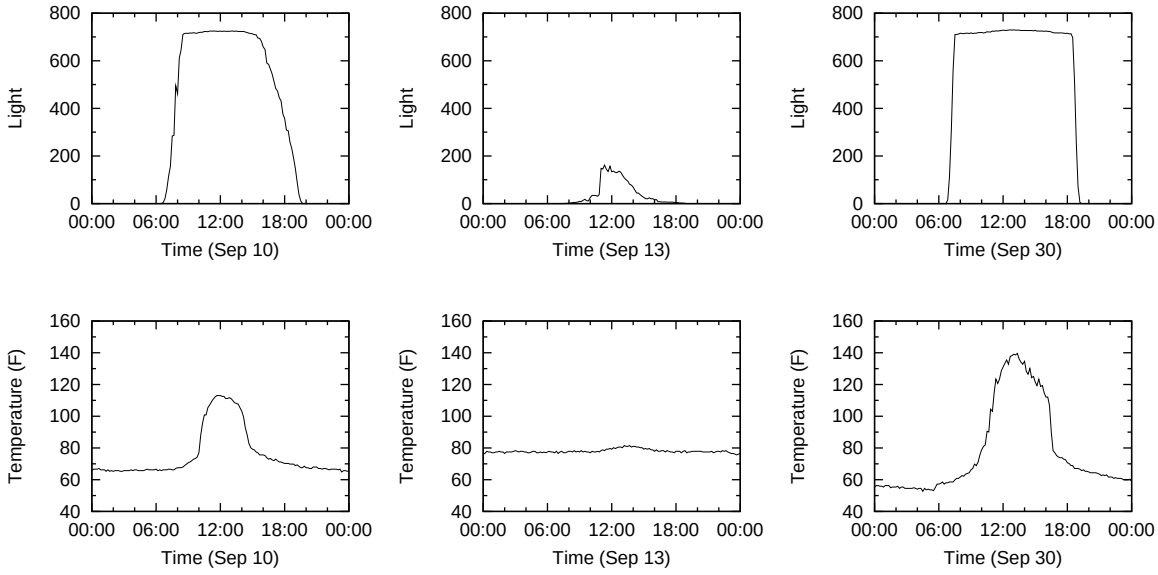
a sink is already in use, additional sinks discovered subsequently are ignored and used only for fail-over if the original sink becomes inoperational or unresponsive.

When a SPOT is completely discharged, its clock gets reset to the Unix epoch (00:00:00 UTC, January 1, 1970) and subsequent samples are sent with an incorrect timestamp. For example, between 2008-09-19 11:00 and 2008-09-23 12:00 (these points are marked 'A' and 'B' in the light plot in Figure 3), the SPOT reported a sample timestamp near Jan 1, 1970 and those samples do not show up in Figure 3. We added a time synchronization mechanism to the SensorFramework so a SPOT that has lost its clocks can reset it correctly upon hearing a beacon from the basestation.

We addressed the database timeout issue by catching link failure exceptions and retrying the database operation after reestablishing the connection.

4.2 Evaluating built-in sensors

Before moving the SPOT outdoors, we experimented indoors by placing the SPOT and solar panel just inside an office window. The temperature plot in Figure 3 clearly illustrates periods when the SPOT was indoors (before Sep 19) and outdoors (after Sep 23). The same plot also helps us differentiate between two indoor positions – on the desk (2008-09-12 15:00 through 2008-09-15 12:10) where the temperature variations are more modest and on the window ledge. Similar trends are also visible in the light plots. Both are highlighted in closeups in Figure 4.



(a) Indoors, near window (b) Indoors, on desk (c) Outdoors

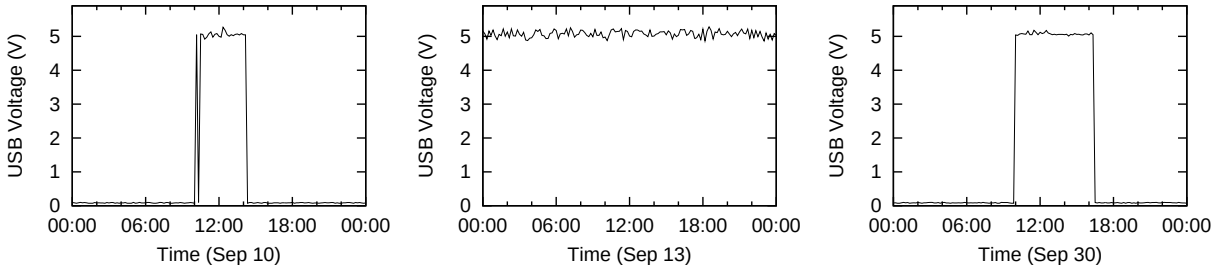
Figure 4: Closeup of the light and temperature plots shown in Fig 3 for three days in September. As expected, the sensor readings show the least amount of change when the SPOT is indoors away from the window and vary the most when the SPOT is outdoors.

The light sensor on the EDemoboard saturates quickly when used outdoors and is ineffective at detecting subtler changes in light intensity during the day. The temperature sensor monitors the on-board temperature and tends to overestimate the temperature of the ambient air. These observations suggest that efforts like Warren Wilson's ecological monitoring of the Cocobolo Nature Reserve in Panama [8] might be better served by more accurate external sensors.

4.3 Charging a SPOT

The battery on a SPOT is recharged by applying 5V to the USB port. The solar panel was the primary mechanism for charging the SPOT in our experiment but, for some time, we also experimented with a standard USB "wall" charger (often used for mobile phones) that plugs into an electric outlet. In Figure 3, non-zero values in the USB voltage plot correspond to periods when the SPOT was charging (irrespective of the charger used). The long flat region from 2008-09-12 15:00 through 2008-09-15 12:10 corresponds to the time when the SPOT was on the desk connected to a wall charger. Figure 5 shows a closeup of the USB voltage plot for three representative days.

Note how the special charging circuitry, shown in Figure 2, ensures that the voltage on the USB port is either 5V or 0V. As expected, the pulses are wider when the SPOT was outdoors – the sunlight was more intense and lasted longer (as seen in the light plots in Figure 4).



(a) Indoors, near window using solar panel

(b) Indoors, on desk using wall charger

(c) Outdoors using solar panel

Figure 5: Closeup of the USB voltage plots shown in Fig 3 for three days in September. The solar panel keeps the SPOT charging longer when outside.

It turns out that the chosen schedule allows a SPOT, left outdoors, to gain charge on a typical sunny day (see the battery level plot in Figure 3 in the regions between 2008-09-24 10:00 through 2008-09-28 21:00 and 2008-09-29 12:30 through 2008-10-03 06:00). The overall shapes of the plots for battery level and battery voltage are very similar since the two have a mostly linear relationship except for regions where the battery voltage is low (around 3.5 volts). Note that the reported battery level in Figure 3 is only around 60% even after several hours of charging using a wall charger (we'd expect the SPOT to be fully charged in that time). We suspect this is because the power controller needs to go through a few cycles of full charge/discharge before it can calibrate the battery capacity and produce accurate estimates for battery level.

4.4 Deep sleep disruptions

Ideally, the deep sleep count and the time spent in deep or shallow sleep should all increase monotonically with each reported sample. However, the deep sleep count plot in Figure 3 indicates several instances where the count stalled. In a couple of instances – at 2008-09-11 16:30 and 2008-10-02 15:00 – the SPOT recovered automatically. In all others, either the SPOT had to be manually reset (to jolt it out of the stalled state) or it rebooted on its own after draining the battery. These reboots, whether automatic or forced, show up as dips in the last three plots in Figure 3. The main issue exposed by our experiments is a software bug that keeps the SPOT either in shallow sleep or fully active when it ought to have been in deep sleep. Figure 6 provides a closer look at the disruptions on September 23 and 28. In both instances, the time spent in deep and shallow sleep stops increasing indicating that the SPOT stays active causing the battery to discharge rapidly.

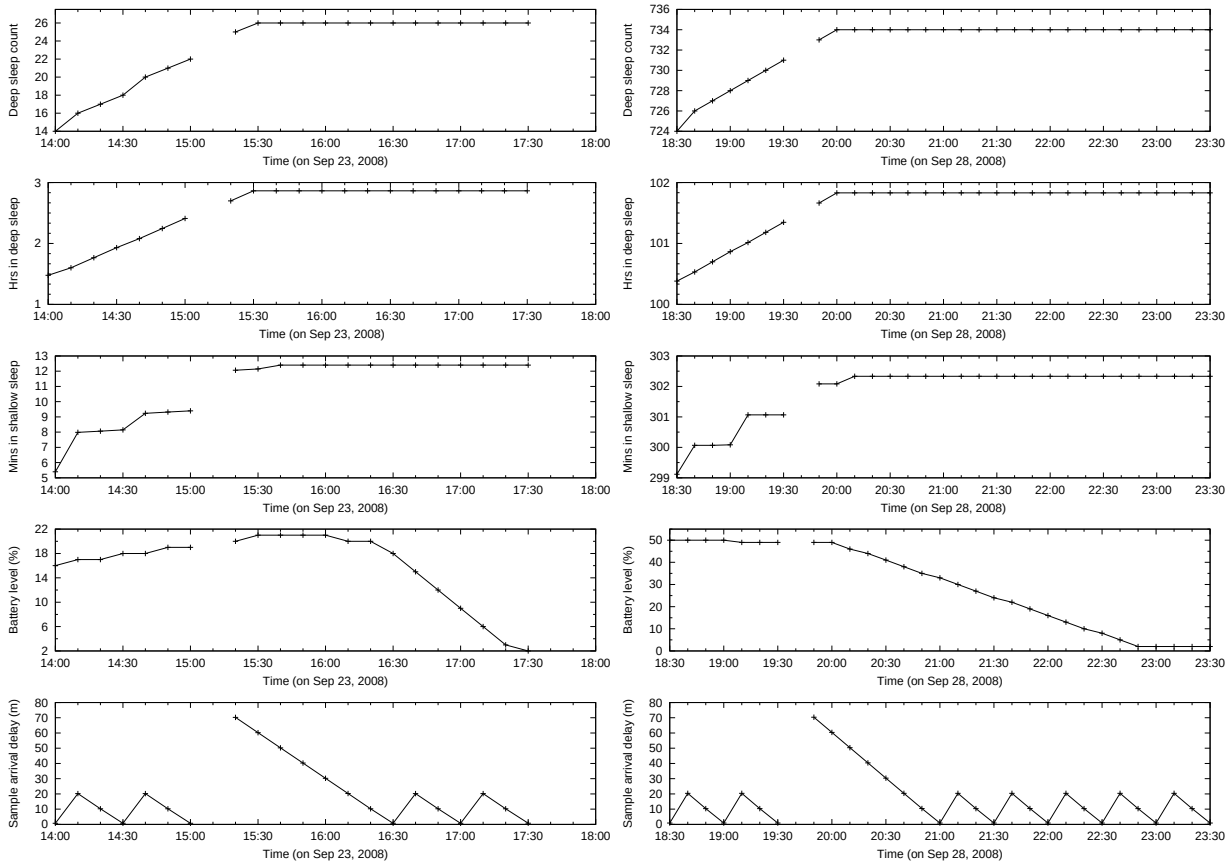


Figure 6: A closer look at the disruptions in deep sleep on September 23 and 28. On both occasions, the SPOT stayed active and the battery drained quickly.

Recall that our data collection framework does not necessarily send each sample as it is collected. Instead, it allows the radio transmission to be scheduled less frequently than sensor sampling. Early on, we decided to record and store the *data reception* or *arrival* time stamp (when a sample is inserted into the database) in addition to the time stamp when the sensor was sampled on the SPOT as a means for studying the delay introduced by our setup. We expect the sample arrival delay measured by (*arrival time* − *sample time*) to display a saw-tooth pattern under normal operation. With our chosen schedules, samples at 10 minutes and 40 minutes past the hour would be delayed by 20 minutes. Those at 20 minutes and 50 minutes past the hour would be delayed by 10 minutes and those on the hour and 30 mins past the hour would have nominal delays. This is visible in the bottom most plots in Figure 6. We also observe that the stall in deep sleep count is preceded by the loss of a sample and an unusually large spike in the sample arrival delay. This observation was found to apply to all instances of the mysterious stalls. This serendipitous discovery was one of the first hints that a possible bug in the networking code was responsible for deep sleep disruption.[2]

4.5 Anomaly detection and debugging

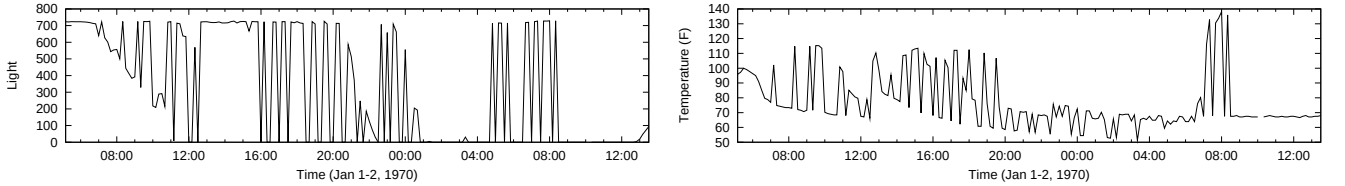
The previous section illustrates how monitoring the deep sleep count and sample arrival delay can help to detect situations where experimental observations deviate from normal behavior. We recommend that

[2]As of this writing, a potential fix for the bug has been identified but hasn't been completely verified.

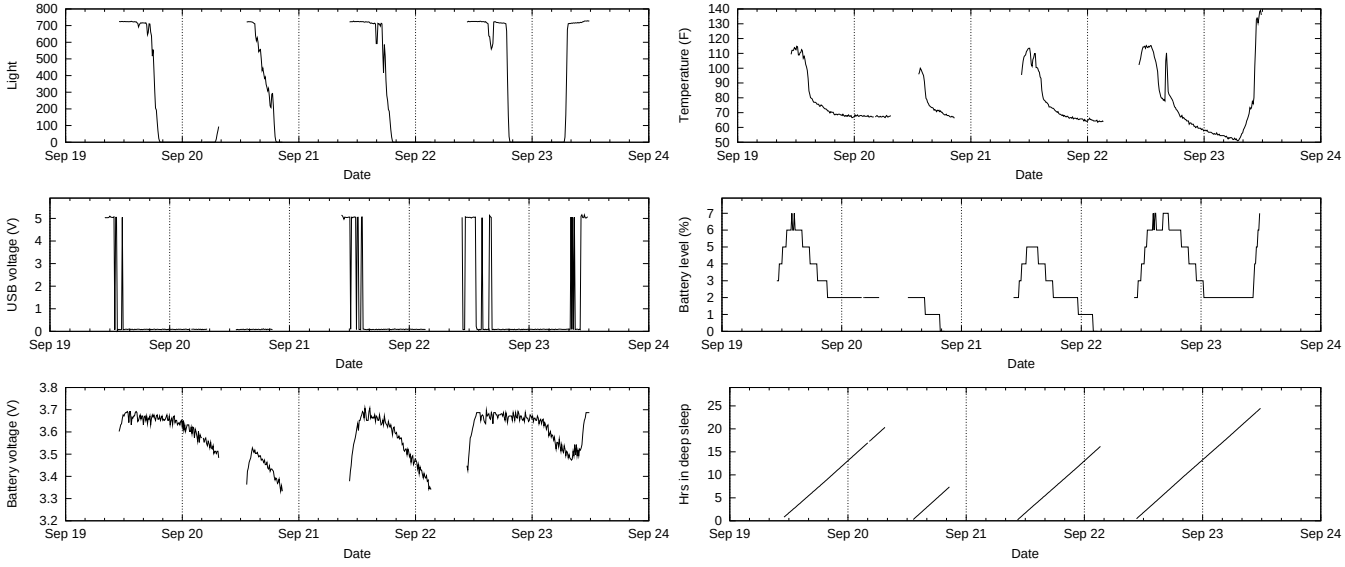
any data collection framework should include a callback mechanism to alert the application of unusual behavior and let the programmer specify different responses depending on the nature of the anomaly, *e.g.*, resetting a SPOT when deep sleep count has stalled unexpectedly. Anomalies are often indicative of bugs and mechanisms for their detection go hand-in-hand with tools that offer better visibility into the state of a SPOT. For example, new over-the-air commands than can report on the state of active threads and the routing software (routing table entries, routing manager threads) would have greatly simplified the task of debugging deep sleep stalls.

4.6 Recreating missing plots

The solar SPOT first ran down its batteries completely on September 18, 2008. For several days thereafter, it came alive every morning around 10 am when the solar panel had enough energy to charge and died soon after the following midnight. Since the SPOT's internal clock resets to January 1, 1970 when the battery is drained, the sample timestamp on each of these days reported the same date. The *arrival* time stamp, however, allowed us to separate out sample streams from different days (see Figure 7). Our decision to store arrival times proved to be fortuitous in this regard. Our framework now implements a time synchronization mechanism to prevent a recurrence of this phenomenon – SPOTs that have lost their clocks can reset them correctly upon hearing a beacon from the basestation.



(a) Light and temperature plots for the region A-B before any timestamp adjustments.



(b) Reconstructed data plots for the region A-B after timestamp adjustments.

Figure 7: Between September 18 and 23, 2008 the Sun SPOT reset its clock and reported data samples with a timestamp around January 1, 1970 on each of those days. This region is marked A-B in Figure 3. The two plots shown at the top were obtained before any adjustments to the sample time and the interleaving of data streams from multiple days makes it impossible to interpret anything meaningful. The bottom six plots were obtained after correcting the sample timestamp using the sample's *arrival time* at the database.

4.7 More is more

Our experiments reinforce the value of collecting as much data as possible with each sample. Including multiple sensor values with different inter-relationships can help deduce events easily. For example, by looking at the plot of USB voltage and light or temperature one can easily figure out if a SPOT is connected to a wall charger or to a solar panel. In the former case, the USB voltage stays high independent of the fluctuations in light or temperature readings. In the latter case, USB voltage is strongly correlated to high values on the light or temperature sensor. Similarly, it is beneficial to store all available meta-information into the database, *e.g.*, storing sleep notifications helps us differentiate between situations when the SPOT thought it was going into deep sleep but didn't (probably because one of the drivers vetoed deep sleep) versus when a SPOT didn't think it was ready to sleep.

5 Summary

These experiments represent only the first step in preparing the Sun SPOTs for unattended, long-term data collection. In particular, most real-world deployments would require data collection over a multi-hop network and reliably relaying data through SPOTs that are awake only intermittently will likely require additional changes to our software stack. Nevertheless, these experiments confirm that by combining duty cycling with renewable energy sources, one can successfully use Sun SPOT devices in long-term experiments. In particular, a SPOT left outdoors with a medium size solar panel can actively gain charge on a typical sunny day while sampling its built-in sensors once every ten minutes and transmitting that data once every thirty minutes. Based on our interactions with several potential customers (USGS scientists, data center administrators and telecom researchers), this sampling rate is adequate for a large class of applications.

6 Acknowledgments

The author wishes to thank Ron Goldman, Keith Hargrove and Arshan Poursohi for their help with various aspects of the experiments. In particular, Ron Goldman wrote the original SensorFramework software which is used in these experiments with some enhancements. Keith Hargrove built the special charging circuitry shown in Figure 2 and Arshan Poursohi supplied the solar panel used in these experiments.

References

[1] Sun Microsystems, Inc., "Sun Small Programmable Object Technology," http://www.sunspotworld.com/.

[2] Sun Microsystems, Inc., "Sun SPOT Developer's Guide," http://www.sunspotworld.com/docs/.

[3] SparkFun Electronics, "Solar Cell PRT-07840," http://www.sparkfun.com/commerce/product_info.php?products_id=7840.

[4] "SensorFramework," https://svn.sunlabs.com/svn/SunSPOT/InternalDemos/trunk/SensorFramework/ (requires login).

[5] Google, Inc., "Annotated Time Line," Google Visualization Gallery, http://code.google.com/apis/visualization/documentation/gallery/annotatedtimeline.html.

[6] Mort Bay, "Jetty WebServer," http://www.mortbay.org/.

[7] “Yggdrasil: A Framework for long-term sensor data collection on Sun SPOTs,” https://yggdrasil.dev.java.net/.

[8] Bart Eisenberg, “Embedded Computing for the Rest of us: Project Sun SPOT,” http://gihyo.jp/admin/serial/01/pacific/200811.

About the Author

Vipul Gupta is a Distinguished Engineer at Sun Microsystems Laboratories where his research interests include next-generation computing devices, networking protocols and resource efficient cryptography. Besides authoring several internet-drafts and RFCs at the IETF, he has published over thirty technical articles in refereed journals and conferences. Gupta has also contributed code to open source projects including Firefox, OpenSSL, and Apache. At Sun, he has been a two-time recipient of the Innovation Award (in 2004 and 2008) and an honoree of the Computerworld Horizon Award 2006. His development of the world's smallest secure web server, Sizzle (about the size of a quarter-dollar coin) received the Mark Weiser Best Paper Award at the IEEE Pervasive Computing and Communications Conference in 2005. He received his Ph.D. in Computer Science from Rutgers University and a B.Tech in Computer Science and Engineering from the Indian Institute of Technology, New Delhi.

Sun Microsystems Laboratories
16 Network Circle
Menlo Park, CA 94025

Experiments with a Solar-powered Sun SPOT

Vipul Gupta

SMLI TR-2009-178