Managing energy consumption as an architectural quality attribute

Editor: Eoin Woods
Endava
eoin.woods@endava.com

Managing Energy Consumption as an Architectural Quality Attribute

Rick Kazman, Serge Haziyev, Andriy Yakuba, and Damian A. Tamburri

ENERGY USED TO be free, or so we thought. In the past, software architects rarely considered software's energy consumption. Those days are gone. With mobile devices as the primary form of computing for most people,[1] with the increasing industry and government adoption of the IoT, and with the ubiquity of cloud services as the backbone of our computing infrastructure, energy has become an issue architects can no longer ignore. Energy is no longer "free" and unlimited. Mobile devices' energy efficiency affects us all, and large corporations are increasingly concerned with their server farms' energy efficiency. *Forbes* reported that in 2016, datacenters globally accounted for more energy consumption (by 40 percent) than the entire UK—about 3 percent of all energy consumed worldwide.[2]

At both the low and high ends, computational devices' energy consumption has become a crucial concern. This means that we, as architects, now must add energy efficiency to the long list of competing qualities we consider when designing a system. And, like every other quality attribute, it involves nontrivial tradeoffs: energy use versus performance, availability, modifiability, security, or time to market.

We can't hope to address all these concerns here, but we want to make two important general points:

- Treating energy efficiency as a quality attribute is no different from treating any other architectural quality.
- We can, with a small effort in experimentation and prototyping, and small design changes, substantially improve an application's energy use.

Both of these points are good news for architects! Their most immediate consequence is that we can reason about energy consumption architecturally. And, by making a relatively small investment in design experiments and design changes, we're repaid by enormous savings in energy use.

To illustrate these two points, we report here on a small case study we performed on the design of an IoT application—an automated weather station. The station reports telemetry from sensors related to the ambient temperature, humidity, wind speed, rainfall, leaf wetness, soil temperature, and so on. All the collected data is processed and used to help farmers achieve more efficient plant growth.

In this domain, energy efficiency is paramount: these weather stations are left unattended for long periods of time, might be snow-covered, and need to work in low-light conditions for weeks on end. The automated weather station we describe here was initially designed with attention to energy efficiency, but without explicitly modeling or measuring energy use. In the following, we describe our experiments, the design changes they helped motivate, their tradeoffs, and their energy savings.

The Experiments

The automated weather station is connected to a 12 V DC power source. Every 15 minutes it wakes up from a low-power deep-sleep mode and reports telemetry to the Azure

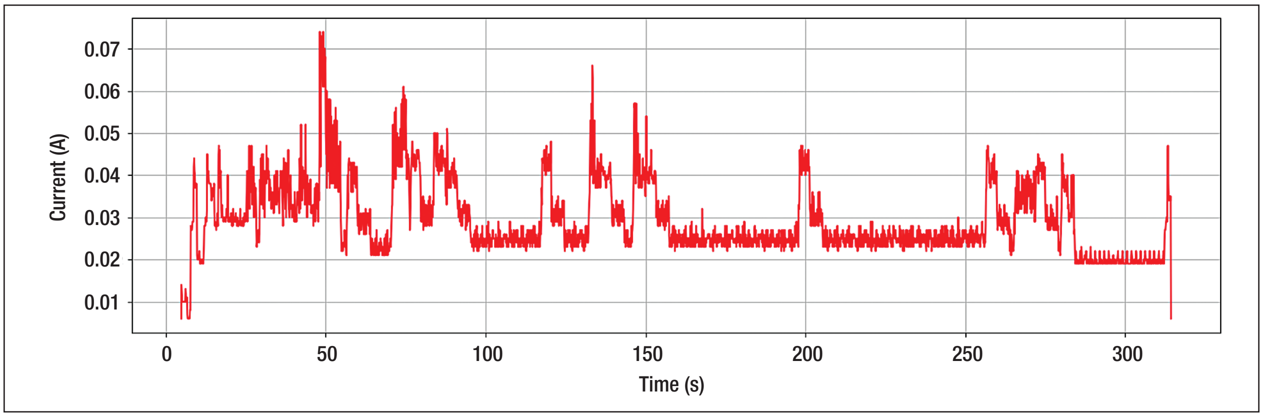


FIGURE 1. A sample plot of work mode for an automated weather station. The average current was 0.0295 A, and the energy consumption was 0.03048 Wh.

cloud using a 3G connection. After that, it suspends all running tasks and falls back into deep-sleep mode.

Between the DC power supply and the weather station, we inserted a current meter based on the MAX471 current-sensing amplifier. (For the MAX471 data sheet, see http://html.alldatasheet.com/html-pdf/73441/MAXIM/MAX471/125/1/MAX471.html.) This meter was connected to an analog sense pin of an AVR MCU (microcontroller). The MCU registered readings every 1 ms; after 100 readings, it sent the average value over a UART (universal asynchronous receiver–transmitter) to a PC. A simple logging application on the other side of the wire converted the UART input into a CSV (comma-separated values) file. A simple Python script then calculated energy consumption on the basis of the CSV file and plotted a graph.

To calculate energy consumption, we used these formulas:

$$E = P * T,$$

where P represents power and T represents time, and

$$P = I * V$$

for DC, where I represents current and V represents voltage.

Because voltage is constant in these devices, we assumed that the power consumed depended fully on the current draw and time. In all experiments, we measured the total energy consumption in watt-hours (Wh).

To simplify measurements and further calculations, we split the timeline for the automated weather system into two main components: work mode and sleep mode. Figure 1 depicts a sample plot of work mode, showing the power (current) consumed over 309 seconds. The average current was 0.0295 A, and the energy consumption was 0.03048 Wh.

Sleep mode consumed little energy, as we expected. The power consumption was nearly zero, with a watchdog process causing minor spikes of energy consumption. For example, for a sleep mode of 490 s, the average current was 0.001013 A, and the energy consumption was 0.0016538 Wh. Thus, sleep mode consumed roughly 3 percent as much energy as work mode.

On the basis of these measurements and some small simplifying assumptions, we estimated the energy consumption for 1 hour (assuming our typical sleep interval of 15 minutes):

- *Work + Sleep Duration* = 309/60 + 15 = 20.15 min
- *Cycles (C) per h* = 60/20.15 = 2.9776
- *Average Power Consumption per h* = $(E_{\text{work}} + E_{\text{sleep}}) * C$ = (0.03048 + 0.0002025 * 15) * C = 0.0335175 * 2.9776 = 0.0998 Wh

Given this baseline, we now describe our experiments.

Experiment 1

In this experiment, we changed the telemetry messages' payload format from plaintext to Google protocol buffers. The assumption was that the reduced message length should result in less time to send data and fewer chances to fail. However, this came with a tradeoff: increased memory and CPU use to transform messages to and from the protocol-buffer format.

Table 1 shows the kinds of messages being sent and their sizes, using plaintext and protocol buffers.

Table 1. For experiment 1, the kinds of messages being sent and their sizes (in bytes), using plaintext and Google protocol buffers.

Message	Plaintext			Protocol buffers		
	Header	Payload	Total	Header	Payload	Total
Status	375	822	1,197	387	275	662
Current telemetry	374	188	562	399	40	439
Historical telemetry*	384	460	844 (5 records)	402	400	802 (10 records)

* Historical telemetry was sent in batches. In the plaintext version, the batch contained five records. With protocol buffers, we could pack 10 records into a batch message, saving 42 bytes.

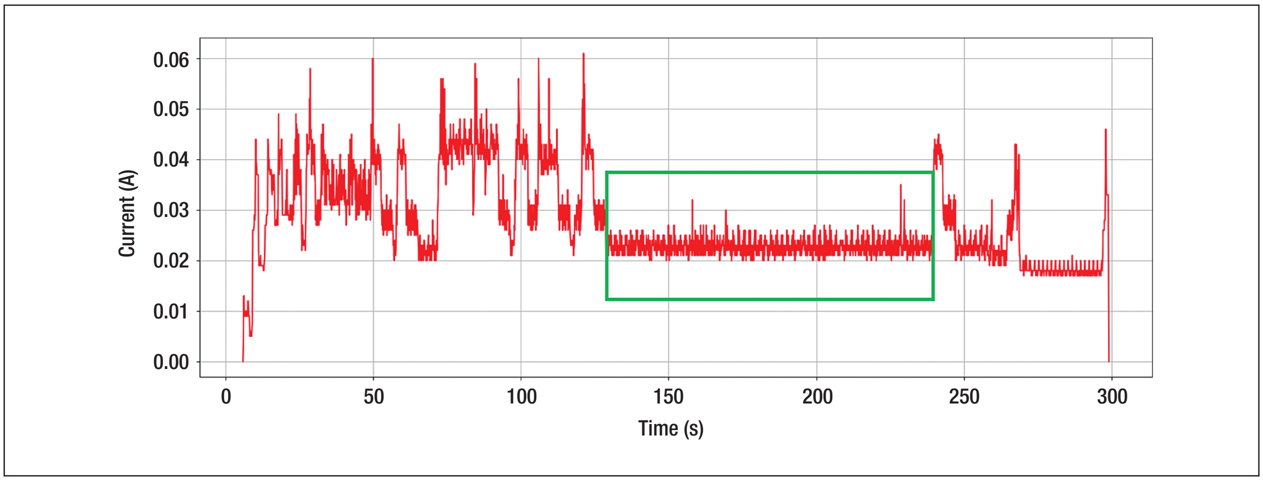


FIGURE 2. Energy use during message transmission, using protocol buffers. Using the buffers improved energy consumption by 8 percent.

But did this change save energy? Sending these messages using protocol buffers required 293 s, with an average current of 0.027362 A (see Figure 2). The power consumption was 0.0267519 Wh. Using the previous formula to calculate the average energy consumption per hour, we got

- *Work + Sleep Duration* = 293/60 + 15 = 19.883 min
- *C per h* = 60/20.15 = 3.017
- *Average Power Consumption per h* = (0.027362 + 0.0002025 * 15) * C = 0.0303995 * 3.017 = 0.0917 Wh

The difference was 0.0998 – 0.0917 = 0.0081 Wh (8 percent). Although 8 percent wasn't a huge improvement, it was nontrivial. Given protocol buffers' other advantages (they describe data using an interface description language and generate the code to handle it), this was clearly a win. But given that this energy savings wasn't huge, the lesson for architects is that tradeoffs among CPU time, memory, and message length must be assessed empirically.

Furthermore, while taking these measurements and observing power consumption graphs, we noticed a strange idle period, just before the device went into sleep mode (see the area in the green rectangle in Figure 2). Debugging and studying this issue led us to discover a bug in EEPROM persistence functionality and the peripheral-device-scanning logic. The bug's details aren't important for this research; what's important is that we didn't notice this problem until we measured and visualized the power consumption. Because the device had been functioning as expected, it wasn't obvious that there was a bug until we measured the energy.

Following are the new measurements of device power consumption.

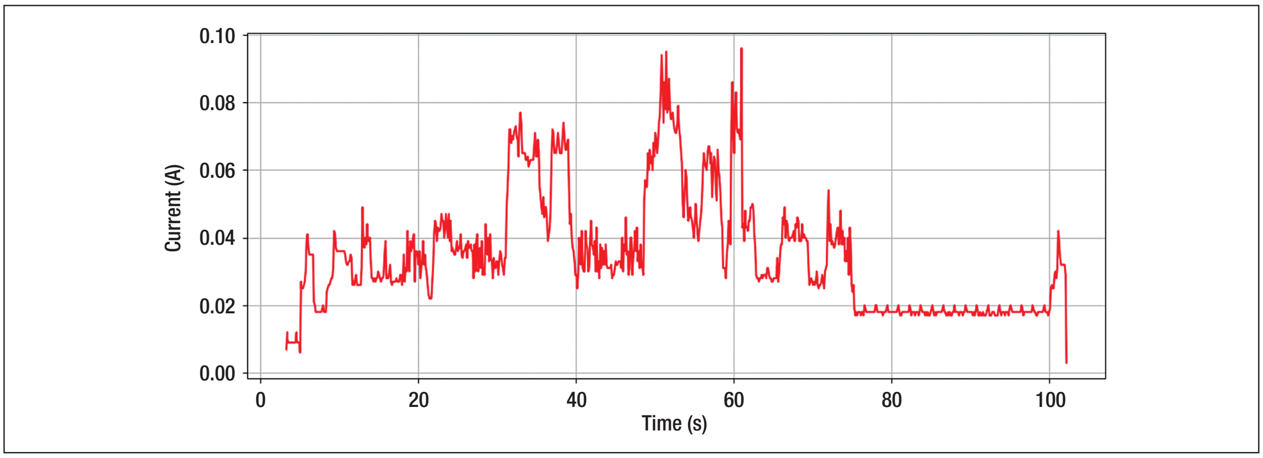


FIGURE 3. Energy use during message transmission after a bug fix. The difference, compared to this experiment's original results, was a 42 percent improvement.

By fixing this bug, we decreased the time required to perform all duties during the polling cycle by almost two-thirds, from 293 to 99 s (see Figure 3). The energy consumption was 0.0116034 Wh. Using the previous values, we recalculated the total power consumption:

- *Work + Sleep Duration* = 99/60 + 15 = 16.65 min
- *C per h* = 60/20.15 = 3.603
- *Average Power Consumption per h* = (0.0116034 + 0.0002025 ∗ 15) ∗ *C* = 0.014641 ∗ 3.603 = 0.0527 Wh

The difference, compared to this experiment's original results, was 0.0917 – 0.0527 = 0.039 Wh (42 percent). The lesson is that you can't manage it if you don't measure it! By measuring energy consumption, we were able to increase our understanding of the inner workings of the automated-weather-station application.

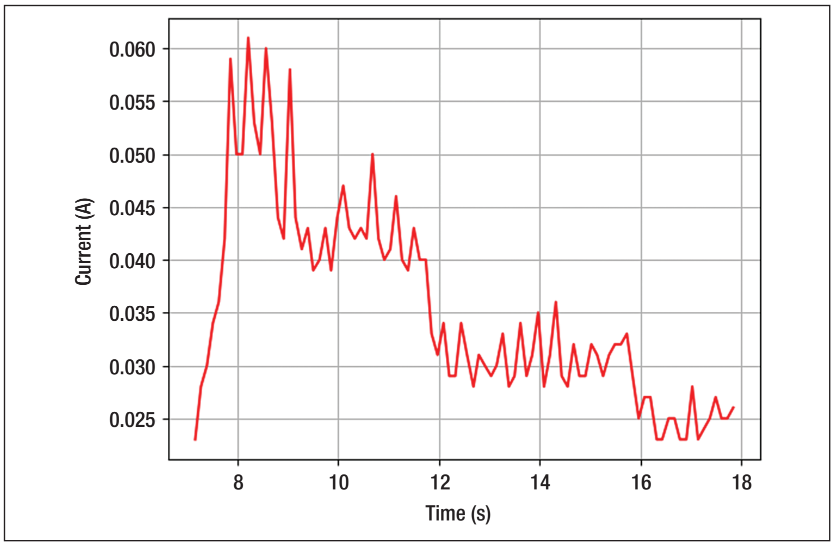


FIGURE 4. The energy profile of a single batch transmission. On the basis of this profile, we could estimate the energy use.

Experiment 2

This experiment aimed to reduce the number of polling cycles, leading to longer sleep times with significantly less power consumption. The tradeoff here is that more data would be sent in fewer batches. This could have the effect of increasing the time and power needed to send a batch, and it also introduced the need for a send-fail-retry mechanism in case of message failure.

As we discovered from experiment 1, the weather station could send up to 10 historical records in a single batch. A single batch had the energy profile shown in Figure 4, consuming 0.00127 Wh over approximately 10.5 s.

On the basis of this profile, we could now estimate the energy use. That is, given our measurements, we could now form a model of the quality attribute that was no different from a model of performance or availability. For simplicity's sake, we

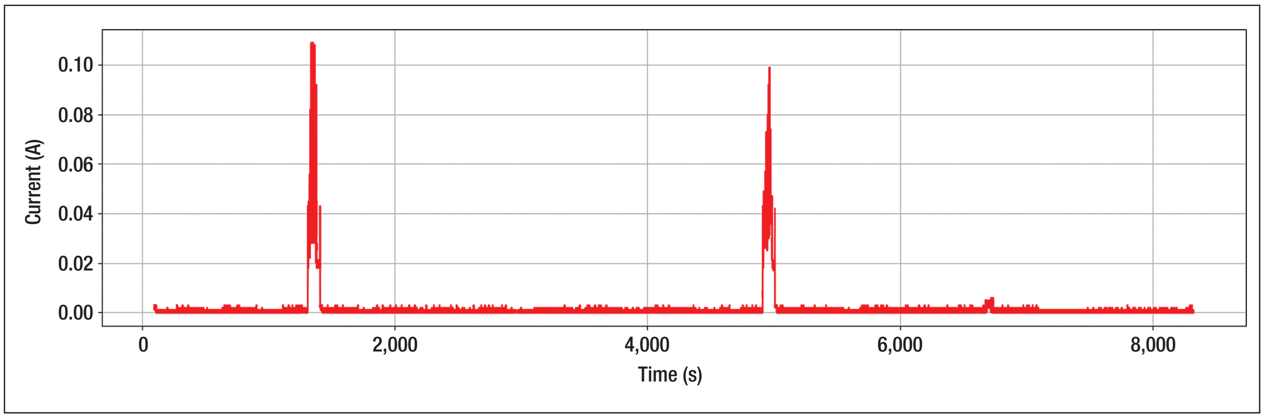


FIGURE 5. The energy profile of a weather station running for several hours with a polling interval of 3,600 seconds. Most of the sleep time was recorded as 0 A, meaning that the actual energy consumption was close to the meter's tolerances.

Table 2. The differences between the experiments.

Setup	Description	Consumption per hour (Wh)	Total energy savings (%)
Original	Plaintext payload and a 15-min polling interval	0.0998	0
Experiment 1	Binary format	0.0917	8
	Binary format + bug fix	0.0527	47
Experiment 2	A polling interval of 1 h	0.0137	86

chose a polling interval of 1 hour. So, the historical data would be sent in batches of four records (one record each 15 min). The predicted power consumption was

- *Work Time* = 99 s + 10.5 s = ~110 s
- *Consumed Power during Awake* = 0.0116034 + 0.00127 = 0.012874 Wh
- *Sleep Time* = 3,600 s = 60 min
- *Consumed Power during Sleep Time* = 60 * 0.0002025 = 0.01215 Wh
- *C per h* = 3,600/(110 + 3,600) = 0.97
- *Estimated Average Power Consumption per h* = $(E_W + E_S)$ *

C = (0.012874 + 0.01215) * 0.97 = 0.02427 Wh

For this experiment, we set up a weather station with a polling interval of 3,600 seconds and ran it for 8,225 seconds (a little over two hours); see Figure 5. The average current draw was 0.0011344 A, and the total power consumption was 0.0312867 Wh. So, we calculated the average power consumption per hour as

E/(*Duration*/3,600) = 0.0312867/(8,225/3,600) = 0.0312867/2.2847 = 0.013694 Wh

Comparing the results with our estimated consumption of 0.02427 Wh, we realized that there was some error in our calculations or our model. Taking a closer look at the raw recording, we noticed that most of the sleep time was recorded as 0 A, meaning that the actual energy consumption was close to the meter's tolerances. So, we were unable to measure such low current consumption precisely in real time, and could only trust sleep mode measurements made over long runs.

This insight highlights a new lesson: the need to build both architectural models and prototypes. On one hand, models let us efficiently reason about architectural quality attributes and their tradeoffs. On the other hand, prototypes mandate empirical testing, thus incrementally refining the assumptions built into both models and prototypes.

The difference in power consumption from Experiment 1 was 0.0527 – 0.013694 = 0.039006 Wh (42 percent). The difference from the original model (before we improved the design) was 0.0998 – 0.013694 = 0.086106 Wh (86 percent). Table 2 summarizes these differences.

As you can see, with some modest changes to the design, we reaped enormous energy savings, which would greatly increase the robustness of the IoT devices and the system as a whole. Despite operating in sometimes adverse conditions, these systems must never lose data, even if they're kept offline for 24 hours. With this 86 percent energy improvement, we could now meet this requirement.

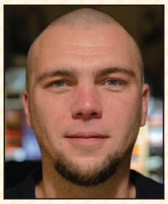
As computing moves toward greater scale and mobility, energy use will inevitably become a key concern for architects. We hope we've convinced you that energy can be treated like any other architectural quality attribute. It's no different, from the perspective of architectural design, than modifiability, performance, or availability. It can be modeled and prototyped, and we can reason about the design tradeoffs required to achieve better energy use. Of course, the design primitives, models, tools, and tradeoffs are specific to energy use, but the fundamental principles and reasoning methods for architectural design don't change.

As we said before, you can't manage what you don't measure. So, architects need to begin thinking about making their software more energy aware, monitoring it and adapting it to environmental or application conditions.

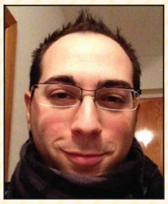
ABOUT THE AUTHORS

RICK KAZMAN is a professor of information technology management at the University of Hawai'i at Mānoa and a researcher at Carnegie Mellon University's Software Engineering Institute. Contact him at kazman@hawaii.edu.

SERGE HAZIYEV is the senior vice president of SoftServe's Advanced Technology Group. Contact him at shaziyev@softserveinc.com.



ANDRIY YAKUBA a senior IoT engineer at SoftServe. Contact him at ayakuba@softserveinc.com.



DAMIAN A. TAMBURRI is an assistant professor in the Jheronimus Academy of Data Science and Technical University of Eindhoven. Contact him at d.a.tamburri@tue.nl.

References

1. T. Bindi, "Mobile and Tablet Internet Usage Surpasses Desktop for First Time: StatCounter," *ZDNet*, 2 Nov. 2016; https://www.zdnet.com/article/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-statcounter.
2. R. Danilak, "Why Energy Is a Big and Rapidly Growing Problem for Data Centers," *Forbes*, 15 Dec. 2017; https://www.forbes.com/sites/forbestechcouncil/2017/12/15/why-energy-is-a-big-and-rapidly-growing-problem-for-data-centers/#65451c435a30.