Energy Isolation Required for Multi-tenant Energy Harvesting Platforms

Joshua Adkins
adkins@berkeley.edu
University of California, Berkeley

Bradford Campbell
bradjc@virginia.edu
University of Virginia

Branden Ghena
brghena@berkeley.edu
University of California, Berkeley

Neal Jackson
neal.jackson@berkeley.edu
University of California, Berkeley

Pat Pannuto
ppannuto@berkeley.edu
University of California, Berkeley

Prabal Dutta
prabal@berkeley.edu
University of California, Berkeley

ABSTRACT

Embedded systems have long been synonymous with special purpose, single stakeholder computing. However, as these systems have become more capable and the demands placed on them have become more varied and variable, embedded software is beginning to embrace multi-tenancy. While the general problem of supporting multiple users and processes on a computing platform has been well explored in computer science, the challenges of supporting multiple users with competing desires on a highly energy-variable system remain unexplored. On an energy-harvesting platform, incoming energy needs to be distributed between stakeholders, and users accessing shared platform resources should be charged for the energy use of those resources. Furthermore, with system designers and application creators being increasingly removed from each other, the software environments of energy-harvesting platforms must provide primitives that enable applications to adapt to system variability. We explore several initial techniques for solving these problems and demonstrate them using Signpost—a modular, energy-harvesting platform for city-scale sensing.

CCS CONCEPTS

• **Computer systems organization** → **Embedded software**; • **Software and its engineering** → *Power management*; • **Hardware** → Energy distribution;

KEYWORDS

Energy Harvesting, Multi-tenancy, Energy Isolation

ACM Reference Format:
Joshua Adkins, Bradford Campbell, Branden Ghena, Neal Jackson, Pat Pannuto, and Prabal Dutta. 2017. Energy Isolation Required for Multi-tenant Energy Harvesting Platforms. In *Proceedings of 5th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems (ENSsys'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3142992.3142995

All authors contributed equally to this work. For questions, email adkins@berkeley.edu

ENSsys'17, November 5, 2017, Delft, The Netherlands

ACM ISBN 978-1-4503-5477-6/17/11…$15.00
https://doi.org/10.1145/3142992.3142995

1 INTRODUCTION

For a long time, embedded systems have been considered a special case of computing. The isolation and protection afforded by virtual memory, kernels, and processes required for modern computing were too expensive, too complex, and ultimately unnecessary for single-purpose motes. Now, however, we are witnessing a shift, as the expanding architectural capabilities and resources of microcontrollers facilitate concurrent applications, and the economies of sharing deployed infrastructure demand improved utilization [7].

Another defining feature of embedded systems is extreme care of energy utilization, with systems that aim to last from days to years on a single battery. Systems looking toward the future, with projections of trillions of connected devices, are recognizing that the limited lifespan afforded by batteries alone is insufficient, and exploit energy scavenging techniques to achieve near limitless lifetimes. Unfortunately, energy harvesting often introduces energy intermittency, as harvesting sources and quality vary over time, making reliable software quite challenging [8]. As a consequence, hybrid systems—energy harvesters with a large energy store—are emerging that transform *energy intermittency* into a less volatile and more manageable *energy variability*, energy that is predictable over reasonably long windows of time.

While energy variability means developers no longer need worry about applications dying on any arbitrary line of code, its guarantees are insufficient to allow reasoning about reliable application performance, especially in the face of multi-tenancy. An energy-conscious application may still face unexpected shutdowns if a co-located application faults and burns power or is simply inefficient. To address this, we argue for *energy isolation*. Akin to memory isolation or processor scheduling, energy isolation affords energy guarantees to an application, independent of other system activity.

Within the isolation contract, platforms have two responsibilities. First is the allocation of incoming energy. Allocation policies can be used to enforce application priority, but also need to be crafted with care to preserve system functionality. Second, to adapt to energy variability, applications need information about the current energy state. To understand how energy isolation might work in practice, we implement these concepts on an energy harvesting platform and demonstrate the effect of various application-level policies.

There are several existing general-purpose software systems for energy harvesting, some focusing on support for high intermittency [4, 9] and others on profiling application energy use at compile time [6]. We believe that a focus on energy isolation can

augment and be supported by these systems. While energy adaptivity hopes to avoid intermittency, failures will still occur and need to be handled correctly. Similarly, a compile-time understanding of application energy use can significantly aid developers.

2 MANAGING APPLICATION ENERGY

Isolation. Operating systems use virtual memory, filesystems, and other forms of isolation to provide proccesses the illusion of running on a private machine. In practice, such isolations are minimums, once granted memory a process can rely on keeping the memory until choosing to return it (though some phones explore APIs where the OS can *ask* for some back). For energy-constrained systems, the minimums guaranteed by isolation allow a developer to reason about whether their application will still be able to run at a future time and how much it will be capable of accomplishing.

To enable this reasoning, one guarantee must be kept: an application's allocated energy must only decrease in a predictable fashion. The application may run, using energy and decreasing its allocation correspondingly. Alternatively, the application may use platform services, such as system calls or commands to an external radio or sensors. Energy used by these shared system resources can be charged against the application which used them. Finally, the platform itself may have a constant overhead that is split between all applications. From this, an application can reason about how long it could run on its current allocation, how long it will be able to run at a future time, and what the impact of its operations will be on its energy allocation.

Providing this invariant leads to a challenge for the platform, which now must accurately measure energy use of applications and services on the system as well as the total stored energy. Battery gas gauges are a fairly standard solution to this problem, but real-time measurements of the power draw of various subsystems can be expensive to implement. We hope to see the adoption of alternative energy-metering techniques such as [5] and the integration of standard techniques into COTS components. This would enable pervasive, real-time energy measurement.

This leads to the question of how often system services should report application energy usage to the platform. A radio could have a gas gauge measuring its energy use that is read whenever the currently running application changes. Alternatively a co-processor could track requests from each application to the radio and the time spent on them, updating the platform as each request completes. The granularity of these updates influences how much energy an application could potentially overspend before being detected.

The guarantee of energy isolation is similar to the goals of Virtual Battery [3], which proposed virtualization of platform energy on resource-constrained systems, emphasizing the illusion of a single-tenant system. We strongly agree with energy virtualization as a key primitive in low-power embedded systems. Furthermore, for systems with energy variability, we identify two additional platform needs: an allocation policy for incoming energy and mechanisms for understanding and adapting to the system energy state.

Energy Allocation Policy. When energy-constrained systems grow to include energy-harvesting systems, the energy isolation guarantee becomes a worst-case bound. At any point, the system may gain additional energy, which can be allocated to any application depending on system policy.

The simplest such policy is a fair sharing policy. The total battery capacity is divided evenly among all applications, creating equally-sized "virtual allocations". As applications use energy, their allocation store is reduced, and when exhausted, the application is no longer scheduled. As the platform harvests energy, charge is split equally amongst applications. If a virtual allocation is full, excess charge is distributed evenly among the remaining applications.

Such a system provides energy isolation and allows predictable application execution. However, it does not necessarily allow for optimal system functionality. Application priority could be included in the distribution policy. Higher priority applications could get larger virtual capacity or be first to claim incoming energy. Even with unfair sharing, energy isolation guarantees hold so long as no application is capable of stealing energy from another application.

The frequency of this energy distribution should also be considered. While energy is continuously collected through the harvester, providing energy immediately to applications which are disabled may actually be a poor choice. Providing enough energy to begin operation but not complete it brings back the problems of energy intermittency. For applications which are active, however, providing timely updates to their virtual allocation can allow them to better plan and adapt.

Energy Abstractions. To adapt to variability and work within the allocations of an energy policy, applications should be provided with high fidelity information on energy availability and usage. Specifically, applications should be able to identify the amount of energy they are currently allocated and measure how much energy they require to perform individual tasks. The latter implies that applications have the ability to start and stop an energy metering service on demand (enabling energy benchmarking of a task) and to identify with sufficient granularity the energy being ascribed to them for the use of system-level services. While the energy required to perform an individual task may be known a priori, we envision a future in which applications have the ability to run on multiple hardware platforms and platforms have the ability to adapt to their environment, necessitating in situ energy metering.

Higher-level application interfaces for energy adaption can then be built on top of these primitives. For example, Dewdrop [2] dynamically changes task execution frequency based on available power, attempting to reach a steady-state of energy used and received. For run-to-completion tasks, this interface may be sufficiently expressive. Eon [10] provides a richer interface, allowing applications to be split into tasks, with a notation of the energy states in which a given task should be run. At runtime, the application can select which task to run at a given time based on available energy. Both of these works represent application adaptivity models which we believe are supported by our lower-level abstraction for accessing system-level energy usage.

3 EVALUATION

We explore these ideas in the context of an existing system. The Signpost platform is a modular, solar energy-harvesting, and signpost-mountable system for enabling easy and ubiquitous installation of smart city applications [1]. The platform includes eight plug-in

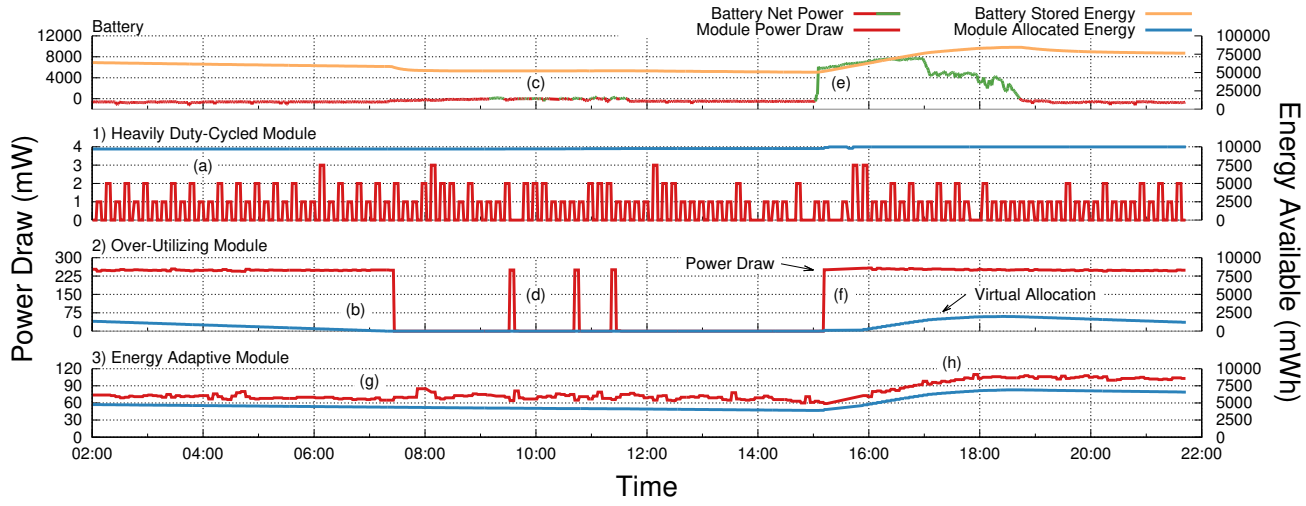

Figure 1: Effect of energy distribution policy on module operation and system energy. Shown are three modules operating for a day within the energy distribution policy described in Section 3. The three modules each employ unique management of their allocated energy. Module 1 asks the platform to perform duty cycling, and is turned on once every 10 minutes, evident in the duty cycling events (a); in this way it is able to achieve extremely low average power with little developer effort. Module 2 consistently draws a high amount of power, causing the platform to cut off the module when it has used all of its share (b). When the battery is charging slowly (c), it is enabled several times, only to quickly use its entire allocation and get disabled (d). Eventually the battery is in more direct sunlight (e) and enough energy is distributed to the module for it to remain enabled (f). Module 3 adapts the amount of energy it uses to the amount of energy it has remaining in its allocation. This can be observed in the lowering average power through the night (g) and the increasing average power as the battery charges (h). These situations show the capability of the software interfaces to support sensing applications in a variable energy environment.

module slots for independent microprocessor based systems. Modules can be sensors or provide resources and services like storage or processing to the entire system. These modules have access to the system-managed shared energy store. The platform includes metering and power control for each module. Some modules provide shared services, such as a radio module that provides network connectivity. Such service modules report back to the controller which applications should be charged for their energy use.

We implement a simple energy policy in which we allocate each sensor module a virtual allocation, which has a capacity of 10% of the physical battery on Signpost. When a module consumes energy, it is subtracted from this virtual allocation, and when energy is harvested, it is evenly distributed to the virtual allocations equally at ten minute intervals. If a battery is full, the energy is redistributed to the other modules' virtual allocations, and if a module uses all of its allocated energy, it is turned off until more energy is harvested.

The platform provides the software interface described in Section 2, and also offers an API to manage a module's power by duty cycling it. We use this interface to implement three different energy adaptive modules. The first module maintains low power by asking the platform to disable it for extended periods of time, effectively a low rate duty cycle. The second module is rather high power, but performs no energy management and is eventually turned off, only to be turned on again during mid day when the battery is charging. The third module uses the energy query mechanism along with the duty cycle API to adapt its energy usage to the amount of energy it has remaining, resulting in lower average power throughout the night, only to increase power consumption after harvesting energy during the day. We run these modules for a day on an outdoor Signpost platform, with the results presented in Figure 1.

4 CONCLUSIONS

Multi-tenant embedded systems have arrived, and the efficiency gains from multiprogramming and utility of dynamic adaptation will continue to push the demand for sharing of deployed resources. At the same time, energy harvesting is becoming commonplace, and the lifetime and deployability benefits afforded by harvesting will continue to push the demand for such systems. These two realities must find a means to interoperate. This paper argues that a simple guarantee is sufficient to allow applications to reason and adapt on multi-tenant, energy-harvesting platforms. Providing additional access to system energy information can expose a wide range of usability, reliability, and complexity tradeoffs for application developers. Incorporating fairness, priorities, and richer meting comprehension into energy interfaces remains ripe for future work.

ACKNOWLEDGMENTS

This work supported in part by the TerraSwarm Research Center, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under grant numbers DGE-1256260 and DGE-1106400.

REFERENCES

[1] Joshua Adkins, Bradford Campbell, Branden Ghena, Neal Jackson, Pat Pannuto, and Prabal Dutta. 2016. Demo Abstract: The Signpost Network. (2016).

[2] Michael Buettner, Ben Greenstein, and David Wetherall. 2011. Dewdrop: an energy-aware runtime for computational RFID. In *Proc. USENIX NSDI*. 197–210.
[3] Qing Cao, Debessay Fesehaye, Nam Pham, Yusuf Sarwar, and Tarek Abdelzaher. 2008. Virtual battery: An energy reserve abstraction for embedded sensor networks. In *Real-Time Systems Symposium, 2008.* IEEE, 123–133.
[4] Alexei Colin and Brandon Lucia. 2016. Chain: Tasks and channels for reliable intermittent programs. In *OOPSLA'16*.
[5] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler. 2008. Energy Metering for Free: Augmenting Switching Regulators for Real-Time Monitoring. In *International Conference on Information Processing in Sensor Networks, 2008.* IEEE, 283–294.
[6] Josiah D Hester, Travis Peters, Tianlong Yun, Ronald A Peterson, Joseph Skinner, Bhargav Golla, Kevin Storer, Steven Hearndon, Kevin Freeman, et al. 2016. Amulet: An Energy-Efficient, Multi-Application Wearable Platform.. In *SenSys'16*.
[7] Amit Levy, Daniel B. Giffin, Bradford Campbell, Branden Ghena, Pat Pannuto, Prabal Dutta, and Philip Levis. 2017. Multiprogramming a 64 kB Computer Safely and Efficiently. In *SOSP'17*.
[8] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 71.
[9] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2012. Mementos: System support for long-running computation on RFID-scale devices. *ACM SIGPLAN* 4 (2012).
[10] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D Corner, and Emery D Berger. 2007. Eon: a language and runtime system for perpetual systems. In *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 161–174.