Mementos: System Support for Long-Running Computation on RFID-Scale Devices

Benjamin Ransford
Department of Computer Science
University of Massachusetts Amherst
ransford@cs.umass.edu

Jacob Sorber
Institute for Security, Technology, and Society
Dartmouth College
jacob.m.sorber@dartmouth.edu

Kevin Fu
Department of Computer Science
University of Massachusetts Amherst
kevinfu@cs.umass.edu

Abstract

Transiently powered computing devices such as RFID tags, kinetic energy harvesters, and smart cards typically rely on programs that complete a task under tight time constraints before energy starvation leads to complete loss of volatile memory. *Mementos* is a software system that transforms general-purpose programs into interruptible computations that are protected from frequent power losses by automatic, energy-aware state checkpointing. Mementos comprises a collection of optimization passes for the LLVM compiler infrastructure and a linkable library that exercises hardware support for energy measurement while managing state checkpoints stored in nonvolatile memory. We evaluate Mementos against diverse test cases in a trace-driven simulator of transiently powered RFID-scale devices. Although Mementos's energy checks increase run time when energy is plentiful, they allow Mementos to safely suspend execution when energy dwindles, effectively spreading computation across zero or more power failures. This paper's contributions are: a study of the runtime environment for programs on RFID-scale devices; an energy-aware state checkpointing system for these devices that is implemented for the MSP430 family of microcontrollers; and a trace-driven simulator of transiently powered RFID-scale devices.

Categories and Subject Descriptors C.3 [*SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS*]: Real-time and embedded systems

General Terms Design, Experimentation

Keywords Mementos, RFID-Scale Devices, Computational RFID, Energy-Aware Checkpointing

1. Introduction

Demand for tiny, easily deployable computers has driven the development of general-purpose *transiently powered computers* that lack both batteries and wired power, operating exclusively on energy harvested from remote supplies or the environment. Such devices range from *computational RFIDs* [36]—microcontroller-based devices that harvest RF from readers and communicate via RFID protocols—to general-purpose batteryless sensor devices [45].

Computing under transient power conditions is a challenge. Transiently powered RFID tags use simple state machines instead of supporting general-purpose computation. Contactless smart cards perform more complicated special-purpose computations (e.g. cardholder authentication); however, they offer no execution guarantees, and instead rely on the user to provide the needed RF power for a sufficient period of time. When energy consumption outpaces energy harvesting, these computations fail and must restart from scratch, when adequate energy becomes available.

With ultra-low-power microcontrollers (MCUs), tiny *programmable* devices can perform computation and sensing under RFID-scale energy constraints; however, these MCUs consume more power than conventional RFID circuitry, and energy consumption can easily outpace harvesting, resulting in frequent power loss.

Today, programs that run CPU-intensive operations like cryptography on these devices are pessimistically and painstakingly hand-tuned to complete within a short time window (often under 100 ms) [7, 9]. The usefulness and power of RFID-scale devices can be dramatically improved if designers can confidently write programs without being limited by power failures.

Mementos is a software system that enables long-running computations to span power loss events by combining compile-time instrumentation and run-time energy-aware state checkpointing [1]. At compile time, Mementos inserts function calls that estimate available energy. At run time, Mementos predicts power losses and, when appropriate, saves program state to nonvolatile memory. After a failure, program state is restored and execution continues rather than restarting from scratch.

This paper makes the following contributions: (1) An energy-aware state checkpointing system that splits program execution across multiple lifecycles on transiently powered RFID-scale devices. The system is implemented for the MSP430 family of microcontrollers, requires no hardware modifications to existing devices, and operates automatically at run time without user intervention. (2) A suite of compile-time optimization passes that insert energy checks at control points in a program. The optimization passes employ three different instrumentation strategies that favor different program structures. (3) A trace-driven simulator to evaluate the behavior of programs on transiently powered RFID-scale devices. The simulator, modeled after a prototype hardware device with an off-the-shelf microcontroller, takes executable code as input and simulates power loss events during runs. We evaluate the simulator's accuracy and Mementos's performance under simulation in Section 5.

ASPLOS'11, March 5–11, 2011, Newport Beach, California, USA.

[1] In the indie film *Memento,* the main character would unpredictably lose short-term memory, especially when sleeping. He checkpointed state with notes and tattoos in an attempt to execute long-running tasks.

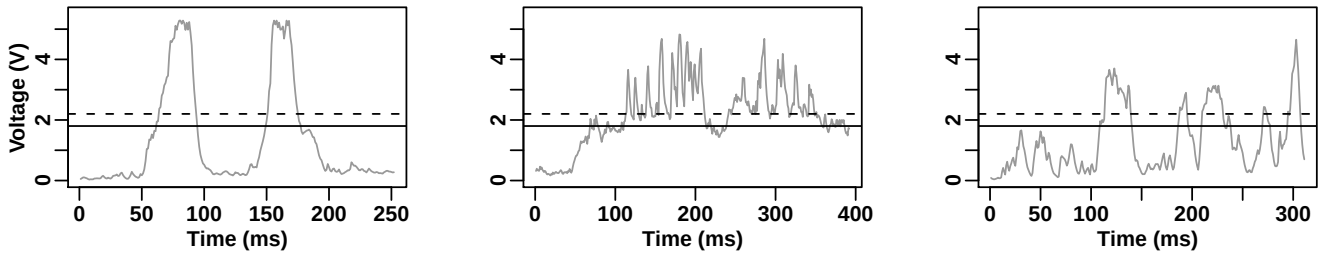

Figure 1. Energy availability under RF harvesting is difficult to predict on a transiently powered computer (TPC), threatening the successful completion of long-running programs. These plots show the output of a prototype TPC's energy-harvesting frontend during three different smooth movements within 2 m of an RFID reader. The dashed line at 2.2 V represents this prototype's nominal minimum voltage for flash writes. The solid line at 1.8 V depicts the prototype's nominal minimum operating voltage, below which it loses volatile state.

The compile-time analysis and program transformation components of Mementos are built on the LLVM compiler infrastructure [21]. Our simulation of a transiently powered device is implemented as a set of enhancements to MSPsim [14] and is guided by the hardware parameters of a WISP [39] (Revision 4.1) prototype computational RFID.

Applications. Mementos enables long-running or computationally intensive applications on RFID-scale devices. Moving computing into environments that are ill-suited to batteries and tethered power, promising applications include environmental monitoring where battery replacement is not practical, insect-scale wildlife tracking where batteries are too heavy, and implantable medical devices [29] where battery recharging might heat and damage surrounding tissue. Mementos aims to enable these new applications by extending the computational capabilities of transiently powered computers beyond simple programs.

An example of a long-running application that could benefit from Mementos's automatic checkpointing is compressive sensing [8]. RFID-scale devices can measure environmental phenomena—like temperature, acceleration, and light—that may exhibit informative trends over time spans larger than a few seconds. Compressive sensing maintains a set of frequently updated variables that collectively represent a sparse, compressible signal, preserving the structure and information of the signal with high probability. By providing automatic state checkpointing, Mementos would enable a compressive sensing program to accumulate measurements over many lifecycles separated by power loss events. Section 5 includes an evaluation of Mementos on a simplified sensing application that cannot complete under reasonable assumptions in a single lifecycle of our trace-driven simulator or the platform it models.

2. Computing on Transient Power

Multiple platforms at various stages of maturity enable batteryless, RFID-scale, transiently powered computing. The WISP [39] uses an MSP430 microcontroller [43] for computation and harvests energy from off-the-shelf RFID readers. A system-on-chip variant, the SoCWISP [46], is small (2.0 mm^2) and light enough to attach to insects in flight. The Blue Devil WISP [44] improves performance for RF harvesting and communication. The UMass WISP under development provides increased storage and supports more peripherals (`http://www.cs.umass.edu/~zhangh/`). The EnHANTs [16] platform under development aims to behave like an RFID tag and a sensor mote. All share the goal of enabling sensing and general-purpose computation under transient power.

Previously proposed applications for transiently powered computers include environmental monitoring [17], activity recognition [6], and cryptographic protocols [9]. Mote-class devices (e.g. Telos [35], TinyNode [12]) offer similar capabilities and can also be used in these applications, but their size, weight, and maintenance cost (i.e., dependence on batteries) significantly limits their deployability. Transiently powered devices potentially provide the benefits of programmability and general-purpose computing without the drawbacks associated with more powerful mote-class devices.

Despite their benefits, designing and deploying these systems is challenging. By definition, these systems cannot depend on continuous power. Figure 1 illustrates typical fluctuations in supply voltage that occur under RF energy harvesting. Prototype systems like the WISP use capacitors as short-term energy buffers. For a sense of scale, consider that a WISP's 10 μF capacitor can store roughly 100 μJ, whereas a Telos sensor mote's two AA batteries can store over 20,000 J—eight orders of magnitude more.

The amount of energy harvested from RF, solar, and other sources varies widely and is difficult to predict [31, 47]—a problem often compounded by device mobility. Consequently, unlike traditional computing systems, transiently powered systems frequently lose power and computational state as a rule, not as a rare exception. Experience with the WISP has shown that power failures every $\sim$100 ms are a reasonable expectation [9, 36]. Existing lightweight operating systems like TinyOS [22] boot too slowly—in an informal test, TinyOS on a TinyNode booted in 253 ms, and 193 ms without clock calibration—to provide robustness via OS services on an RFID-scale device. Under these conditions, long-running programs may never complete, as they restart their work after each failure. In the context of transiently powered devices, we refer to such long-running programs as *Sisyphean tasks*.[2]

A key to solving the problem of Sisyphean tasks on RFID-scale devices is that many general-purpose microcontrollers, notably the MSP430 found on WISP-derived devices, feature nonvolatile memory that can be written to at run time. The most common form of on-chip nonvolatile memory is *flash memory*, typically available on prototype RFID-scale devices in the amount of several kilobytes. Four complications make it nontrivial to use flash memory for checkpoint storage. First, even small flash memories are coarsely divided into segments, 512 bytes each on the MSP430. Each segment must be erased all at once, and erasing a segment requires energy comparable to filling the entire segment with data. Second, flash memories have a *one-way* property: once a bit is set to `0`, the only way to set it back to `1` is to erase the entire segment that contains it. Third, flash reads are nearly as fast as volatile RAM reads, but flash writes are two orders of magnitude slower (and correspondingly more energy intensive) than RAM writes. Fourth, many microcontrollers use flash memory for program storage, which limits the amount of nonvolatile storage available for other purposes.

[2] In Greek mythology, Sisyphus was the first king of Corinth and a conniving malefactor. His punishment in Tartarus was forever to repeat the task of rolling a boulder to the top of a hill only to have it roll back to the bottom.

3. Design of Mementos

The key observation motivating the design of Mementos is that it is difficult to predict the behavior of energy harvesting on a transiently powered RFID-scale computer. For example, devices that harvest energy from RFID readers are subject to fluctuations in voltage (Figure 1) that are highly dependent on the operating environment and the device's physical orientation. With the advent of *programmable*, *general-purpose* transiently powered computing comes a need for general-purpose power failure recovery mechanisms. Without general-purpose mechanisms, programs on these devices must either finish quickly—not always an option—or include potentially complicated application-specific logic to manage their own computational state. Mementos aims to remove both of these obstacles.

Mementos has two parts: a set of program transformation passes that insert energy-measurement code at control points in a program, and a compact library that provides state checkpointing and recovery functions. Mementos can be integrated into a project's build system via standard means (e.g. a Makefile). Following are Mementos's high-level design goals and guiding design principles. Given the constraints of RFID-scale devices, we consider the goals of minimizing overhead and maximizing efficiency to be self-evident. Section 5 evaluates Mementos against our design goal.

Goal: Split programs across multiple lifecycles. Mementos must, at run time, automatically suspend and resume programs without user intervention. This program splitting aims to expand the range of applications suitable for RFID-scale devices.

Principle #1: Run on existing hardware. Mementos requires no special hardware support other than the ability to measure the voltage of the platform's energy buffer. Circuitry for voltage measurement is common on computing devices that operate on finite energy supplies (e.g. batteries).

Principle #2: Reason minimally about energy at compile time, maximally at run time. Past work has demonstrated that even expert programmers cannot be trusted to reason correctly about energy [40]. Reasoning about run-time energy availability at compile time may be impossible because of inconsistent harvesting and limited computation available for prediction. Mementos estimates available energy at run time and inserts energy checks at compile time, obviating the need for complex logic to deal with changing energy conditions.

3.1 Compile-Time Instrumentation

Mementos modifies programs in two ways at compile time. First, it places *trigger points*—calls to a Mementos library function that estimates available energy—at control points in the program. Second, it wraps the program's `main()` function with code that restores execution from an available checkpoint.

To suspend execution in time for a checkpoint to complete, Mementos should insert enough energy measurements at compile time so that run-time energy trends are effectively sampled; however, it should not insert so many that measurement cost predominates over execution. To satisfy our goal of supporting a wider range of applications, Mementos must also be compatible with programs that are structured in different ways. To these ends, Mementos offers three different instrumentation strategies that enable it to instrument common structures—loops and function calls. In *loop-latch mode*, Mementos places a trigger point at each loop latch (the back-edge from the bottom to the top of a loop), resulting in an energy check for each iteration of each loop in the program. In *function-return mode*, Mementos places a trigger point after each call instruction, resulting in an energy check each time a function returns. In *timer-aided mode*, which is designed to reduce the frequency of energy-intensive checkpointing operations, Mementos adds to either the loop-latch or function-return mode a hardware timer interrupt that raises a flag at predetermined intervals. Each trigger point then checks the flag and proceeds with an energy check only if the flag is up. The flag is lowered again for the next trigger point.

Besides offering three strategies for automatic trigger-point placement, Mementos supports application-specific customization by providing a simple API. A programmer can opt not to run any of Mementos's instrumentation passes and instead insert trigger points manually, simply by including a header file and placing function calls in the program. She can similarly insert her own calls to Mementos functions to skip energy checks and force checkpointing.

3.2 Run-Time Energy Estimation

At run time, Mementos estimates the energy remaining in the device's energy buffer by measuring its voltage. Microcontrollers suitable for RFID-scale devices typically have on-chip analog-to-digital converters (ADCs) that sample voltage as a proxy for any number of environmental phenomena (e.g. temperature and physical orientation); Mementos simply makes use of this subsystem.

For an ideal capacitor, the amount of energy it presently contains (E) is determined by the capacitor's present voltage (V) and its fixed capacitance (C), via the following equation: $E = CV^2/2$. However, since calculating energy from voltage may require computationally intensive operations such as squaring or floating-point arithmetic, Mementos uses the ADC's voltage measurement directly when making checkpointing decisions: it compares the measured voltage to a *checkpoint threshold voltage* (V_{thresh}). Above this voltage, Mementos assumes that it does not need to write a state checkpoint. It interprets a voltage below the threshold as indicating that power failure is imminent and begins checkpointing state.

Ideally, program state should be saved at the last practicable opportunity before a power failure in order to minimize unsaved computation. However, unpredictable energy harvesting and the variations in the cost of saving checkpoints make perfect failure prediction infeasible.

Mementos predicts future power failures *conservatively* by assuming that no energy will be harvested between the trigger point and a power failure. Worst-case run times can be calculated as follows. The charge on a capacitor is $Q = CV$. Under a constant current draw I, the charge decreases as $\frac{dQ}{dt} = I$, and the time between two voltage levels V_{max} and V_{min} is $\Delta t = C(V_{\text{max}} - V_{\text{min}})/I$. If, for example, an MSP430 draws 238 μA in active mode, fails to write to flash below 2.2 V, and needs 17.5 ms to write a 200-byte checkpoint, Mementos should start checkpointing *at the latest* when supply voltage falls to 2.62 V. However, two factors complicate the task of checkpointing at the last moment: checkpoint sizes and times may vary at run time depending on stack depth; and Mementos's ability to precisely time a checkpoint depends on the frequency of trigger points. Section 4 describes the mechanisms that help a programmer choose a reasonable checkpoint threshold voltage above the lower bound.

Because it may suffer power loss during a checkpointing operation, Mementos exhibits defensive behavior that ensures correctness at a cost of time—i.e., its precautions err on the conservative side and may increase the amount of redundant computation during a complete execution. Mementos's first precaution is that it writes checkpoints *head first* and *tail last*: the first word of data it writes to nonvolatile memory contains enough length information for a complete checkpoint to be reconstructed and an incomplete checkpoint to be detected; the last word it writes is the magic number that ends every valid checkpoint. Second, if Mementos detects an incomplete checkpoint during recovery or next-checkpoint location, it refuses to write any more information to the containing segment of nonvolatile memory and marks the segment for deletion. Mementos erases such marked segments immediately after boot when available energy is guaranteed to be above a predefined threshold.

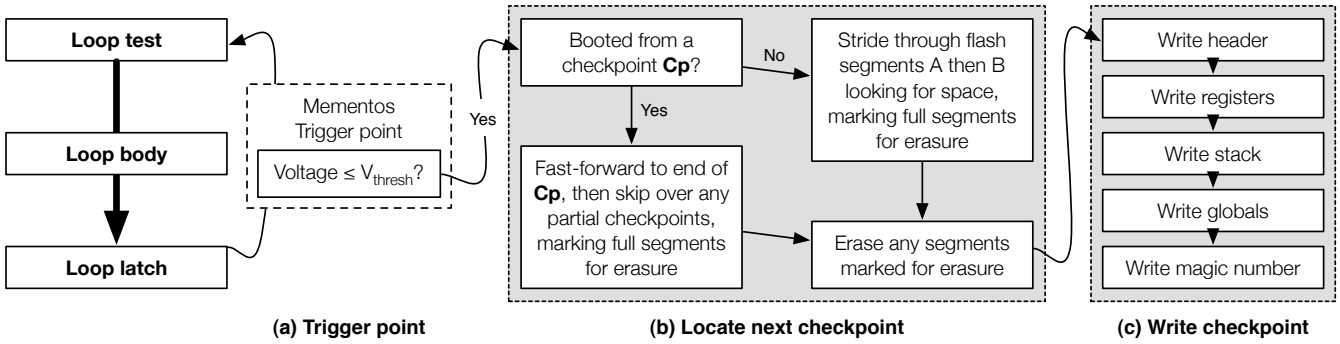

Figure 2. Overview of run-time checkpointing in Mementos. This diagram depicts the *loop-latch mode* in which Mementos instruments loop back-edges with energy checks that conditionally trigger checkpointing.

3.3 Run-Time Checkpointing

Mementos links trigger-point instrumented programs with a run-time checkpointing library. When a trigger point initiates a checkpoint, Mementos copies relevant program state to nonvolatile memory along with meta-information (Figure 2). After a power failure, Mementos searches for a restorable checkpoint and, if it finds one, copies the stored state into RAM and resumes execution.

Checkpointing on RFID-scale devices is more difficult than checkpointing on more powerful platforms. With no operating system, Mementos must be linked into a deployed program. Mementos shares all of the program's resources and must perform *in-place* checkpointing to capture the state of the program as it was immediately before entering the trigger point. Additionally, the limitations of flash memory (discussed in Section 2 and below) complicate checkpoint management.

In-place checkpointing. Most checkpointing systems are designed to run on multiprogrammed operating systems (e.g. the MPI checkpointing of Bronevetsky et al. [3]) or in hardware environments that support the issuance of commands by other devices (e.g. the sensornet checkpointing of Österlind et al. [30]). Mementos runs on RFID-scale devices without resources to run conventional operating systems and may have no electrical connection to their environs. It interacts with its host program via function calls and shares the program's address space, stack, registers, and globals.

Flash writes are slow and energy intensive relative to volatile memory writes, so instead of blindly copying the entire contents of RAM in each checkpoint, Mementos captures only the regions of RAM that are in use at the time the trigger point is called. These comprise the stack, whose depth can be calculated via the stack pointer; the global variables, captured by Mementos in an analysis pass at compile time; and the register file, which includes the stack pointer, program counter, and a status register. Mementos does not capture the program's executable code because this code is typically already stored in nonvolatile memory.

Keeping with a convention generally followed by programs targeting memory-limited mote-class devices, none of our test cases allocates memory dynamically. This allows our implementation of Mementos to sidestep the problem of checkpointing a fragmented heap by not checkpointing the heap at all; we note instead that the feasibility of efficient heap checkpointing depends on the quality of the dynamic memory allocator and its internal state at the time of checkpointing. `setjmp` and `longjmp` operations are unsupported for similar reasons; exception-style control flow is not typically used in embedded software. Notably, the Embedded C++ standard lacks support for exceptions [34]. However, Mementos *is* compatible with interrupt service routines (ISRs) that behave like normal function calls, as they do on MSP430. Finally, Mementos does not currently provide special support for reentrant code or threading libraries; its checkpointing operation is non-reentrant.

At checkpoint time, Mementos first pushes all of the registers onto the stack; registers tend to change during program execution. (On the MSP430, the register file includes the status register `R2`; Mementos takes minor precautions to avoid affecting this register's value before saving it.) It stores the value of the stack pointer, adjusted for the function call that initiated the checkpoint, and sets the stored value of the program counter to the return address from the checkpoint function's own stack frame. It then finds space for a new checkpoint (details below), and writes at the beginning of the free space a checkpoint size header that includes the adjusted stack depth. Mementos then writes the saved registers, stack, and globals to flash. Finally, it writes a magic number to indicate the end of the checkpoint. The location of the magic number is trivial to calculate from the size header, allowing Mementos to detect incomplete checkpoints that are due to power failures during checkpointing.

At boot, Mementos searches for an active checkpoint (details below), then copies its contents into RAM. As when checkpointing, it must copy carefully so that it restores the saved state rather than a mixture of the saved state and its own state. For example, on the MSP430 architecture, it restores the register file in descending numeric order, leaving the stack pointer (`R1`) and the program counter (`R0`) for last. Restoring the program counter from the checkpoint implicitly transfers control to the program where it left off.

Idempotent actions. Some code cannot be re-executed safely. For example, an RFID-scale device may contain an actuator that toggles a property of another device. To enable programmers to work with non-idempotent code, Mementos allows programmers to selectively disable instrumentation on a per-loop or per-function basis. Appending the token `_mnotp` (mnemonic: "Mementos, no trigger points!") to any function's name causes Mementos to skip the function, i.e., not instrument its loops or a return from it. A programmer can disable instrumentation even for inline functions, which means she can direct Mementos to ignore any piece of code she wishes. We implemented an optional additional pass that emits compile-time warnings upon encountering possibly non-idempotent actions such as volatile writes—e.g. a write to a memory address that is mapped to a hardware output pin. The warning messages suggest points at which the programmer might profitably disable checkpointing.

Checkpoint management. Unlike prior systems that rely on OS facilities to simply dump process memory to a file system (e.g. *libckpt* [33]), Mementos must manage its own checkpoint storage. Its strategy approximates a circular buffer, but the characteristics of flash memory require special care.

Mementos is designed to facilitate the execution of programs from beginning to end; as a result, once a checkpoint is success-

fully written to nonvolatile memory, all previous checkpoints are *superseded*. Mementos maintains at most one *active* checkpoint at any given time. At boot or when searching for free space, Mementos uses a simple active-checkpoint search algorithm: it walks a reserved region of flash memory, skipping over sequentially stored valid checkpoints (those that end with correct magic numbers) and stopping when it discovers a valid checkpoint that is followed by a byte in the erase state (`0xFF` for flash).

Flash is erasable only segment-by-segment. To erase old or invalid checkpoints without destroying active checkpoints, Mementos reserves two segments of flash memory to checkpoint storage. When a checkpoint is completely written to one of these segments, it supersedes all checkpoints stored in the other, and Mementos marks the other segment erasable by zeroing its first word—an operation that can be reversed in flash only by erasing a whole segment. Mementos erases segments marked erasable at two times: at boot, when energy is likely to be plentiful, and when it cannot find space for a new checkpoint in either segment (i.e., when one segment is marked for erasure and the other is full of checkpoints).

Energy polling versus interrupts. A natural question is why Mementos polls the hardware energy supply instead of waiting for an interrupt to occur when voltage falls below a threshold. Voltage supervisors are common circuit components, but most—crucially, including existing prototype RFID-scale devices—do not feature an *adjustable* threshold voltage. Mementos is designed to work on existing devices without design modifications, so it polls for supply voltage. However, if such a device featured an adjustable or multi-level voltage supervisor, Mementos could avoid polling and simply associate itself with the appropriate interrupt(s). The checkpointing routine is designed to be called as a subroutine and works equally well as an interrupt handler (which is how the *checkpointing oracle* described in Section 5 works).

4. Implementation

Mementos formulates its program transformations as LLVM [21] optimization passes. These passes operate on intermediate LLVM bitcode before LLVM's MSP430 backend generates target-specific assembly. They are implemented in C++ and comprise a total of 601 lines of code including whitespace, comments and header files. Mementos's run-time library comprises an additional 761 lines of C and inline MSP430 assembly.

We provide a build harness that instruments an existing C program with Mementos and builds multiple MSP430 ELF executables per input program: one version for each of Mementos's three instrumentation strategies and an uninstrumented version for comparison. A script repeatedly calls the build harness with different parameters, varying the checkpointing voltage threshold V_{thresh} and, when applicable, the timer interval used for timer-aided checkpointing. Finally, another script allows a programmer to compare the performance of all the variants in a simulator. Section 5 details our use of this simulator for evaluation.

Implementation tradeoffs. Instrumenting programs at the level of LLVM's intermediate representation allows Mementos to use simple, target-agnostic transformations, but it limits Mementos's visibility into later compilation stages. In particular, Mementos's LLVM passes do not have access to the results of code generation, so they cannot, for example, leverage empirically measured instruction-energy estimates [36]. We originally considered static analysis to count post–code-generation instructions and simply tag basic blocks with energy estimates, but that simple approach proved inadequate because of the complexity of calculating at run time the amount of work remaining until program completion; such calculations necessarily occur at trigger points, which may be frequent.

5. Evaluation

This section evaluates Mementos's ability to correctly and efficiently preserve computational state across frequent power failures. To simulate the energy conditions a deployed device might face, we feed voltage traces from a hardware prototype's analog energy-harvesting frontend into a trace-driven, cycle-accurate simulation of an RFID-scale device. We consider three distinct workloads that exercise different computational resources found on prototype RFID-scale devices: computation, sensing, and storage. We evaluate executions of these workloads with each of Mementos's instrumentation strategies and compare the results to baseline measurements taken against predictable energy conditions and uninstrumented programs. Finally, we offer the results of running Mementos on hardware instances of the model we simulate.

5.1 Mementos in Simulation

Mementos is designed with RFID-scale devices in mind, so we developed a flexible, trace-driven testbed featuring a simulated microcontroller (MCU) and energy supply modeling those found on a hardware device (a prototype WISP [39], revision 4.1). While the WISP supports interactive debugging via a standard JTAG interface, the simulator adds several key features: the ability to perform repeatable experiments against recorded traces of RF energy harvesting; the ability to vary hardware parameters (e.g. available memory) to overcome limitations of the prototype device; and the ability to obtain exact profiling information such as cycle counts. We therefore primarily present results obtained in simulation.

We augmented MSPsim [14], a cycle-accurate MSP430 simulator that accepts MSP430 ELF binaries, with a simulated capacitor of our design that obeys the basic capacitor equations for charging and discharging.[3] The simulated capacitor halts execution whenever its voltage falls below the MCU's nominal minimum operating level and resets the MCU when the voltage returns to an operable level after a power failure. We also added to MSPsim a notion of electrical current, which governs the speed at which a capacitor's energy is depleted, and associated each of the simulated MCU's operating modes (active mode, flash write, analog-to-digital conversion, and five low-power modes) with current values measured from a hardware WISP. We made other minor changes to MSPsim to simulate power failures (e.g. preserving nonvolatile memory contents across resets).

Our simulator optionally accepts a voltage trace that governs energy availability over time. On a physical RF-harvesting device that buffers energy in a capacitor, such as a WISP, the capacitor's voltage increases when the analog frontend gathers energy from an oscillating radio wave and sends charge through a diode; the voltage decreases as a factor of both time (via leakage) and current draw (via circuit usage). To capture voltage traces from real hardware, we isolated a WISP's RF-harvesting analog frontend, attached it to a ~ 10 KΩ resistor that approximated the electrical load of the WISP's microcontroller during active computation, and recorded ten traces of voltage across the resistor (sequences of *time, voltage* pairs) corresponding to ten different patterns of motion near an RFID reader. Figure 1 shows several of these traces.

As for the simulated capacitor, we tested it under the simulator's suite of simulated electrical currents (drawn from our measurements of a real MSP430 in its various modes) and confirmed that its decay time under each regime was accurate to within 5 ms. Figure 3 compares the voltage trends exhibited by both a hardware WISP and our simulated WISP after each was charged to 4.5 V and allowed to discharge to 2.2 V (the voltage threshold for flash writes) while executing an infinite loop in *active* mode.

[3] We refer the reader to Horowitz and Hill [18] for a detailed discussion of capacitor behavior.

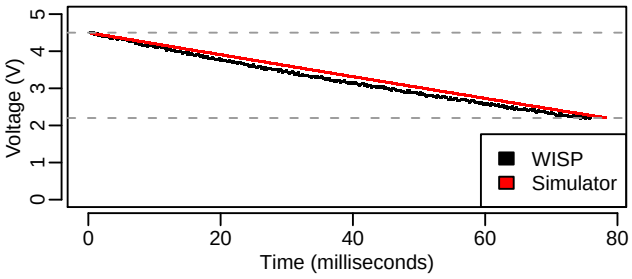

Figure 3. Simulated capacitor's voltage approximates the discharge time and voltage drop of a hardware WISP's capacitor. Both were charged to 4.5 V and allowed to discharge while executing an infinite loop at 1 MHz in active mode. Both traces end at 2.2 V, the nominal minimum voltage for flash writes on an MSP430.

The WISP and our simulator are instantiations of the same model: both are transiently powered computational platforms. We tuned the simulator according to empirical measurements of WISP devices, including current consumption in every power mode and the timing and current consumption of flash and ADC operations. However, the only properties of the simulator that bear directly on our design goal of spreading program execution across multiple lifecycles are: (1) our simulations of the hardware peripherals Mementos employs—ADC and flash memory—must incur realistic time and energy costs (Section 5.1.2), and (2) the time and energy available between Mementos's checkpointing threshold V_{thresh} and the power cutoff threshold must be realistic (Section 5.1.3).

WISP property	Simulator mechanism
MSP430 MCU	MSPsim fully supports MSP430 ISA
RF harvesting	Sim. accepts voltage trace recorded on hardware WISP's analog frontend
Low-power modes	Sim. obeys low-power setting, changes simulated MCU's current consumption
Electrical current	Tracks *(power mode, voltage)* $\rightarrow$ *current* mapping measured empirically
Dynamic power	Sim. capacitor obeys capacitor equation for discharging under load
Quiescent power	Sim. capacitor obeys exponential decay equation
Flash, ADC	Simulated current tracks empirical time and current measurements
Radio	(Not simulated)

Table 1. Mapping of WISP hardware properties to simulator mechanisms.

Table 1 maps simulation mechanisms to the hardware features they simulate. The simulator does not simulate the cost of using radio for two reasons: (1) the device being simulated has no active radio, instead using backscatter to reflect radio waves; and (2) backscatter modulation operates on the same radio waves that provide energy, making communication effectively free on backscattering devices. The simulator also makes no attempt to account for environmental parameters such as temperature because the variations they induce are typically small.

5.1.1 Test cases

Our evaluation of Mementos considers three test cases representing common tasks for low-power embedded systems.

The `rsa64` test case uses iterative left-to-right modular exponentiation of multiple-precision integers to encrypt a 64-bit message under a 64-bit public key and 17-bit exponent. (Larger sizes cause the computation to exceed the RAM capacity of a WISP.) The program's data segment comprises 124 bytes of globals. Mementos instruments 24 loop latches in loop-latch mode and 51 call sites in function-return mode, in both cases primarily inside the multiple-precision integer library underlying the RSA implementation.

The `sense` test case takes 64 consecutive ADC samples of a simulated accelerometer and computes the minimum, maximum, mean, and standard deviation of the samples, then stores these statistics to nonvolatile memory. Such computations are common in sensing applications that sample environmental phenomena. The program stores raw sensor readings in a 128-byte global in RAM. Mementos instruments three loop latches in loop-latch mode and four call sites in function-return mode.

The `crc` test case, drawn directly from WISP firmware, computes a CRC16–CCITT checksum over a 2 KB region of on-chip flash memory. The WISP firmware computes CRC checksums to send along with responses to an RFID reader. CRC also provides a mechanism for data-integrity checks; we imagine such a check being important for future in-place firmware updates on RFID-scale devices. The program comprises a tight CRC nested loop and an outer loop that repeatedly calls the CRC function; Mementos instruments three loop latches in loop-latch mode and two call sites in function-return mode. Because the CRC loop is tight, we use the `crc` test case to evaluate the `_mnotp` mechanism for selectively disabling checkpoints (Section 3), reducing the number of loop latches instrumented to one.

A testbed provided with Mementos compiles each test case against a variety of Mementos configurations, varying by instrumentation strategy and static checkpoint threshold voltage V_{thresh}. It runs each variant against two modes of the simulator: trace-driven mode and decay-only mode. In trace-driven mode, a voltage trace from real hardware governs simulated energy availability. In decay-only mode, the simulated capacitor's voltage begins at a fixed value and strictly decreases with time and use; it starts at the same voltage in subsequent lifecycles. In both modes, trigger points induce checkpointing below V_{thresh} and simulated power loss occurs at the platform's nominal minimum voltage for flash writes (2.2 V). Finally, the simulator collects baseline measurements by running each variant without instrumentation, then without energy constraints, and finally in an "oracle" mode that predicts power losses and checkpoints at the last practicable opportunity (Section 5.1.2).

5.1.2 Baselines and Metrics for Comparison

Programs instrumented with Mementos differ in several key ways from their uninstrumented counterparts. First, they include additional code that provides checkpointing and restoration mechanisms. This extra code endows programs with robustness properties at a cost of storage space and available cycles at run time. Second, they can execute over multiple device lifecycles instead of restarting from the beginning with each failure. Third, they exhibit different behavior under different energy conditions; checkpoint sizes vary with stack size, and the point at which checkpointing begins varies with supply voltage. With these differences in mind, we adopt the metrics shown in Table 2.

A natural question is *what is the smallest number of CPU cycles in which a program can complete?* To determine the minimum number of CPU cycles required to execute each test case variant, we consider a scenario in which the simulated capacitor's voltage is held in the CPU's normal operating range. Under these circumstances, Mementos's voltage checks never trigger checkpointing and the program completes in a single lifecycle. It is easy to see that, for a given program, its run time against unlimited energy is a lower bound on its run time against any energy scenario. Table 3 gives results for all three test cases. For the instrumented variants,

Metric	Description
Lifecycles	The number of reboots (including initial boot) required to complete the program
Total cycle count	The number of CPU cycles required to complete the program over all lifecycles
Mementos cycles	Percentage of total cycle count spent in Mementos code
Waste	Percentage of total cycles that occurred between the last checkpoint in a lifecycle and the subsequent power loss

Table 2. Metrics for evaluation.

the program spends a nonzero number of cycles executing energy checks at trigger points. The differences in percentage of cycles spent in Mementos are due to the prevalence of different control-flow structures in the test cases.

Instr. Type	`crc` / %	`sense` / %	`rsa64` / %
Uninstr.	575,315 / —	157,635 / —	70,218 / —
Loop latches	619,450 / 6.9	284,795 / 44.3	303,250 / 76.2
Fn. returns	577,702 / 0.2	201,151 / 21.9	214,177 / 66.5
Timer+latches	598,171 / 3.4	166,375 / 4.5	78,914 / 8.6

Table 3. Cycle counts (and percentage of cycles spent in Mementos code) for three Mementos test cases under an *unlimited-energy* scenario, i.e., voltage always above V_{thresh}. This table illustrates the base cost of Mementos's energy checks at run time.

Versus an Energy Oracle. Another natural question is *what is the minimum number of lifecycles over which a program's execution can spread?* If energy is scarce, lifecycles may occur infrequently and the difference between k and $k+1$ lifecycles may be vast. To establish a baseline for evaluation, we implemented an *oracle mode* for the simulator. In oracle mode, the simulator accepts an *uninstrumented* program that has been linked against Mementos. It monitors the simulated capacitor's voltage during the program's execution and, for each power lifecycle, initiates checkpointing at the last practicable opportunity—i.e., when allowing the voltage to fall any farther would result in an incomplete checkpoint. Given the difficulty of predicting power loss events at run time (Section 3), the simulator uses a binary search strategy to adjust its notion of "last practicable" over repeated executions of each lifecycle.

Because programs executed in oracle mode are uninstrumented, they contain no trigger points and no automatically inserted calls to the checkpointing function. The build process splices Mementos's restoration code in front of the program's original `main()` function to enable boot-time restoration from saved checkpoints.

The oracle mode's main benefit is establishing a lower bound on the number of lifecycles and CPU cycles a program needs to complete under a given energy condition, e.g. a trace in the simulator. An ancillary benefit is that the oracle mode guides the implementer in selecting a fixed voltage threshold V_{thresh}. As the simulator runs in oracle mode, it reports the last practicable threshold voltage discovered for each lifecycle. If the implementer can provide a representative voltage trace to the simulator, then choosing a V_{thresh} suitable for the application is a matter of observing the oracle mode's reported cutoff voltages and choosing a slightly higher value.

Table 4 shows results obtained in oracle mode under *variable voltage* (rather than the fixed high voltage of Table 3). The cycle counts and lifecycle counts in the table should be considered lower bounds on those metrics for instrumented versions.

Fixed overhead. Mementos adds space overhead in two ways: by increasing code size and by reserving two flash segments (1 KB total on the MSP430) for checkpoint storage. Without compiler optimizations for code size, Mementos increases executable size by a constant amount (just under 2.4 KB) plus 4 bytes per trigger point. While a 2.4 KB increase in code size accounts for 30% of the code memory on our prototype, the sibling chips used in newer RFID-scale devices have much more code space (up to 116 KB) and nearly identical energy characteristics.

	`crc`	`sense`	`rsa64`
Decay-only	621,501 (8)	197,215 (3)	70,886 (1)
Trace #1	685,555 (15)	308,986 (7)	179,626 (4)
#2	685,150 (15)	304,678 (6)	158,187 (3)
#3	685,801 (15)	308,724 (7)	219,970 (5)
#4	685,641 (15)	288,063 (6)	157,438 (3)
#5	685,096 (15)	840,594 (16)	153,181 (3)
#6	685,099 (15)	287,876 (6)	139,211 (3)
#7	683,884 (15)	287,573 (6)	180,195 (4)
#8	685,005 (15)	287,741 (6)	139,840 (3)
#9	685,608 (15)	287,422 (6)	157,613 (3)
#10	685,045 (15)	287,927 (6)	158,805 (3)

Table 4. Oracle-mode lower bounds [CPU cycles (lifecycles)] for three test cases against ten voltage traces and decay-only mode. For the `crc` test case, the mean proportion of cycles spent in Mementos code was $24.5 \pm 1.3\%$; for `sense`, $49.1 \pm 11.0\%$; for `rsa64`, $56.6 \pm 5.6\%$.

We used an oscilloscope to precisely time flash and ADC operations on a WISP prototype, then confirmed that the simulator matched the measured values exactly. Each trigger point that uses the ADC to check voltage consumes 647 clock cycles (647 μs at 1 MHz). Checkpoint operations consume 105 μs per 16-bit word written, plus the aforementioned ADC cost for the trigger point, plus several hundred cycles for bookkeeping (depending on the number of currently stored checkpoints). When Mementos must erase a flash segment, the erasure takes $13,062$ μs. Matching these timings with corresponding electrical current measurements gives us confidence that the simulator's behavior is accurate.

5.1.3 Performance and Overhead

Via experiments on our test cases in the simulator and on WISP hardware, we find that Mementos satisfies the design goal of splitting program execution across multiple lifecycles with intervening power losses.

For a given voltage trace and test case, the performance of Mementos along all of our metrics depends on the compile-time choice of voltage threshold V_{thresh}. There are natural bounds on practicable values for this variable. From above, V_{thresh} is practically bounded by the wakeup threshold V_{w} at which the platform boots after a power failure. If $V_{\text{thresh}} \geq V_{\text{w}}$, Mementos will in the worst case begin checkpointing the first time a trigger point is encountered. From below, V_{thresh} is strictly bounded by the minimum flash-write voltage, but a higher application-specific threshold effectively lower-bounds V_{thresh} because incomplete checkpoints are not restorable.

For an example of how the above bounds apply, consider the `crc` test case and a particular voltage trace (#9). Uninstrumented, the test case fails to complete against the trace because it cannot sustain computation for long enough. According to the simulator's oracle mode (Table 4), the minimum number of lifecycles required to run the test case to completion against trace #9 is 15. With loop latches instrumented, the test case runs to completion in 17 or more lifecycles depending on the fixed voltage value V_{thresh}. With function-return instrumentation, the test case fails to complete against the trace: the computation's work loop does not use function calls to transfer control flow, so there are no function returns to instrument until the main loop completes. With timer-controlled loop-latch instrumentation, the test case fails to complete against

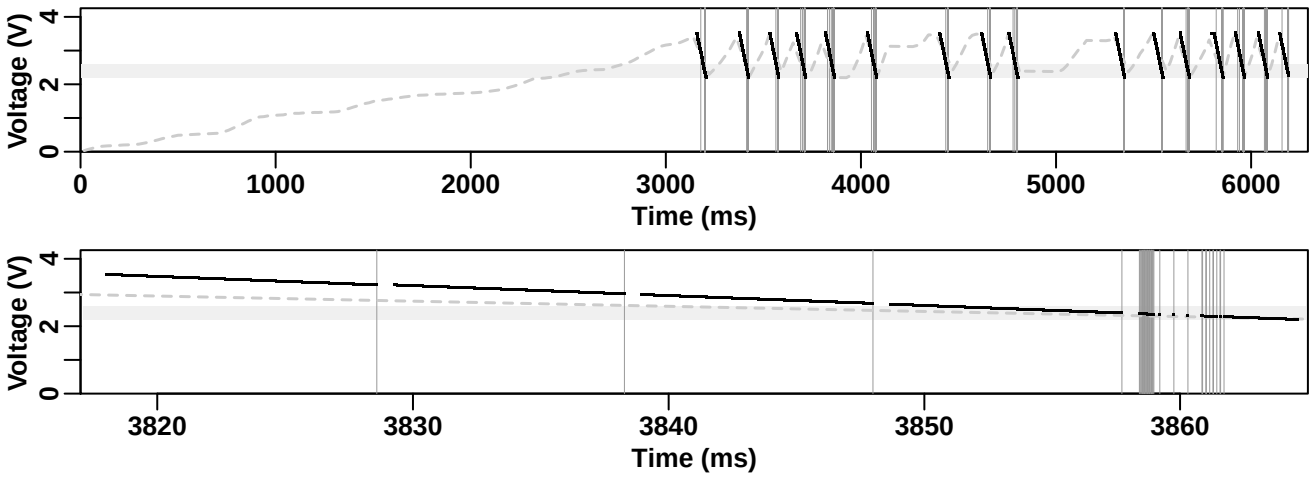

Figure 4. Simulated voltage versus time as Mementos spreads the execution of the `crc` test case across 17 power lifecycles (16 resets) against a voltage trace (#9). The bottom plot highlights a single power lifecycle from the top plot.

the trace because of infelicitous timing (the voltage when the timer fired was too low for a subsequent checkpoint to complete).

Figure 4 depicts a complete execution of the `crc` test case against the same voltage trace (#9) in the simulator. The simulated capacitor charges (dotted gray line) when the input voltage increases and discharges (solid black line) during computation and storage. When capacitor voltage falls into the shaded region between V_{thresh} (set to 2.6 V at compile time) and the CPU's minimum voltage for flash writes (2.2 V), Mementos checkpoints. Mementos uses the CPU (vertical lines) to check energy, find space for checkpoints, collect state, and write state to flash. Gaps in the trace are due to waiting for hardware peripherals (flash and ADC).

Table 5 shows the relationships among V_{thresh}, Mementos's share of CPU cycles, and waste for the `sense` test case instrumented in each of Mementos's modes. For brevity, we delve into detail for only a single test case and only two energy conditions. The top half of the table gives results for *decay-only* mode, in which the capacitor's energy is set to a fixed level and no more energy is delivered until the next lifecycle. The bottom half of the table gives results for a voltage trace (#9) that elicits representative behavior. Without Mementos instrumentation and given unlimited energy, the `sense` test case requires $157,635$ CPU cycles to complete. However, when run against any of our voltage traces, the uninstrumented program cannot complete because it never receives enough energy to run for long enough; this uninstrumented program satisfies our definition of a *Sisyphean task*.

Mementos spreads the `sense` task across multiple lifecycles, but it increases the total number of CPU cycles needed for program completion. In oracle mode, the simulator reports that, if Mementos checkpoints at the last practicable opportunity in each lifecycle, the program can complete in 3 lifecycles and $197,215$ CPU cycles, an increase of 25.1% over the uninstrumented program. Varying the compile-time V_{thresh} and (in timer-aided mode) the timer interval, we find that Mementos adds between 65.4% and 360.0% CPU cycles over the uninstrumented program. We infer from Table 5 that V_{thresh} is effectively lower-bounded by 2.6 V; the oracle-mode log confirms that the oracle began its last-minute checkpointing at 2.61 and 2.64 V in the two lifecycles before program completion. The evident difference between loop-latch and function-return modes in this scenario is that, while loop-latch mode can accommodate a lower V_{thresh} with its more frequent trigger points on this loop-structured program, function-return mode requires fewer CPU cycles and lifecycles when it is applicable. We also see that timer mode, in which trigger points at loop latches are activated only when a timer interrupt raises a flag, offers a tradeoff: a program may require less time to complete, but a suboptimal value for the timer interval can introduce unexpected timing-related failures.

Against a specific voltage trace (#9), oracle mode reports that the `sense` test case requires at least 6 lifecycles and $287,422$ CPU cycles; the last-minute checkpointing voltage thresholds it found were between 2.58 V and 2.62 V. Table 5 shows the effects of automatically varying the instrumentation mode and timer interval over a series of simulated runs. It shows that Mementos adds at least 64.7% CPU cycles over the uninstrumented program. Table 5 also illustrates a hazard of choosing a fixed V_{thresh} value too close to upper bound of the oracle's reported last-minute voltage threshold choices: at $V_{\text{thresh}} = 2.6$ V, the latch-instrumented program makes progress only when it runs against certain felicitous portions of the voltage trace; none of the other instrumentation modes allow the program to complete. At $V_{\text{thresh}} = 2.8$ V, the program completes in all tested instrumentation modes, and it becomes evident that the program's structure lends itself to function-return instrumentation as well as loop-latch instrumentation. At $V_{\text{thresh}} = 3.0$ V, the program's run time increases because Mementos begins checkpointing earlier in any given lifecycle, and the timer-aided latch-instrumented program begins to evince an advantage over the plain latch-instrumented program because of less-frequent checkpointing. At higher V_{thresh} values, the non-timer-aided modes fail to complete in fewer than 500 lifecycles because the time between checkpoint restoration and subsequent checkpointing leaves little time for computational progress. A reasonable interpretation of Table 5 is that a fixed value of 3.0 V for V_{thresh} is appropriate for the `sense` test case if energy conditions are not known in advance; in this case a programmer can reasonably expect the Mementos-instrumented computation to complete within twice the number of lifecycles that the oracle reports.

5.2 Mementos on Hardware

We have tested Mementos on a WISP (revision 4.1) prototype like those described in Section 4. Running Mementos on this prototype device required only one small hardware change: we substituted a readily available 2.8 V voltage regulator for the supplied 1.8 V regulator in order to meet the nominal minimum voltage requirement to write to the MSP430's on-chip flash memory. (Other prototype

V_{thresh}	**Loop-latch mode** *lifecycles (cycles / %waste / %M)*	**Function-return mode**	**Timer-aided mode interval (μs)** 20,000	40,000
2.4 V	—	—	—	—
2.6	—	—	—	—
2.8	7 (502,576 / 6.8 / 63.9)	—	4 (298,890 / 52.8 / 21.2)	—
3.0	8 (586,788 / 12.6 / 68.6)	5 (368,255 / 13.0 / 52.9)	4 (298,890 / 52.8 / 21.2)	—
3.2	9 (671,849 / 7.2 / 72.1)	6 (429,754 / 2.6 / 58.4)	4 (260,796 / 2.6 / 34.5)	—
3.4	9 (725,214 / 15.9 / 73.3)	6 (453,827 / 16.5 / 59.8)	5 (361,471 / 4.6 / 49.1)	4 (314,181 / 20.4 / 27.8)
2.4	—	—	—	—
2.6	89 (5,235,399 / 89.4 / 64.2)	—	—	—
2.8	13 (708,641 / 10.5 / 71.3)	11 (557,164 / 10.8 / 63.7)	53 (2,779,635 / 70.7 / 32.8)	22 (1,044,242 / 70.8 / 29.5)
3	46 (2,852,111 / 20.4 / 81.6)	16 (843,093 / 22.6 / 73.4)	10 (473,396 / 13.8 / 55.1)	31 (1,421,150 / 77.9 / 29.7)
3.2	—	—	10 (489,739 / 13.5 / 55.3)	15 (702,151 / 56.6 / 29.4)
3.4	—	—	—	22 (1,194,221 / 30.0 / 58.3)

Table 5. In decay-only mode (top half) and against a voltage trace (#9, bottom half), the `sense` test case exhibits behavior that is dependent on the voltage threshold $V_{\texttt{thresh}}$ and, in timer-aided mode, the timer interval. *%M* refers to the portion of CPU cycles spent within Mementos code. This table illustrates the key differences among Mementos's various instrumentation modes.

RFID-scale devices, including others in the WISP family, regulate the chip's voltage at the required level, so we expect this substitution to be unnecessary except in the case of this prototype.)

We added to Mementos a simple hardware signaling mechanism using several of the WISP's general-purpose I/O pins; toggling these pins at run time allowed us to observe Mementos's basic activity at run time. We observe via the signaling mechanism that Mementos successfully completes simple computations that span multiple lifecycles, but we omit results comparing hardware runs of our test cases to simulator runs because the poorly suited 2.8 V regulator dominates the platform's energy consumption—its high static power consumption roughly halves the lifecycle duration.

5.3 Improvements

We suggest—but do not evaluate, for brevity—several improvements to Mementos that may improve its performance. Code for these improvements is available with the Mementos distribution unless noted otherwise.

Techniques to reduce trigger point frequency. As we observed above for the `crc` test case, Mementos's loop latch instrumentation can result in excessively frequent trigger points when applied to loops with small bodies and large trip counts. Detecting small loop bodies and large trip counts, whether via static analysis or profiling or a combination, may prove useful toward reducing Mementos's share of CPU cycles.

Compression. Reducing checkpoint sizes has been a concern for previous checkpointing systems; past approaches have included memory exclusion [32] and straightforward file compression via external programs. On an RFID-scale device with flash memory that is expensive to write and erase, Mementos should minimize checkpoint sizes to minimize the cost of writing them (and the amortized cost of erasing them). However, Mementos is designed to run without an operating system or file system, and we found that most implementations of well-known compression algorithms were too large to fit in our devices' limited code space.

Because many programs do not use all available registers, one promising but not fully implemented scheme compresses the register file by using a 16-bit bitmask to indicate which of the CPU's 16 registers are zero valued. During checkpointing, Mementos walks the register file, builds the bitmap, and avoids storing any registers that are zero valued.

We have also considered compressing full checkpoints instead of just the register file; all of the options predictably trade checkpoint size for run time. Our simulator saves checkpoints to files as it validates them, so we used checkpoint files as inputs to compression algorithms running in a separate MSP430 simulator. We implemented a reduced variant of the WK compression algorithm [19] but found that, while it reduced checkpoint sizes by an average of 55% for the `crc` example, it required 3.5 times as many CPU cycles as it would have taken to write the full checkpoint to flash. We implemented a variant of the popular LZ compression algorithm and found that it reduced checkpoint sizes 30% more than WK but was 18 times slower than simply writing the checkpoint to flash.

A third type of compression, not yet implemented, is incremental compression of checkpoints. We have not yet implemented incremental compression because of the complexity of computing incremental updates in a small memory footprint at run time, but the idea is promising—often the changes between two successive loop iterations, for example, are small. Additional compile-time analysis combined with per-trigger-point adjustments might make incremental checkpointing feasible in a future version of Mementos.

Run-time adaptations. As the rest of Section 5 illustrates in detail, compile-time tuning of Mementos's parameters can significantly change its behavior. We have designed several schemes by which Mementos could adapt its behavior at run time based on its measurement of key metrics. For example, to avoid executing time- and energy-intensive flash erasures at the beginning of lifecycles, Mementos could decrease the frequency of failed checkpoints by including in each checkpoint header the current value of the checkpoint threshold voltage. If Mementos notices an aborted checkpoint, it can adjust the checkpoint threshold voltage as appropriate. A similar technique might enable Mementos to gradually minimize the amount of wasted work at run time to approach the results of oracle-mode simulations.

We have not designed an energy prediction model for Mementos's run-time system because we assume that such a scheme would be prohibitively time intensive. Relaxing some of our assumptions about unpredictability might lead us to develop lightweight prediction schemes—integer versions of first- and second-order voltage trend approximations, for example—that could allow Mementos to avoid checkpointing if it believes power failure is not imminent.

Sleeping when appropriate. Most microcontrollers have RAM-retention modes that retain processor state and the contents of volatile memory. Such modes typically require two orders of magnitude less electrical current than active-mode computation, which slows—but does not stop—capacitor drain. We designed Mementos to be useful when energy delivery is arbitrarily sporadic. Some energy-harvesting mechanisms, such as solar panels, exhibit sudden or prolonged periods of harvesting nothing; in this case, Mementos's strategy of checkpointing to nonvolatile memory would

be more suitable than simply entering RAM-retention mode. However, we suspect that a hybrid approach incorporating both RAM retention and nonvolatile checkpoints would be a fruitful avenue for improvements to Mementos.

6. Discussion and Future Work

In this section, we discuss some alternatives to Mementos that an application developer might consider. We then suggest some future extensions to Mementos.

6.1 Alternative approaches

As discussed previously, traditional RFID-scale devices provide system designers with three options: use only trivially simple programs, require users to provide adequate power, or add state-saving logic to application code. Mementos uses automatic checkpointing to expand the use of RFID-scale devices beyond simplistic computation—without placing additional requirements on the user or the programmer. In addition to checkpointing, we have also considered other approaches to attain these goals.

Applications permitting, computations might instead be shortened using profiling and quality-of-service (QoS) information [1, 28]. These techniques make sense only for applications that tolerate lossy or noisy results, and would require more accurate predictions of power failures. To remain general, Mementos does not include these program transformations, though they are trivially compatible provided that Mementos's instrumentation points are preserved.

Another approach would perform long-running computation on a more powerful device (e.g. a Linux-based RFID reader), rather than the more constrained RFID-scale device. While this removes many impediments, it imposes others. To save power, prototype RFID-scale devices use low-throughput radio mechanisms, including software-controlled backscatter, that limit radio throughput. In security-sensitive applications, outsourcing cryptographic operations may violate security requirements, and existing techniques for outsourcing computation to untrusted infrastructure [15] still require a nontrivial amount of work on the client.

Finally, some CPU-intensive computations such as cryptography can be executed on dedicated peripheral hardware instead of a general-purpose microcontroller. Hardware acceleration allows some operations to complete more quickly but removes the flexibility afforded by reprogrammable microcontrollers. To remain general, Mementos does not assume that such peripherals are available.

6.2 Future Hardware

Moore's Law does not have an analogue for batteries [31], and increasing energy storage will likely continue to add significant bulk and weight. Since larger energy storage also makes devices less responsive—they take more time to charge—we expect RFID-scale devices to continue to have small energy buffers akin to today's capacitors.

While Mementos is currently implemented to use widely available flash memory, other types of nonvolatile memory may be better suited to frequent checkpointing in the future. Flash cells can tolerate 10,000 to 1 million erasures before becoming unusable. Information coding schemes that allow rewrites without erasures [4] might extend a system's lifetime; however, alternatives like phase-change memory (PCM), magneto-resistive RAM (MRAM) and ferroelectic RAM (FeRAM) all promise fewer complications. Our initial experiments with EEPROM storage on a prototype WISP indicate that its energy characteristics are similar to flash memory's, but EEPROM reads are significantly slower. Still, EEPROM or other auxiliary storage may be useful for storing small pieces of metadata without necessitating coarse-grained erasures.

6.3 Future Work

A key opportunity for improvement lies in making Mementos more adaptive. Each binary linked against Mementos is currently subject to a single set of checkpointing parameters—checkpoint threshold voltage V_{thresh} and (if applicable) timer interrupt interval—chosen by the programmer prior to run time based on simulation results. Instead of using a fixed configuration that may perform poorly when program behavior is highly variable or depends on environmental factors, future versions of Mementos will adaptively tune checkpointing behavior as described in Section 5.3.

7. Related Work

A wealth of research on checkpointing exists at various levels of computer systems. Most related approaches adopt a similar (if broader) approach to Mementos's: capture relevant program state. A key difference between Mementos and previous work is that, on RFID-scale devices, Mementos must consider catastrophic failure to be the *common case* and not an *occasional event*. We group related work into general checkpointing papers and papers related to tolerating failures on small-scale devices.

Checkpointing. We borrow our definition of *checkpointing* from Bernstein et al. [2], who define it as "an activity that writes information to stable storage during normal operation in order to reduce the amount of work [the system] has to do after a failure." Automatic checkpointing has long provided insurance against occasional failures. Systems in the 1980s and 1990s explored checkpointing for distributed systems [20, 26, 27], particularly for process migration or high-assurance computing. Checkpointing is especially useful for systems that handle precious data or make promises about fidelity, such as databases [2, 25] or file systems [37, 42].

Plank et al. [33] discuss checkpointing strategies in detail. Their portable *libckpt* library for UNIX implements both automatic (periodic, checkpoint-on-write) and user-directed checkpointing strategies. In the terminology of libckpt, Mementos implements *sequential* checkpointing, wherein the checkpointing procedure stops execution of the main program to capture its state. Like Mementos in timer-aided mode, libckpt automatically captures application state (registers and RAM) at a predefined frequency. Unlike Mementos, libckpt also supports *incremental* checkpointing by using page protection mechanisms to keep track of pages dirtied since the last checkpoint operation. We have not implemented a similar system because Mementos is designed to run directly on hardware.

Previous work has considered the use of static analysis and compile-time modifications to facilitate checkpointing. Compiler-assisted checkpointing systems [23, 24] require users to insert checkpointing cues into programs, unlike Mementos, although Mementos shares the notion of using compile-time instrumentation to make programs amenable to checkpointing. The Porch source-to-source compiler [41] enables programs to be suspended, migrated and resumed on different architectures. Porch uses compile-time analysis to generate program-specific checkpoint and resume functions specific to each possible stopping point. We consider Porch to be too heavyweight for Mementos's target platforms (owing to its lofty goals) although the checkpointing mechanism is similar.

Also relevant, perhaps surprisingly, are checkpointing systems that work on large-scale computers. These computers must tolerate frequent node failures, so job migration is a key feature. Bronevetsky et al. [3] propose a compile- and run-time system that modifies shared-memory programs and coordinates checkpointing and recovery among application threads. Their compiler techniques are essentially the same as Porch's and import the same differences versus Mementos.

Checkpointing for small-scale devices. Recent work in sensor networks considers the problem of whole-network checkpointing [30], using MSPsim for experimentation on continuously-powered sensor networks running the Contiki OS [13]. Their checkpointing mechanism saves the entire contents of a sensor node's memory, plus the state of several peripherals, via the node's serial port. A master node freezes and restores nodes using serial-port commands. Using an OS thread to save a complete memory dump is considerably simpler than Mementos; however, the required OS support for threads and the size of the resulting checkpoints make this approach impractical for RFID-scale devices.

The Neutron operating system [10], based on TinyOS [22], uses selective software restarts to mitigate the effects of software errors. Neutron allows programmers can mark "precious" state that must be preserved across software resets—but not across hardware reboots. Rather than requiring programmers to manually mark important state, Mementos favors an automatic approach. Mementos also does not require an operating system like TinyOS or Contiki. In our tests on a MSP430-based TinyNode [12], a vanilla TinyOS instance required 253.4 ± 1.5 ms to boot—much too slow to run a device that loses power every ~ 100 ms.

Specific to RFID-scale devices, Buettner et al. [5] describe WISP-based *RFID sensor networks* (RSNs) and the difficulty of predicting energy availability. They suggest, but do not implement, program splitting as an approach to execute large programs.

Chae et al. [9] implemented the RC5 block cipher on a WISP by carefully choosing parameters so that computations would finish in a single lifecycle. Mementos, aims to enable such resource-intensive programs to run to completion without requiring modifications to already-complex existing code.

Clark et al. [11] and Gummeson et al. [17] modify the WISP's hardware to increase its ability to survive power outages. Specifically, they experiment with larger capacitor sizes—which store more energy but take longer to charge—and auxiliary solar panels that together prolong the WISP's ability to retain state in low-power modes. Mementos eschews these hardware modifications for the sake of generality.

The flash storage mechanisms of Salajegheh et al. [38] treat flash memory as probabilistic "half-wits" and provide reliable writes to flash memory at voltages well below the nominal operating voltage specified by microcontrollers such as the MSP430. Mementos currently treats the nominal threshold values as hard boundaries; the half-wits result suggests that Mementos could relax those constraints to reliably write checkpoints to flash memory at voltages significantly below the 2.8 V threshold.

Scheduling on computational RFIDs. Buettner et al. integrate a voltage-aware task scheduler into the firmware of a WISP [7]. Given a measured voltage level, their scheduler selects a task from predefined set based on its stated resource requirements; they define tasks as small programs that can run to completion in a single lifecycle under reasonable energy conditions. Mementos instead focuses on completing a single task that might otherwise not complete in a single lifecycle.

8. Conclusions

Transiently powered RFID-scale devices enable general-purpose computation in scenarios where energy is scarce. However, the lack of a steady supply of energy results in frequent complete losses of power and state. Today, programmers either write short programs or hand-tune assembly code to ensure that computation finishes before a power loss—severely limiting the application space for these devices and making programming cumbersome and error prone.

Mementos addresses the challenge of enabling long-running programs to make steady progress on transiently powered devices. It instruments programs with energy checks at compile time and provides automatic state checkpointing and recovery at run time. A suite of simulation tools based on MSPsim enables a programmer to evaluate the behavior of Mementos-instrumented programs before deploying them.

Source code for Mementos, the simulator, and voltage traces for testing are available for download via the first author's web page.

Acknowledgments

This material is supported by a Sloan Research Fellowship and the NSF under CNS-0627529, CNS-0845874, NSF CNS-0923313, and a Graduate Research Fellowship. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

We thank John Brattin for code and discussions; Shane Clark for help with simulation; Mark Corner for providing resources; Michael Buettner, Chris Erway, and Quinn Stewart for feedback on drafts; Edwin Foo, Anton Korobeynikov, Dmitriy Matveev, and John Regehr for help with LLVM's MSP430 backend; Jeremy Gummeson and John Tuttle for conducting measurements; Scott Kaplan for advice on compression; Emery Berger for discussions; and Joshua Smith and Alanson Sample at Intel Labs Seattle for providing the WISP over the last three years.

We thank our shepherd, David Lie, for guidance and the anonymous ASPLOS reviewers for their helpful comments.

References

[1] W. Baek and T. Chilimbi. Green: A system for supporting energy-conscious programming using principled abstractions. Technical Report MSR-TR-2009-89, Microsoft Research, July 2009.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. ISBN 0-201-10715-5.

[3] G. Bronevetsky, D. Marques, K. Pingali, P. K. Szwed, and M. Schulz. Application-level checkpointing for shared memory programs. In *Proc. 11th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI)*, pages 235–247. ACM, Oct. 2004.

[4] J. Bruck, A. Vardy, A. Jiang, E. Yaakobi, J. Wolf, R. Mateescu, and P. Siegel. Storage coding for wear leveling in flash memories. In *IEEE Int'l Symposium on Information Theory (ISIT '09)*, pages 1229–1233, June 2009.

[5] M. Buettner, B. Greenstein, A. Sample, J. R. Smith, and D. Wetherall. Revisiting smart dust with RFID sensor networks. In *Proc. 7th ACM Workshop on Hot Topics in Networks (HotNets-VII)*, Oct. 2008.

[6] M. Buettner, R. Prasad, M. Philipose, and D. Wetherall. Recognizing daily activities with rfid-based sensors. In *Proc. 11th Int'l Conference on Ubiquitous Computing (UbiComp '09)*, pages 51–60. ACM, Sept. 2009.

[7] M. Buettner, B. Greenstein, and D. Wetherall. Dewdrop: An energy-aware task scheduler for computational RFID. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*. USENIX Association, Mar. 2011. To appear.

[8] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, Feb. 2006.

[9] H.-J. Chae, D. J. Yeager, J. R. Smith, and K. Fu. Maximalist cryptography and computation on the WISP UHF RFID tag. In *Proc. Conference on RFID Security*, July 2007.

[10] Y. Chen, O. Gnawali, M. Kazandjieva, P. Levis, and J. Regehr. Surviving sensor network software faults. In *Proc. ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*, pages 235–246, Oct. 2009.

[11] S. S. Clark, J. Gummeson, K. Fu, and D. Ganesan. Towards autonomously-powered CRFIDs. In *Workshop on Power Aware Computing and Systems (HotPower '09)*, Oct. 2009.

[12] H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler. TinyNode: a comprehensive platform for wireless sensor network applications. In *Proc. 5th Int'l Conference on Information Processing in Sensor Networks (IPSN '06)*, pages 358–365. ACM, Apr. 2006.

[13] A. Dunkels, B. Grönvall, and T. Voigt. Contiki—a lightweight and flexible operating system for tiny networked sensors. In *Proc. First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*. IEEE Computer Society, Nov. 2004.

[14] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and T. Voigt. MSPsim—an extensible simulator for MSP430-equipped sensor boards. In *Proc. 4th European Conference on Wireless Sensor Networks (EWSN '07), Poster/Demo session*. Springer, Jan. 2007.

[15] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology—CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, Aug. 2010.

[16] M. Gorlatova, P. Kinget, I. Kymissis, D. Rubenstein, X. Wang, and G. Zussman. Challenge: Ultra-low-power Energy-Harvesting Active Networked Tags (EnHANTs). In *Proc. 15th Annual Int'l Conference on Mobile Computing and Networking (MobiCom '09)*, pages 253–260. ACM, Sept. 2009.

[17] J. Gummeson, S. S. Clark, K. Fu, and D. Ganesan. On the limits of effective micro-energy harvesting on mobile CRFID sensors. In *Proc. 8th Annual ACM/USENIX Int'l Conference on Mobile Systems, Applications, and Services (MobiSys '10)*. ACM, June 2010.

[18] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, 1989. ISBN 0-521-37095-7.

[19] S. F. Kaplan. *Compressed caching and modern virtual memory simulation*. PhD thesis, University of Texas at Austin, Dec. 1999.

[20] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. In *Proceedings of 1986 ACM Fall Joint Computer Conference (ACM '86)*, pages 1150–1158. IEEE Computer Society, 1986.

[21] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. 2004 Int'l Symposium on Code Generation and Optimization (CGO'04)*. ACM, Mar. 2004.

[22] 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 sensor networks. In *Ambient Intelligence*. Springer Verlag, 2004.

[23] C. Li and W. Fuchs. CATCH—Compiler-assisted techniques for checkpointing. In *Digest of Papers, 20th Int'l Symposium on Fault-Tolerant Computing (FTCS-20)*, pages 74–81. IEEE, June 1990.

[24] C. Li, E. Stewart, and W. Fuchs. Compiler-assisted full checkpointing. *Software: Practice & Experience*, 24(10):871–886, 1994.

[25] J.-L. Lin, M. H. Dunham, and M. A. Nascimento. A survey of distributed database checkpointing. *Distributed and Parallel Databases*, 5(3):289–319, July 1997.

[26] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor—a hunter of idle workstations. In *Proc. 8th Int'l Conference on Distributed Computing Systems (ICDCS '88)*, pages 104–111. IEEE Computer Society, June 1988.

[27] J. A. McDermid. Checkpointing and error recovery in distributed systems. In *Proc. 2nd Int'l Conference on Distributed Computing Systems (ICDCS '81)*, pages 271–282. IEEE Computer Society, 1981.

[28] S. Misailovic, S. Sidiroglou, H. Hoffman, and M. C. Rinard. Quality of service profiling. In *Proc. 32nd ACM/IEEE Int'l Conference on Software Engineering (ICSE '10)*. ACM, May 2010.

[29] J. Olivo, S. Carrara, and G. D. Micheli. Energy harvesting and remote powering for implantable biosensors. *IEEE Sensors Journal*, PP(99), October 2010.

[30] F. Österlind, A. Dunkels, T. Voigt, N. Tsiftes, J. Eriksson, and N. Finne. Sensornet checkpointing: Enabling repeatability in testbeds and realism in simulators. In *Proc. 6th European Conference on Wireless Sensor Networks (EWSN '09)*. Springer, Feb. 2009.

[31] J. A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1):18–27, 2005.

[32] J. S. Plank, M. Beck, and G. Kingsley. Compiler-assisted memory exclusion for fast checkpointing. *IEEE Technical Committee on Operating Systems and Application Environments*, 7(4):10–14, Winter 1995.

[33] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under Unix. In *Proc. USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems*, pages 213–223, Jan. 1995.

[34] P. Plauger. Embedded C++: An Overview. *Embedded Systems Programming*, 10:40–53, 1997.

[35] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proc. 4th Int'l Symposium on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS '05)*. IEEE, Apr. 2005.

[36] B. Ransford, S. Clark, M. Salajegheh, and K. Fu. Getting things done on computational RFIDs with energy-aware checkpointing and voltage-aware scheduling. In *USENIX Workshop on Power Aware Computing and Systems (HotPower '08)*, Dec. 2008.

[37] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):26–52, 1992.

[38] M. Salajegheh, Y. Wang, K. Fu, A. A. Jiang, and E. Learned-Miller. Exploiting half-wits: Smarter storage for low-power devices. In *Proc. 9th USENIX Conference on File and Storage Technologies (FAST '11)*, Feb. 2011. To appear.

[39] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith. Design of an RFID-based battery-free programmable sensing platform. *IEEE Transactions on Instrumentation and Measurement*, 57(11):2608–2615, Nov. 2008.

[40] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: A language and runtime system for perpetual systems. In *Proceedings of The Fifth Int'l ACM Conference on Embedded Networked Sensor Systems (SenSys '07)*, pages 161–174, Nov. 2007.

[41] V. Strumpen. Portable and fault-tolerant software systems. *IEEE Micro*, 18(5):22–32, 1998.

[42] S. Sundararaman, S. Subramanian, A. Rajimwale, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. M. Swift. Membrane: Operating system support for restartable file systems. In *Proc. 8th USENIX Conference on File and Storage Technologies (FAST '10)*, Feb. 2010.

[43] Texas Instruments Incorporated. MSP430 Ultra-Low Power Microcontrollers. http://www.ti.com/msp430.

[44] S. Thomas, J. Teizer, and M. Reynolds. Electromagnetic energy harvesting for sensing, communication, and actuation. In *Proc. 27th Int'l Symposium on Automation and Robotics in Construction (ISARC '10)*. IAARC, June 2010.

[45] Y. Yang, L. Wang, D. K. Noh, H. K. Le, and T. F. Abdelzaher. SolarStore: Enhancing data reliability in solar-powered storage-centric sensor networks. In *Proc. 7th Annual Int'l Conference on Mobile Systems, Applications, and Services (MobiSys '09)*, pages 333–346. ACM, 2009.

[46] D. Yeager, F. Zhang, A. Zarrasvand, N. George, T. Daniel, and B. Otis. A 9 μa, addressable Gen2 sensor tag for biosignal acquisition. *IEEE Journal of Solid-State Circuits*, 45(10):2198–2209, Oct. 2010.

[47] E. Yeatman. Advances in power sources for wireless sensor nodes. In *Proc. Int'l Workshop on Wearable and Implantable Body Sensor Networks (BSN '04)*, pages 20–21. IEEE Computer Society, Apr. 2004.