5-2013

Transiently Powered Computers

Benjamin Ransford

University of Massachusetts Amherst, ben@ransford.org

TRANSIENTLY POWERED COMPUTERS

A Dissertation Presented

by

BENJAMIN RANSFORD

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2013

School of Computer Science

TRANSIENTLY POWERED COMPUTERS

A Dissertation Presented

by

BENJAMIN RANSFORD

Approved as to style and content by:

_____

Kevin Fu, Chair

_____

Emery D. Berger, Member

_____

Wayne P. Burleson, Member

_____

Deepak Ganesan, Member

_____

Lori A. Clarke, Chair
School of Computer Science

For Ilse and Linus and Megan.

ACKNOWLEDGMENTS

March 2011. Another version appeared in October 2010 as Tech Report UM-CS-2010-060 of the University of Massachusetts Amherst Department of Computer Science.

- "CCCP: Secure Remote Storage for Computational RFIDs" by Mastooreh Salajegheh, Shane S. Clark, Benjamin Ransford, Kevin Fu and Ari Juels. In Proceedings of the 18th USENIX Security Symposium, Montreal, Ontario, Canada, August 2009.

- "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses" by Daniel Halperin, Thomas S. Heydt-Benjamin, Benjamin Ransford, Shane S. Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno and William H. Maisel. In Proceedings of the 29th IEEE Symposium on Security and Privacy (Oakland), Berkeley, CA, May 2008.

- "Design Challenges for Secure Implantable Medical Devices" by Wayne P. Burleson, Shane S. Clark, Benjamin Ransford, and Kevin Fu. In Proceedings of the 49th Design Automation Conference (DAC), San Francisco, CA, June 2012.

ABSTRACT

TRANSIENTLY POWERED COMPUTERS

May 2013

BENJAMIN RANSFORD

B.S., CORNELL UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Kevin Fu

Demand for compact, easily deployable, energy-efficient computers has driven the development of general-purpose *transiently powered computers* (TPCs) that lack both batteries and wired power, operating exclusively on energy harvested from their surroundings.

TPCs' dependence solely on transient, harvested power offers several important design-time benefits. For example, omitting batteries saves board space and weight while obviating the need to make devices physically accessible for maintenance. However, transient power may provide an unpredictable supply of energy that makes operation difficult. A predictable energy supply is a key abstraction underlying most electronic designs. TPCs discard this abstraction in favor of *opportunistic* computation that takes advantage of available resources. A crucial question is *how should a software-controlled computing device operate if it depends completely on external en-*

tities for power and other resources? The question poses challenges for computation, communication, storage, and other aspects of TPC design.

The main idea of this work is that **software techniques can make energy harvesting a practicable form of power supply for electronic devices.** Its overarching goal is to facilitate the design and operation of *usable* TPCs.

This thesis poses a set of challenges that are fundamental to TPCs, then pairs these challenges with approaches that use software techniques to address them. To address the challenge of computing steadily on harvested power, it describes Mementos, an energy-aware state-checkpointing system for TPCs. To address the dependence of opportunistic RF-harvesting TPCs on potentially untrustworthy RFID readers, it describes CCCP, a protocol and system for safely outsourcing data storage to RFID readers that may attempt to tamper with data. Additionally, it describes a simulator that facilitates experimentation with the TPC model, and a prototype computational RFID that implements the TPC model.

To show that TPCs can improve existing electronic devices, this thesis describes applications of TPCs to implantable medical devices (IMDs), a challenging design space in which some battery-constrained devices completely lack protection against radio-based attacks. TPCs can provide security and privacy benefits to IMDs by, for instance, cryptographically authenticating other devices that want to communicate with the IMD before allowing the IMD to use any of its battery power. This thesis describes a simplified IMD that lacks its own radio, saving precious battery energy and therefore size. The simplified IMD instead depends on an RFID-scale TPC for *all* of its communication functions.

TPCs are a natural area of exploration for future electronic design, given the parallel trends of energy harvesting and miniaturization. This work aims to establish and evaluate basic principles by which TPCs can operate.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Designers of computing devices constantly run into barriers when the time comes to consider power. Tethered AC power is not available everywhere and requires relatively large switching circuitry to obtain the needed DC power. Batteries are an appealing alternative to tethered power, but they have failed to keep pace with trends in chip and circuit scaling. Consequently, many electronic devices have their size, weight, and therefore mobility determined by the batteries that power them.

This thesis posits a world in which *energy harvesting* allows electronic devices to operate only on the energy they harvest, without batteries. Some such devices already exist; others will appear as the building blocks for energy harvesting circuits become more widely used and more efficient.

Depending on harvested energy can be challenging for devices. They rely on external parties (or phenomena) instead of themselves. The performance of energy harvesting can fluctuate, leading to inconvenient power losses or unpredictable power. Technical limitations of harvesting impose an upper bound on device specifications such as maximum current. These are the problems that afflict what we call *transiently powered computers*; addressing them is the focus of this thesis.

1.1 Background and Motivation

Demand for small, easily embedded computers and sensors is driving the development of general-purpose *transiently powered computers* (TPCs) that lack both batteries and wired power and that operate exclusively on energy harvested from ex-

ternal supplies or environmental phenomena. Such devices range from *computational RFIDs* (CRFIDs) [30], which are microcontroller-based devices that harvest energy from RFID readers via radio (RF) waves, to general-purpose batteryless sensor devices [169] powered by solar panels, to platforms powered by RF–solar hybrid harvesting [62], to "interaction-powered" wireless user-interface devices [164]. Academic and industrial research continues to produce new energy-harvesting mechanisms, generating power from phenomena as diverse as vibration [7], heat differentials [71], and stomach fluids [43].

Without batteries—which require periodic replacement, are relatively heavy, and dominate board layouts—TPCs require virtually no maintenance and can be embedded in situations for which battery-powered computers are unsuitable, such as inside building materials or living tissue.

1.2   Challenges

The transient manner of TPCs' operation has burdensome implications for system design, which manifest as the following challenges:

1. **Power.** TPCs that operate on harvested energy may operate very close to their minimum requirements, resulting in constant losses of power that result in the disappearance of volatile state and increased risk of soft errors.

2. **Communication.** On unpredictable power, a TPC cannot depend on being able to schedule, or even initiate, communications—which may break protocol participation or other application assumptions. Additionally, TPCs' power budgets may preclude them from having active radios, depending instead on other less flexible forms of communication (e.g., backscatter radiation for CRFIDs).

3. **Security.** TPCs completely depend on entities that are external to them to provide power. For some TPCs such as CRFIDs that also use the power link for

2

data exchange, this complete dependence raises the question of trust: how can such a TPC offer any kind of application-level guarantees of data integrity or confidentiality if the tag will communicate with any entity that agrees to power it?

4. **Storage.** Nonvolatile memories such as NOR flash are available on-chip in many microcontrollers, but these memories require two orders of magnitude more energy than volatile memories per byte written. Applications that require persistent storage must use nonvolatile memory judiciously, if at all.

Each of these challenges limits the suitability of energy harvesting as the sole supply of power for devices. However, software-based approaches can provide solutions that correct or ameliorate each of these problems.

1.3 Thesis Statement and Summary

The motivating idea of this thesis is:

> *Software techniques on simple energy-harvesting hardware can make energy harvesting a practicable way to power computing devices, and can expand the capabilities of electronic devices without imposing extra energy costs.*

The software techniques in this thesis are meant to complement, not replace, energy-harvesting hardware. Rather than posing the question of how to harvest energy most effectively, the thesis focuses on how to build capable, trustworthy systems from components that are unreliably powered, energy constrained, and dependent on possibly untrustworthy external devices.

The thesis comprises two main thrusts, summarized below. The first describes problems that are peculiar to TPCs and evaluates software systems that are designed to address the challenges listed at the beginning of this section. The second explores a specific application of TPCs to demonstrate empirically that introducing a TPC can materially improve an existing system.

1. **Thrust #1: Addressing TPC challenges.** In light of the challenges described above, I describe and demonstrate the effectiveness of Mementos, a software system designed to make computation on TPCs robust against frequent power failures. In particular, I perform end-to-end evaluation with a cycle-accurate, energy-accurate simulator to show that Mementos enables TPCs to run applications that fail to run without it. I also describe CCCP, a communication mechanism that provides cryptographically protected off-board storage for CRFIDs that is *less energy intensive* than on-board flash storage for data sizes above 32 bytes.

2. **Thrust #2: Improving devices with TPCs.** This thrust focuses on an application area for TPCs: medical devices that operate inside body tissue. I first offer a template for augmentation of a device with a TPC, in the form of a challenge–response protocol for CRFIDs that endows an implantable cardiac device with an authentication mechanism. Cardiac devices such as pacemakers are particularly good candidates for augmentation with TPCs because (1) the total implanted size depends largely on battery capacity and (2) battery capacity depends largely on non-life-supporting functions such as wireless communication and signal processing. Extending the aforementioned template for augmentation with a TPC, I implement Noradio, a simplified pacemaker, and show how to decompose its functions into therapeutic sensing and actuation, on one hand, and communication and storage, on the other. For the communication subsystem, I use a UMass Moo CRFID to replace the pacemaker's on-board radio.

1.4   Contributions

To address the problem of constant power loss destroying volatile state, I describe *Mementos*, a system that instruments programs with energy checks that induce

energy-aware state checkpoints [124]. Designed for use on CRFIDs but applicable to other TPCs, Mementos continually monitors the voltage of an energy buffer (e.g., a storage capacitor) at run time. When the voltage drops below a programmer-specified threshold value, a checkpointing routine writes all of the program's volatile state (registers, stack, and globals) to nonvolatile memory, where it survives during a power loss. At next boot, a checkpoint-restoration routine copies the state from nonvolatile memory back to volatile memory and resumes execution. Mementos thereby spreads computations across multiple power lifecycles. Compared to previous checkpointing systems, Mementos is designed for tighter resource constraints and a finer, configurable granularity of checkpointing based on energy available at run time.

Additionally, I describe a CRFID simulator, originally designed for use with Mementos, that couples an architectural cycle-accurate MSP430 microcontroller simulator [49] with an energy-trace-driven simulated energy-harvesting front end that governs execution. Unlike most architectural or circuit simulators, this CRFID simulator incorporates inputs from the analog *and* digital domains.

A second contribution is a mechanism for secure outsourced storage for CRFIDs. A system called CCCP—for *cryptographic computational continuation passing* [135]—addresses the challenges of security and storage and partially addresses the challenge of communications. CCCP is software that provides CRFIDs with an off-tag storage facility layered on existing radio protocols. As data size increases, CCCP requires less energy than local storage on flash memory.

A third contribution is a "zero-power" security mechanism that endows a medical devices with enhanced security properties without requiring any of the medical device's limited battery power. Modern implantable medical devices (IMDs) are sensing and actuation devices that depend on a nonrechargeable battery for power. Unfortunately, IMDs designed without security as a design goal are vulnerable to battery-depletion attacks that can significantly reduce the device's availability over

5

time. I present *WISPer*, a mechanism designed to augment an IMD with security features missing from its original design: authentication of external parties, and a "sensible" key-exchange mechanism that enables patients to sense security-sensitive events.

A final contribution is an extension of WISPer that outsources *all* the communications of a medical device to a TPC in order to save energy versus an active radio. Extending the example of WISPer, I design and implement *Noradio*, a prototype simulated pacemaker, and evaluate its energy use both with a radio—when it resembles cardiac IMDs on the market—and with all of its communication outsourced to a connected CRFID. The most salient novel component of this preliminary work is the method of completely outsourcing communication, turning synchronous protocol participation into asynchronous, query-oriented data collection. Restructuring telemetry transmissions as an asynchronous, CRFID-driven task results in a 44% reduction of current consumption during communication. The techniques I develop to build Noradio are applicable to other devices that are battery powered and use radios to communicate.

1.5 Thesis Outline

The remainder of this thesis is structured as follows.

Chapter 2 describes computational RFIDs, a kind of TPC that is powered by harvested radio frequency energy. Reproducibility of experiments is problematic on CRFIDs, and existing simulation tools are not well suited to simulating them, so the chapter develops a simulation framework that accurately captures the behavior of a CRFID.

Chapter 3 considers the problem of computing steadily on TPCs. It describes Mementos [124], a system that protects computations from power failures with energy-aware state checkpointing.

Chapter 4 addresses communication and storage on CRFIDs. It describes CCCP [135], a mechanism for outsourced storage that improves energy efficiency for CRFIDs storing state checkpoints.

Chapter 5 summarizes the security problems inherent in an implantable medical device [27, 69], then describes WISPer, a TPC-based communication subsystem that uses lightweight cryptography and a simple challenge–response protocol to address several of the security problems.

Chapter 6 describes an emulated cardiac pacemaker that couples with a TPC to obviate the need for any radio—typically an energy-hungry component—on board the pacemaker at all.

Chapter 7 discusses the implications of this work's improvements to and study of TPCs—in particular, ways in which they can enable new kinds of devices and improve battery-powered devices.

CHAPTER 2

SIMULATING TRANSIENTLY POWERED COMPUTERS[1]

Modeling and simulation together provide a convenient way to experiment on the design of computing systems. A typical research approach is to change designs in simulation before fabricating hardware. System designers can choose tools that model computers from the circuit level to the OS level—but these models typically do not span the analog and digital domains.

Transiently powered computers are difficult to model with conventional analog- or digital-domain tools because analog-domain inputs (specifically, the amount of energy available from the power supply) govern execution in the digital domain. Digital-domain simulators' abstraction of the power supply as a bottomless resource therefore does not apply.

This chapter's contribution is a simulator for TPCs that incorporates both digital-domain and analog-domain inputs—a computer program and an energy trace against which to run the program. The simulator is an extension of MSPsim [49], a cycle-accurate simulator for the MSP430 family of microcontrollers, that newly incorporates models of energy harvesting and consumption. We tuned and evaluated the model according to empirical measurements of two *computational RFIDs* (CRFID, Section 2.1) that perform sensing and computation tasks solely on harvested radio frequency (RF) energy.

---

[1]This chapter describes extensions to a simulator developed for the Mementos system presented in Chapter 3 and in an earlier paper by Ransford, Sorber, and Fu [124].

8

To simulate the running of a program on a CRFID under energy harvesting, the simulator adjusts current consumption to match empirical measurements of hardware MSP430 microcontrollers. An input energy trace, collected with an analog RF-harvesting front end from a CRFID, charges a simulated capacitor that models a CRFID's storage capacitor. The simulated microcontroller executes instructions from the input program, drawing an empirically calibrated simulated current according to its activity mode, thereby discharging the capacitor. When the capacitor's simulated voltage declines to a certain level, a simulated power loss occurs, and components of the simulator reset—all in-progress operations stop, all volatile memory is zeroed, and the program counter is reset to the beginning of code memory.

2.1 Computational RFIDs

Computational RFIDs (CRFIDs) are a class of programmable, batteryless computers that operate solely on harvested energy, much like widespread passive RFID tags used in supply-chain applications [23, 22, 122, 124].

The key difference between supply-chain RFID tags and CRFIDs is that CRFIDs use general-purpose microprocessors that allow them to execute general-purpose programs; the former are typically implemented as application-specific integrated circuits (ASICs), resulting in low power consumption but fixed functionality that is tightly coupled to certain applications. Supply-chain ASICs often implement only an RFID protocol state machine and a chunk of memory containing static information. This static information is particularly useful for tracking retail objects.

The first instance of a CRFID is the Wireless Identification and Sensing Platform, or WISP [138], Figure 2.1, a prototype device slightly smaller than a postage stamp (discounting its antenna, which is several centimeters long). The WISP is built around an off-the-shelf TI MSP430 microcontroller, as are many of its descendants, such as the UMass Moo [175], Figure 2.2.

The WISP and its descendants share physical attributes and, to an extent, circuit designs. The common elements of these prototype CRFIDs are:

- a low-power microcontroller such as the TI MSP430 [155];

- on-chip RAM;

- flash memory (on or off the microcontroller);

- energy-harvesting circuitry tuned to a certain frequency (e.g., 913 MHz for EPC Gen 2 RFID);

- an antenna;

- a transistor between the antenna and the microcontroller to modulate the analog front end's impedance;

- a capacitor for storage of harvested energy;

- optional sensors for physical phenomena such as acceleration, heat, or light; and

- one or more analog-to-digital converters (ADCs) for measuring voltage (and interacting with sensors).

The energy-harvesting circuitry, antenna, and transistor comprise the CRFID's *analog front end*. The microcontroller, memory, sensors, and other digital components together comprise the *digital back end*. The analog front end provides power (by charging the capacitor) and information to the digital back end, which can in turn transmit its outputs via the analog front end.

CRFIDs combine properties of supply-chain RFID tags with those of sensor motes such as the Telos [117]. Like passive RFID tags but unlike motes, CRFIDs are powered solely by harvested RF energy and lack active radio components (dedicated radio circuits or chipsets that require their own power). Instead of active radios, CRFIDs
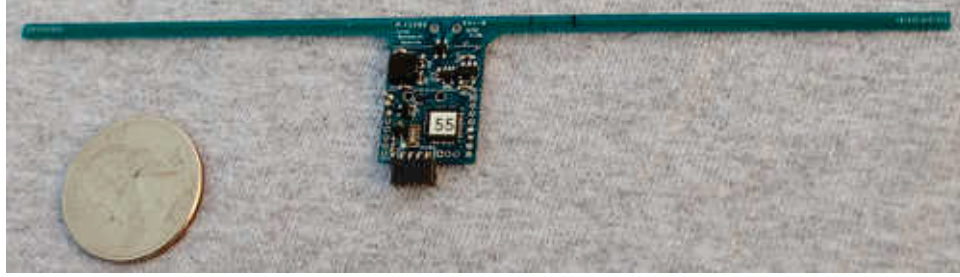
Figure 2.1. Photo of DL WISP 4.1 [138].

Figure 2.2. Photo of UMass Moo [175].

use *backscatter* communication: in the presence of incoming radio waves, a CRFID electrically modulates its analog front end's impedance using a transistor, encoding binary information by varying the amount of signal the antenna reflects. While the omission of active radio circuitry saves energy, it gives up the tag's autonomy; a CRFID can send and receive information only at the command of an RFID reader.

The small size and low maintenance requirements of CRFIDs make them especially appealing for adding computational capabilities to contexts in which placing or maintaining a conventional computer would be infeasible or impossible. However, CRFID systems require that nearby, actively powered RFID readers provide energy whenever computation is to occur, a requirement that may not suit all applications.

2.2 Reproducibility and Simulation

A key concern in system design and evaluation is the classic scientific goal of *reproducibility*: People other than the researcher should be able to achieve the same

results under similar conditions, and the researcher herself should be able to achieve the same results multiple times.

A typical approach to enabling reproducible experiments is *simulation*, in which a computer program models all or part of a system to predicts its behavior under given input conditions. For computers, traditional simulation methods fall into two categories. *Architectural* simulators such as Simics [145] or gem5 [56, 99] implement the target architecture's instruction set and model pipelines, memory, and various other hardware features at the digital level. In contrast, *circuit-level* simulators such as those based on SPICE [100] model hardware components at the analog level, enabling accurate predictions of electrical behavior, thermal behavior [147], or component reliability [12] under a variety of conditions.

In contrast to conventional computers that are continuously powered, a TPC's behavior can be difficult to predict because energy harvesting can be unpredictable. RF harvesting, in particular, depends on factors such as the physical orientation of the harvesting antenna in relation to the energy source, the distance from the source, the variety of paths an electromagnetic wave can take from the source to the harvester, and the physical properties of materials in the vicinity of the source and harvester. Other harvesting modalities come with similar complications—light levels for solar panels, elasticity and reverberations for vibration harvesters, heat dissipation for thermal harvesters, and so on.

Because TPCs' behavior depends so much on environmental factors, traditional simulators used to model continuously powered computers are a poor match for simulating TPCs. Traditional simulators are designed for *deterministic* simulations in which the same inputs—either analog or digital—produce the same outputs. In the case of a TPC, both kinds of inputs matter. Nondeterministic energy harvesting results in pauses or stoppages of computation, re-execution of instructions, and loss of computational state at inconvenient times, and it is unlikely that any two runs of a

12

Feature	Analog sim. (e.g., SPICE)	Digital sim. (incl. MSPsim)	MSPsim + this work
Electrical simulation	✓	—	✓[a]
Instruction-set emulation	—	✓	✓
Can run compiled programs	—	✓	✓
Accurate memory timing	✓	✓	✓
Cycle-accuracy	✓	✓	✓
Simulation of power supply	✓	—	✓
Radio-wave simulation	—	—	—

Table 2.1. Comparison of simulator features.

---

[a]Our tools simulate a *subset* of the electrical components on a CRFID, most crucially the storage capacitor that buffers incoming energy and obeys the standard capacitor equations.

program on a TPC will have exactly the same energy conditions. Even "full-system" simulators such as FeS$_2$ [51] abstract away analog-domain details such as power and temperature. This abstraction is reasonable for many scenarios, but for TPCs, ignoring analog effects on digital computations may result in incorrect or misleading conclusions.

This chapter describes a set of modifications to MSPsim [49], a cycle-accurate simulator for the MSP430 family of microcontrollers, to capture the peculiarities of transiently powered computers. MSPsim models mote-class sensor devices by simulating an MSP430 microcontroller [155] and a variety of input/output devices (including active radio hardware and serial peripherals). MSPsim's microcontroller simulation implements the 16-bit MSP430 (and 20-bit MSP430X) instruction set architecture, allowing it to accept MSP430 object code and execute it in a cycle-accurate manner.

Our modifications to MSPsim fall into two categories: modeling energy harvesting and simulating power loss and restoration. The result is a simulation that incorporates inputs from the analog and digital domains and enables reproducible, realistic behavior for a TPC under simulation. Table 2.1 compares this work to conventional analog and digital simulators.

CRFID property	Simulator mechanism
MSP430 MCU	MSPsim fully supports MSP430 ISA
RF harvesting	Accept voltage traces recorded on a hardware CRFID's analog front end
Low-power modes	Recognize programmatic transitions to low-power mode; change simulated MCU's current consumption
Electrical current	Use empirically determined mapping of MCU power mode and voltage to current
Dynamic power	Obey capacitor equations for charging and discharging under load
Quiescent power	Obey capacitor equations for exponential decay (leakage)
Flash, ADC	Use empirically measured time and current per operation
Radio	Support bit-banging transmissions onto GPIO

Table 2.2. Mapping of Intel WISP and UMass Moo hardware properties to simulator mechanisms.

For a concrete choice of TPC to simulate, the system described in this chapter models the DL WISP 4.1 [138], an RF-harvesting TPC with an MSP430F2132 microcontroller and a $10\,\mu$F storage capacitor, and its descendant the UMass Moo [175], which resembles the WISP but features an MSP430F2618 microcontroller with increased RAM and flash memory. Table 2.2 maps specific hardware properties of these CRFIDs to their simulations.

The simulation models most properties of the WISP and Moo, but its treatment of backscatter communication merits special description. Backscatter modulation operates on the same radio waves that provide energy, making synchronous uplink communication effectively "free"—modulo the negligible cost of switching a single transistor—on backscattering devices. Our simulation therefore models only the time and cycle count of the backscatter modulation. The simulator also makes no attempt to account for certain environmental parameters such as temperature because the variations they induce are typically small under laboratory conditions (and predictable otherwise).

2.3 Modeling Energy Harvesting in MSPsim

Like the traditional simulators described above, MSPsim models computing components in the digital domain, abstracting away analog components such as the power supply. In MSPsim's simulation of a mote, the digital components are conceptually backed by an imaginary battery that provides sufficient voltage and current to every subsystem. Unlike the continuously powered motes MSPsim models, a CRFID may lose power several times per second as it simultaneously computes and harvests RF energy, and the available energy may not be enough to support the components that should run. To match the behavior of a CRFID simulated in MSPsim to that of a real CRFID, we extended MSPsim to model the WISP's energy harvesting. (The Moo shares the WISP's energy-harvesting components, so we did not model it separately.)

The WISP's energy-harvesting hardware, depicted in Figure 2.3 and described in detail in Sample et al. [138], comprises a dipole antenna, a charge pump, and a (manually) tuned impedance-matching network that optimizes the transmission of power from the antenna to the charge pump. The charge pump increases the voltage coming from the antenna—which is small at distance from an RFID reader—and charges a storage capacitor that smooths and buffers the supply. The storage capacitor powers the microcontroller via a voltage regulator that maintains a constant voltage.

From the microcontroller's perspective, the storage capacitor hides the details of the analog energy-harvesting front end. By measuring the capacitor's voltage V_c, an application on the microcontroller can learn several things [124]:

- If $V_c \gg V_{reg}$, the regulator's target voltage, then there is likely enough voltage to continue computing; the CRFID may be near a reader.

- If V_c is slightly greater than V_{reg}, the CRFID's power consumption may be outpacing its energy harvesting, or the CRFID may be at such a distance from the reader that harvesting is difficult.
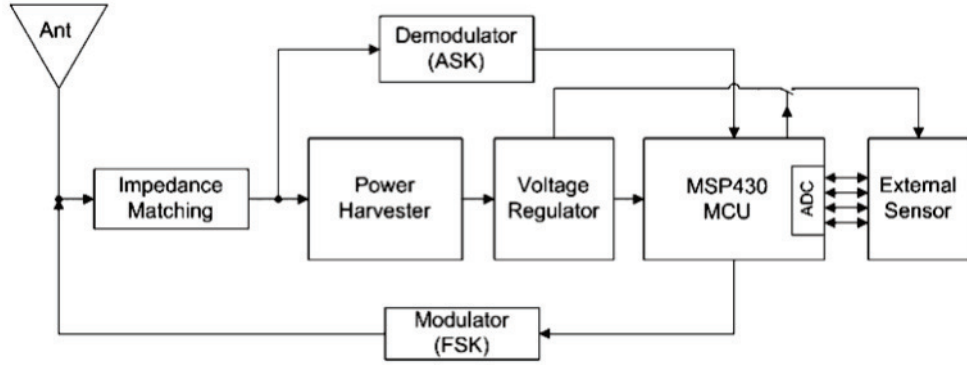
Figure 2.3. Block diagram of DL WISP 4.1. Reproduced from Sample et al. [138] with permission.
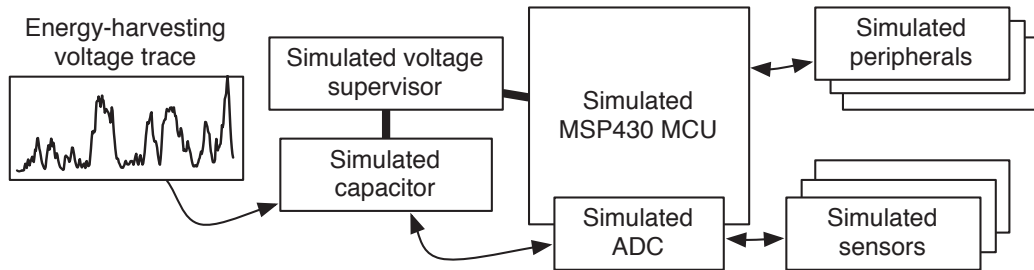
Figure 2.4. Block diagram of our simulation environment based on MSPsim.

- If $V_c < V_{reg}$, a power outage may be imminent.

Because the capacitor effectively isolates the analog front end from the digital back end, our simulation approach is to abstract away the harvesting hardware and simulate the storage capacitor rather than the harvesting components—effectively *emulating*, rather than simulating, the analog front end. Figure 2.4 is a block diagram depicting the simulation environment.

The main loop of the simulator consumes instructions from the input executable. At each iteration of the main loop (i.e., after each instruction is executed), it recalculates the capacitor's voltage. Being a cycle-accurate simulator with a known clock rate, the simulator keeps track of time via its cycle counter; from this source

the simulated capacitor derives its notion of time for use in its voltage calculations, described in the following paragraph.

The simulated capacitor follows the time-dependent equations that govern a real capacitor's operation. The equation for a charging capacitor is:

$$V(t) = V_0(1 - e^{-t/RC}),$$

(2.1)

where R and C are the load and capacitance of the circuit, and V_0 is the "initial" charge. Each time the simulator recalculates the capacitor's voltage (i.e., with each CPU instruction), V_0 is the voltage calculated at the previous check, R depends on the power mode, and C is the storage capacitor's (constant) capacitance, which dominates the overall circuit's capacitance. The discharge equation is

$$V(t) = V_0 e^{-t/RC}$$

(2.2)

where V_0, R, and C are defined in a similar manner.

2.3.1 Energy-Harvesting Traces

We modified MSPsim to accept a voltage trace that governs energy availability over time. On a hardware CRFID, the capacitor's voltage is a proxy for energy availability and usage—it increases as the analog front end gathers energy from RF waves, and decreases as a factor of both time (via leakage) and current consumption (via computation).

To capture voltage traces from real hardware, we physically isolated a WISP's RF-harvesting analog front end by cutting its connections to other parts of the circuit, attached it to a ~ 10 KΩ resistor that approximated the electrical load of the WISP's
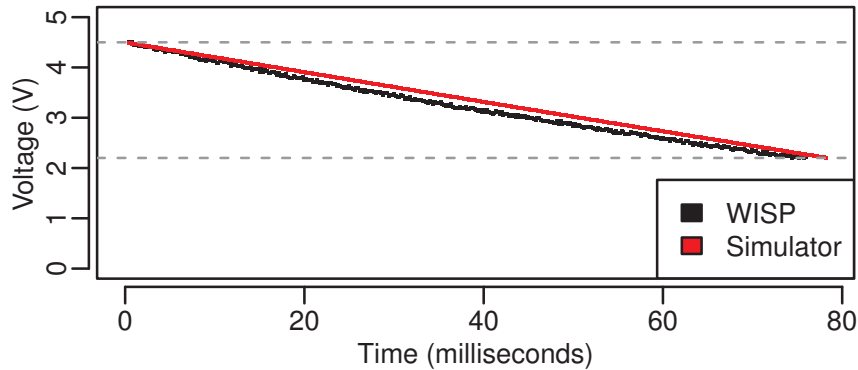
Figure 2.5. Simulated capacitor's voltage approximates the discharge time and voltage drop of a hardware WISP's capacitor. Both were charged to 4.5 V and allowed to discharge while executing an infinite loop at 1 MHz in active mode. Both traces end at 2.2 V, the nominal minimum voltage for flash writes on an MSP430.

microcontroller during active computation,[2] and connected an Agilent U2541A data acquisition unit sampling at 1 KHz. With this measurement setup, we recorded traces of voltage across the resistor (sequences of *time, voltage* pairs) as the measurement setup moved within the read range of an RFID reader. Figure 3.1 [p. 29] shows several of these traces.

2.3.2 Simulating Power Loss

To model the frequent losses of power that afflict CRFIDs under normal circumstances, we extended MSPsim with an MSP430 microcontroller's shutdown and reset behavior (documented in its datasheet [156]).

When energy is no longer sufficient, our modified MSPsim stops executing instructions (finishing the currently executing instruction, if there is one), records statistics about the computation's progress (e.g., number of cycles executed since the last reset), and informs the simulated capacitor that it should use the input voltage trace as

---

[2]Because the microcontroller typically spends much of its time in "active" mode when computing, for the purpose of trace collection we chose a fixed value for the resistor that matched the load of an "active" MSP430 microcontroller.

its time reference. When the simulated capacitor has recharged to a threshold value, MSPsim receives a reset interrupt, beginning the boot sequence.

At boot, MSPSim resets the program counter and status register. The firmware program's initialization routine resets the stack pointer and contents of RAM.

We did not attempt to the model *memory remanence*—the accidental retention of memory contents after power loss—though it appears that the contents of an MSP430 microcontroller's SRAM cells may persist for at least a few seconds before decaying [121], depending on the size of the storage capacitor, which can provide sufficient power for RAM retention after harvesting becomes ineffective. Instead, our modified MSPsim resets all registers and the contents of memory at each boot.

2.4   Profiling Energy Consumption

Section 2.3 discussed our model of energy harvesting and storage. This section concerns the model of energy consumption we incorporated into MSPsim. We profiled execution of a variety of micro-benchmarks on DL WISP 4.1 hardware [138] with MSP430F2132 microcontrollers, comparing energy properties with values from the microcontroller's datasheet [156] when possible. We repeated this task for the UMass Moo [175].

2.4.1   Instruction-Level Energy Measurements

As on larger-scale architectures, programs on the MSP430 microcontroller are subject to constraints imposed by hierarchical memory organization. On a typical CPU, registers are faster than caches, which are faster than RAM, which is faster than disk, with register access requiring the least energy and disk the most (preserving the ordering). A flash-based MSP430 like the WISP's includes registers, RAM, and flash memory within a single package. To understand the relative energy costs of operations on these different memory types, we wrote a set of programs to test memory operations

in a loop; each program tests one type of memory access (e.g., a memory-to-register read) repeated a known number of times after an initial period of low-power sleeping. We charged a WISP's storage capacitor to 4.5 V with a benchtop power supply and allowed the capacitor to discharge such that the loop of instructions occurred after the power supply was disconnected. We used an Agilent Infiniium 54832D MSO oscilloscope sampling at 1 KHz to measure the voltage drop ΔV due to instruction execution. The energy stored in a capacitor with capacitance C at voltage V is $E = CV^2/2$, so the energy required to execute n instructions can be expressed as

$$\Delta E = E_f - E_0 = C(V_f^2 - V_0^2)/2, \qquad (2.3)$$

from which the per-operation energy is straightforward to derive.

Table 2.3 illustrates the results of measuring per-operation energy for the MSP430 memory hierarchy on a WISP's (Revision 1) MSP430F1232 microcontroller, with flash writes requiring disproportionately large amounts of energy. Note that *reading* from flash memory is comparable to reading from RAM.

Instr.	Dest.	Src.	Energy/Instr. (nJ)	Perc. Error
NOP	—	—	2.0	4%
MOV	reg	reg	1.1	23%
		flash	5.2	17%
		mem	6.3	33%
MOV	mem	reg	8.1	13%
		flash	11.8	4%
		mem	11.7	7%
MOV	flash	reg	461.0	4%
		flash	350.3	1%
		mem	1126.2	4%

Table 2.3. Energy required per instruction varies on the TI MSP430F1232. Each figure is the average of 5 measurements (smallest and largest discarded) on a WISP (Rev. 1).

Additionally, we observed that the energy consumption of a flash memory write on the MSP430 is not data dependent. For each of four values containing different
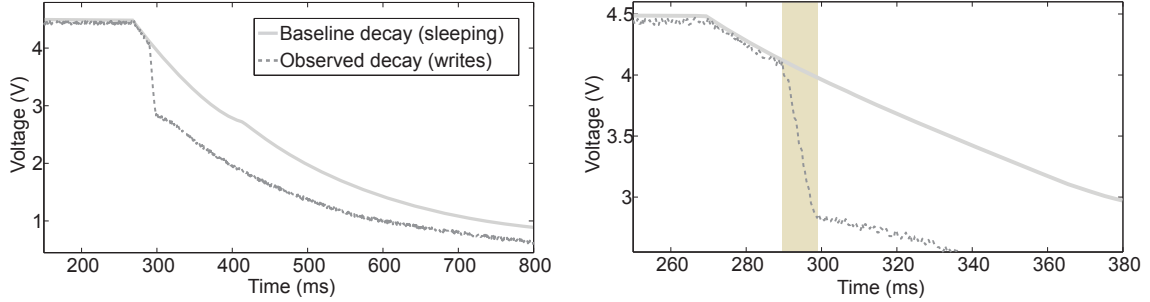
Figure 2.6. Measuring instruction energy for 64 flash writes, taking leakage into account. The right figure is a detailed view of the left figure.

numbers of 0 bits, we measured the total energy consumption of writing the value to five consecutive words of flash memory (averaged over five runs). We observed that, for example, the energy costs of writing an all-0 value and an all-1 value were indistinguishable within the error bounds.

2.4.2 Capacitor Leakage and Quiescent Current

In an idealized CRFID, the microcontroller's active current consumption—plus that of any peripherals it has explicitly powered on—would account for all of the energy use from the capacitor. However, real capacitors leak energy, and real circuit components, such as the WISP's voltage supervisor, often impose a nonzero quiescent current that extracts energy from the supply. Fine-grained energy measurements must therefore take these omnipresent factors into account.

Capacitors leak charge at a rate that depends on their chemistry, the composition of the circuits connected to them, their age, and their usage patterns. Electronic components that play a supervisory or monitoring role, e.g., those that must trigger interrupts when certain conditions occur, generally draw current at all times. For simplicity, since the WISP components' quiescent current is approximately constant, we account for both capacitor leakage and quiescent current with the umbrella term "leakage."

21

We performed experiments to measure leakage on WISP hardware's ceramic storage capacitors. In subsequent experiments, we used these leakage measurements to discount the effect of leakage when measuring the energy consumption of operations.

To measure leakage, we charged a WISP's storage capacitor to 4.5 V using an external power supply, then removed the power supply and measured the capacitor's voltage as the microcontroller slept in its lowest-power RAM retention mode. For example, on a WISP (Revision 1.0) with a $10\,\mu$F capacitor, the voltage of the capacitor fell from $4.5\,$V to $2.7\,$V (this model of WISP's minimum operating voltage, comfortably above the 2.2 V threshold for flash writes and the 1.8 V threshold for microcontroller operation) in 700 ms, after which the WISP's voltage supervisor cut power to the microcontroller and the decay slowed. Figure 2.6 depicts a measurement of 64 flash writes. We compared the voltage drop for the set of flash writes (dotted line) against a baseline voltage drop (solid line). The measurements in Table 2.3 discount for leakage in this way.

2.4.3   Current Measurements

Per-instruction microbenchmarks (Section 2.4.1) yield energy-consumption figures that are useful to illustrate the relative costs of single memory operations. To measure the energy consumption of longer workloads—and to validate the MSP430 datasheets' approach of giving current-consumption figures for power modes rather than instructions—we measured the current consumption in each power mode using a multimeter.

We wrote a sequence of programs that exercised each power mode of the MSP430: active, flash write, analog-to-digital converter (ADC) sampling, and five low-power modes LPM0, LPM1, LPM2, LPM3, and LPM4. The active-mode test case consists of a single, endlessly repeated jump to the entry to `main()`. The ADC and flash test cases consist of endless loops in which the microcontroller reads from its onboard
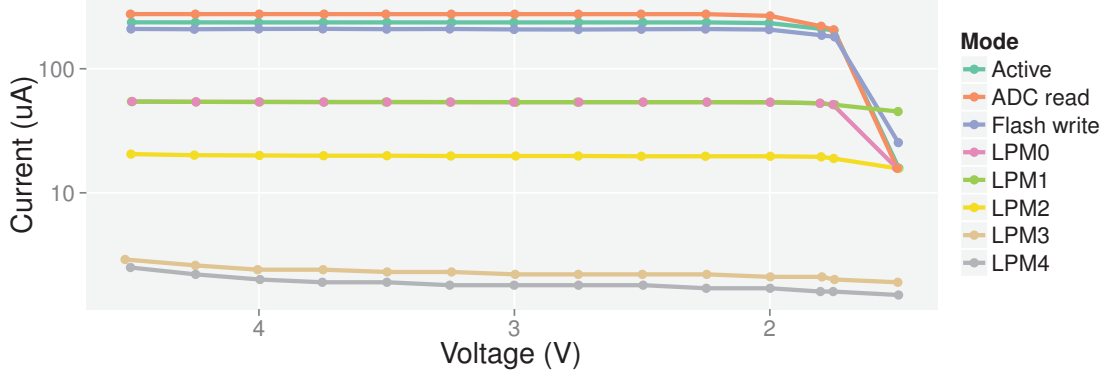
Figure 2.7. Current consumption versus voltage for the MSP430F2132 microcontroller on a DL WISP 4.1 [138] CRFID in distinct power modes. We model each of these modes in our simulation environment.

ADC or writes to a portion of onboard flash, respectively. (To minimize loop overhead for the ADC and flash test cases, we verified that the compiler generated jumps to immediate addresses instead of conditional branches.) The low-power-mode test cases simply enter the appropriate low-power mode and terminate.

Our modified MSPsim incorporates these empirical current-consumption measurements, which are also depicted in Figure 2.7. Each operation that changes the power mode sets the simulated circuit's resistance (via $R = V/I$, for use in Equation 2.1 and Equation 2.2).

2.5   Related Work

Computer architects and compiler writers have identified power consumption as a factor that limits scalability. Some simulation tools take this factor into account by including a notion of per-logical-block power consumption. Power-aware simulators built on SimpleScalar [26, 8], such as Wattch [20], and other simulators aimed at identifying power hotspots [34, 33] model power consumption in terms of the subset of chip components that are active at the time an operation is executed. This

approach maps cleanly to the approach we take in the modified MSPsim, in which certain operations can cause the CPU to switch into higher- or lower-power modes. However, it maintains the abstraction of a power supply that cannot run out of energy, and therefore does not include a feedback loop that affects program execution. We took the approach of modifying an MSP430 simulator (MSPsim [49]), rather than adding a simulated capacitor to a SimpleScalar-based simulator, for the prosaic reason that SimpleScalar (and its derivatives) does not include an MSP430 target. Other researchers have adopted MSPsim for its simplicity and simulation accuracy [107].

PowerTimer [19], a simulator that targets the PowerPC architecture, follows an approach similar to Wattch: the simulator keeps track of which processor units are active and adds power consumption accordingly. Via measurements obtained with PowerTimer, its authors argue for architectural support for turning off components when possible. Work on modeling OS power consumption [93] based on the SoftWatt full-system simulator [64] has revealed correlations between architectural metrics such as instructions per cycle (IPC) and power consumption when profiling operating-system energy consumption. Like the SimpleScalar-based simulators mentioned above but unlike our modified MSPsim, these systems do not model the power supply as a potentially limited resource that falls short and causes system resets.

HotSpot [147, 75] brings the power simulations of Wattch to the thermal domain—another important concern for modern architecture—solving for thermal properties per block at each time step of its input. HotSpot takes a Wattch power trace and a chip floorplan and produces a temperature trace that helps designers anticipate heat dissipation. Like other SimpleScalar- or Wattch-based profiling mechanisms, HotSpot processes traces *post facto* instead of providing a real-time feedback loop to a running simulation.

The CRFID Crash Test Simulator (CCTS) [62] is designed to evaluate the suitability of solar power to augment energy harvesting on CRFIDs. To model a solar-

harvesting supply, CCTS takes a data set describing a solar panel's voltage–current (V–I) curve. It accepts two input traces: a sequence of illuminance values representing the light available to the solar panel over time, and a trace of a CRFID's overall power consumption as a program executes. With this representation of computation, CCTS enables high-level reasoning about solar harvesting but does not support the execution of arbitrary firmware.

Mindful of nondeterminism in real workloads that comes from timing variations—run-time nondeterminism that traditional simulations ignore—some simulation methodologies employ random perturbations of timing parameters in order to more accurately simulate variable conditions under deployment [3, 2]. These improved methods allow for statistical conclusions that may be more accurate than conclusions drawn from completely deterministic runs that depend only on the experimenter's choice of inputs. In contrast, our trace-driven TPC simulation incorporates unpredictability (from the CPU's and program's perspective) as a crucial property of the simulation, for the purpose of straightforward, reproducible simulation rather than statistical validity under random perturbations.

Ekho [176] is a promising energy-trace recording and playback platform meant to enable reproducible runs of energy-harvesting devices like CRFIDs. It is premature for our application in its current prototype form, with low capturing resolution and not-yet-validated V–I curve models for RF harvesting, but a future incarnation may provide a simulation-free way to debug TPCs in a predictable environment.

2.6   Summary

Reproducible results are difficult to obtain under energy harvesting because physical inputs—which may completely determine run-time performance—vary from run to run. Our simulation environment for TPCs, based on the MSPsim cycle-accurate simulator for MSP430 microcontrollers, addresses this challenge by running firmware

programs against empirically obtained energy traces. This simulation fits conceptually between analog circuit simulation (e.g., SPICE) and digital architectural simulation (e.g., Simics) because runs incorporate inputs from both the analog and digital domains.

CHAPTER 3

COMPUTING UNDER TRANSIENT POWER[1]

You build on failure. You use it as a stepping stone. Close the door on the past. You don't try to forget the mistakes, but you don't dwell on it. You don't let it have any of your energy, or any of your time, or any of your space.
—Johnny Cash [29]

By definition, transiently powered computers do not receive a constant supply of operating power. When power availability is limited, or when a program's power consumption outpaces energy harvesting, TPCs may lose power before completing their assigned tasks. The constant threat of power loss is a key limitation.

This chapter explores the power-related challenges inherent to computation under intermittent power, in particular under energy provided by naturally unsteady RF harvesting. Considering the primary challenge of dealing with frequent losses of volatile state caused by power losses, this chapter describes Mementos, a system that protects computations on TPCs from state loss by way of an automatic, energy-aware state-checkpointing system. Mementos monitors available energy at run time after instrumenting a program with energy checks at compile time. The key contribution of Mementos is that it enables the completion of computations that require more resources than are available during a given burst of power (a *lifecycle*).

---

[1]This chapter extends three papers by Ransford et al. [122, 123, 124].

3.1 Introduction

Batteries, the traditional source of operating power for embedded computing devices, impose serious limitations on the size and deployability of these devices. Unlike microchips, which have continually improved in a way that is consistent with Moore's Law, batteries obey no such law [110], resulting in devices whose computing components are minuscule—in both size and weight—in comparison to their battery packs.

The desire to sidestep batteries' limitations has driven the development of general-purpose *transiently powered computers* that operate solely on harvested energy. Such devices range from *computational RFIDs* [122]—microcontroller-based devices that harvest RF from readers and communicate via RFID protocols—to general-purpose batteryless sensor devices [169].

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.

The status quo of frequent power loss on transiently powered devices hinders developers in two ways. First, there are certain operations that are simply too time consuming to complete in the typical amount of time a device may run. For example, a smart card is unlikely to receive power for more than a few hundred milliseconds, which is not enough time for its low-power microcontroller to complete an RSA
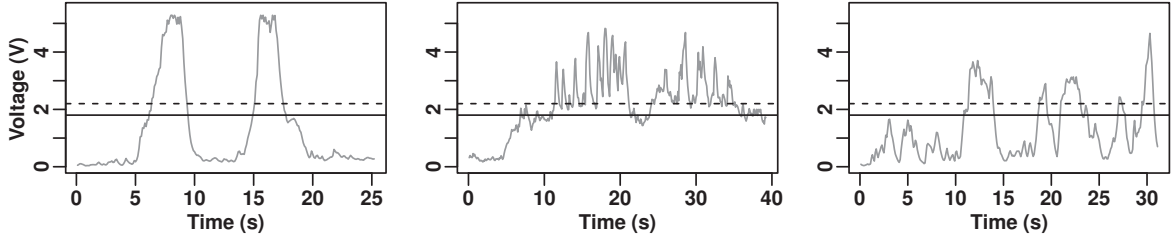
Figure 3.1. Energy availability under RF harvesting is difficult to predict on a transiently powered computer (TPC), threatening the successful completion of long-running programs. These plots show the output of a prototype TPC's energy-harvesting frontend during three different smooth movements within 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.

public-key decryption with a long key [132]. Second, developers must pessimistically and painstakingly hand-tune programs that run CPU-intensive operations like cryptography or sensor data processing on transient power, so that these programs will complete within a short time window appropriate to their physical use case, often under 100 ms [24, 31]. (Section 3.3 characterizes power loss for a CRFID in detail.)

If developers can confidently write programs for TPCs while being *protected* from the repercussions of frequent power loss, they can write more-sophisticated, longer-running programs. *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.[2] 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, Mementos restores program state and execution continues rather than restarting from scratch.

This chapter makes the following contributions:

_____

[2]In the 2000 film *Memento* [76], the main character would unpredictably lose short-term memory, especially when sleeping. He checkpointed state with notes and tattoos in an attempt to execute a single (all-consuming) long-running task.

- 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.

- A suite of compile-time optimization passes that insert energy checks at control points in a program. The optimization passes implement three different instrumentation strategies that are designed to work well on different program structures. Mementos provides a test harness to help developers choose appropriate instrumentation strategies for their programs.

- A set of enhancements to the basic Mementos strategy that add *adaptive* state checkpointing, allowing Mementos to adjust its run-time parameters to improve its use of resources.

The compile-time analysis and program transformation components of Mementos are built on the LLVM compiler infrastructure [88]. We evaluate the performance of Mementos using a modified version of the MSPsim cycle-accurate MSP430 simulator [49] (described in Chapter 2). This simulator accepts empirically recorded energy-harvesting traces and MSP430 executables and runs programs under simulated energy-harvesting conditions.

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 [108] 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.

3.2 Background and Challenges

Multiple platforms at various stages of maturity enable batteryless, RFID-scale, transiently powered computing. The WISP [138] uses an MSP430 microcontroller [155] for computation and harvests energy from off-the-shelf RFID readers. A system-on-chip variant, the SoCWISP, is small (2.0 mm^2) and light enough to attach to insects in flight [171]. The Blue Devil WISP improves upon the WISP's RF-harvesting and communication performance [160]. The UMass Moo provides more memory and storage than the WISP, improves upon its sensing precision, and supports a greater number of peripherals [175]. Solar-harvesting "leaves" aim to combine the properties of RFID tags and sensor motes, with solar harvesting powering active radios to form multi-hop networks [173]. The EnHANTs platform also situates solar-harvesting nodes between RFID tags and motes, targeting flexible sensor tags that could be embedded in objects [59]. All share the goal of enabling sensing and general-purpose computation under harvested power.

Previously proposed applications for transiently powered computers include environmental monitoring [62], activity recognition [25], and cryptographic protocols [31]. Mote-class devices (e.g. Telos [117], TinyNode [45]) 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.

3.2.1 Challenges

Despite their benefits, designing and deploying these systems is challenging. A first challenge is that of limited, unreliable power. By definition, TPCs cannot depend on continuous power. Figure 3.1 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 [110, 172]—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 ∼100 ms are a reasonable expectation [31, 122]. Existing lightweight operating systems like TinyOS [89] 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*.[3]

The second challenge is that of disappearing volatile state. With each loss of power on a TPC, some or all of the volatile state in the CPU disappears, with each unit of memory probabilistically decaying to a ground state in some amount of time [137].

A key to addressing the problem of Sisyphean tasks on TPCs is that many general-purpose microcontrollers, notably the MSP430 found on WISP-derived devices, fea-

---

[3]In Greek mythology, Sisyphus was the first king of Corinth and a conniving malefactor [73]. 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.

ture 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.

But a third challenge—constraints on nonvolatile memory—makes it nontrivial to use flash memory for checkpoint storage, for the following reasons:

1. 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.

2. 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.

3. 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.

4. Many microcontrollers use flash memory for program storage, which limits the amount of nonvolatile storage available for other purposes.

As mentioned in Section 3.1, Mementos uses energy-aware state checkpointing to insulate programs from continual power losses. It stores state checkpoints in on-chip flash memory. Despite the above caveats, Mementos can use flash as a nonvolatile backing store for volatile state by judiciously copying data—minimizing the frequency and duration of flash writes—and by carefully managing bundles of checkpointed state to minimize the risk of corruption from power failures. Mementos incorporates a flash-management strategy that reserves two segments of flash memory, erasing a segment only when it no longer contains the most recent state checkpoint. To minimize flash writes, Mementos avoids checkpointing until the energy supply drops below a threshold level.

3.3 Power Loss and CRFIDs

To quantify the frequency of power loss events on a prototype TPC, we performed an experiment using UMass Moo CRFIDs [175] harvesting energy from a commercial Impinj Speedway Revolution RFID reader.

We connected a Tektronix MSO5204 oscilloscope between a general-purpose input/output (GPIO) pin on the Moo's microcontroller and the Moo's ground pin. This setup allowed us to observe the state of the digital output on the pin. Setting the GPIO "high" and then "low" in the Moo's firmware, with a 0.1 ms delay in between, resulted in a visible spike on the oscilloscope. Raising and lowering a pin for this amount of time draws a negligible amount of the Moo's capacitor's energy, so we used these visible spikes to signify boot events as we tested various workloads. To calculate the frequency of boot events in an oscilloscope trace, we saved a 5-second period of time-series data to a text file, plotted the text file, counted the spikes, and divided by 5 to yield an average number of boot events per second over the 5-second period.

We tested four workloads:

- A 100% duty-cycle workload that executes an infinite loop after spiking the GPIO pin.

- A 75% duty-cycle workload that executes an infinite loop for 75% of the time after spiking the GPIO pin. For the remaining 25% of the time, the microcontroller sleeps in a lower-power mode (MSP430's LPM0) with a timer enabled.

- A 50% duty-cycle workload, similar in all other respects to the 75% duty-cycle workload.

- A 25% duty-cycle workload, similar in all other respects to the 75% and 50% duty-cycle workloads.
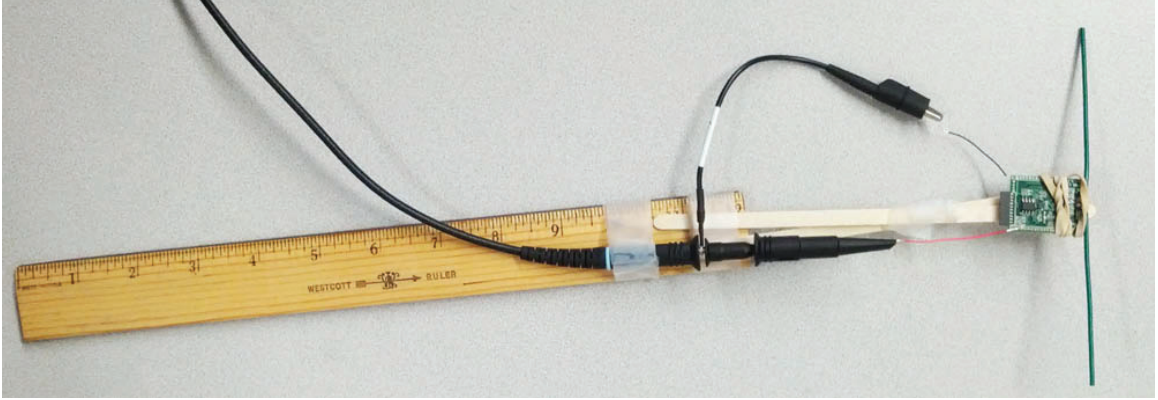
Figure 3.2. Test harness for testing the frequency of reboots on a UMass Moo. An oscilloscope probe observes the voltage of a pin that the Moo raises and lowers on each boot.

We set up the RFID reader with its panel antenna (a Cushcraft S9028PCL panel antenna) pointing straight down an empty hallway ~ 2.2 m wide. We marked points at 25 cm intervals up to 3 m.

For each workload, we programmed the Moo with the workload, attached the Moo to a non-conductive test harness to hold oscilloscope probes (Figure 3.2), and placed the Moo at each distance marker with its dipole antenna parallel to the plane of the reader's antenna (the optimal orientation). We then recorded data as described above.

3.3.1 Experimental Results

The metric we used to measure the frequency of power losses was the number of boots events per second over a 5-second window. Boot events per second, as a metric, is an upper bound on the number of destructive losses of volatile state a workload experiences. This is because the workload may finish before it loses power, or data remanence in RAM may preserve some of the volatile state. In the case of our infinite-loop workloads, boots per second is the same as reboots per second.
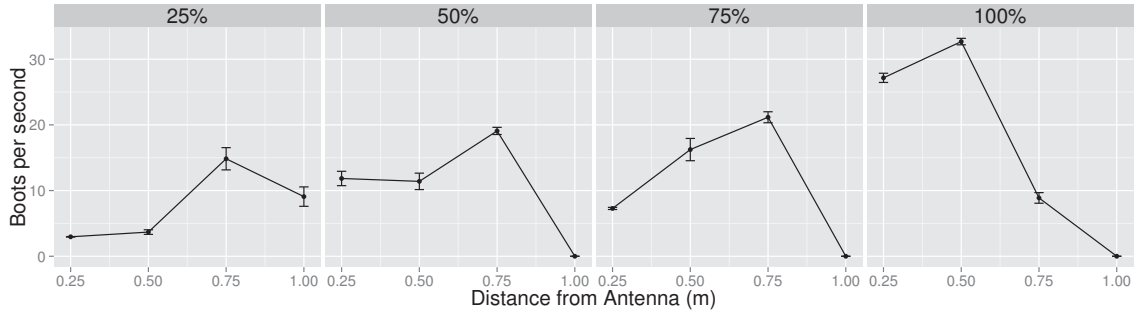
Figure 3.3. Boots per second for a UMass Moo for all four test workloads (25% duty cycle to 100% duty cycle).

Figure 3.3 summarizes our experimental results for all four workloads. Each workload exhibited increasing boots per second as we moved the Moo farther away from the antenna, up to a certain distance; the boot frequency decreased as we moved the Moo past this distance. Before this inflection point, the workload faced power losses during execution; after this inflection point, the Moo faced the same power losses during execution *and* increasingly could not boot at all because it could not harvest enough energy.

Only the least-strenuous workload—the 25% duty cycle workload—booted at all at 1 m during the 5-second measurement windows, because it had more time (the 75% of its duty cycle that it slept) to harvest energy in between boots. Figure 3.4 shows the effect of varying the distance under this workload. Near the antenna (at 25 cm and 50 cm), the Moo booted with regularity, indicating that it completed each 25% portion of its workload, then lost power during the remaining 75% of its duty cycle. Farther from the antenna (at 75 cm), the Moo booted more frequently and with less regularity, indicating that it was interrupted by power losses while waiting to begin its next duty cycle. At the greatest distance at which it booted at all (100 cm), the Moo booted infrequently and with no discernible regularity, indicating that it needed a long charge time between boots *and* lost power before completing its duty cycles.
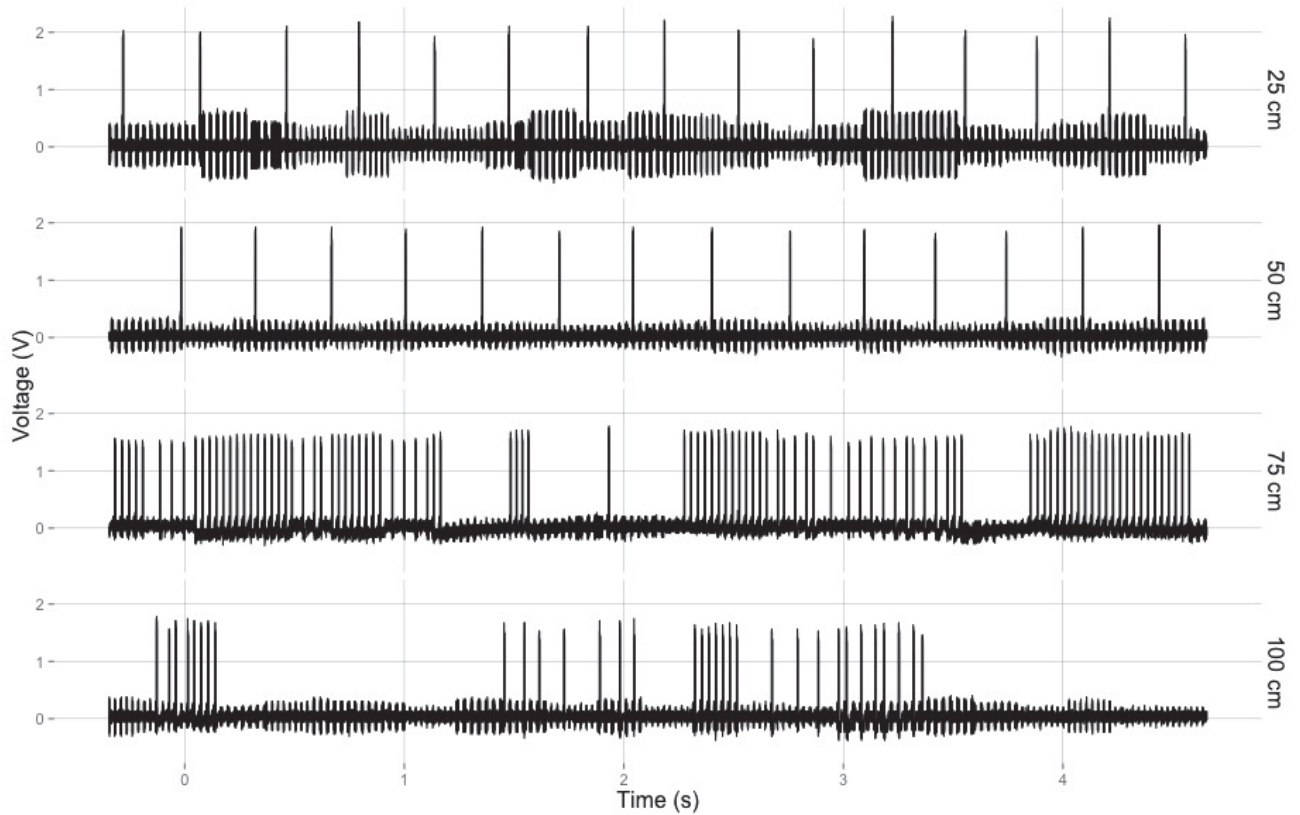
Figure 3.4. Boot events for a single workload (25% duty cycle) as we vary the UMass Moo CRFID's distance from an RFID reader.

A second Moo programmed with the same workloads produced results that were visually indistinguishable from these results, so we omit them here for brevity.

This experiment provides evidence that power failures occur multiple times per second on a UMass Moo CRFID. These results may not generalize to other devices or harvesting modalities, for several reasons:

- Antenna shape, orientation, and efficiency vary from CRFID to CRFID. We used hand-tuned (manually impedance-matched) Moos to maximize RF-harvesting efficiency; other prototypes (e.g., the WISP) may exhibit different patterns of power loss because their harvesting circuits differ.

- Environmental factors affect harvesting efficiency; this is true for RF harvesting and other modalities (e.g., solar). For a CRFID, the RF supply is subject to multipath interference (signal bouncing), the presence of occluding bodies, and physical properties of the RFID reader. Changing these variables may increase or decrease the frequency of power losses.

3.4 Design of Mementos

The key observation motivating the design of Mementos is that, in general, 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 3.1) 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.

Goal: Split programs across multiple lifecycles. Mementos must, at run time, automatically suspend and resume programs without user intervention, so that a

program that runs in n cycles with unlimited energy may run in $n' \geq n$ cycles when interrupted by power losses.

Design 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 batteries—for example, many MSP430 variants include an internal measurement channel dedicated to monitoring supply voltage.

Design principle #2: Reason minimally about energy at compile time, maximally at run time. Past work has demonstrated that even expert programmers have trouble reasoning about run-time energy [149]. In fact, reasoning about run-time energy availability before run time may be impossible because of inconsistent harvesting, and accurately predicting energy availability at run time may be infeasible because of the limited computational resources available for prediction. To sidestep these difficult problems, Mementos uses a simple metric—the voltage on the supply capacitor— to estimate available energy at run time. It inserts voltage checks at compile time, obviating the need for complex logic to deal with changing energy conditions.

3.4.1 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 a shim function that searches for a restorable state checkpoint and resumes computation if it finds one.

Where should Mementos place trigger points? To ensure that it suspends execution with enough time for a checkpoint to complete, Mementos should insert enough trigger points at compile time so that it can effectively sample run-time energy trends (increasing or decreasing voltage); however, it should not insert so many that mea-

Term	Definition
Checkpoint	A copy of program state information from which execution may be restored after a reboot.
Trigger point	A check of available energy that may cause a checkpoint.
Checkpoint threshold voltage	The voltage below which Mementos turns trigger points into checkpoints.
Sisyphean task	A task that exhausts the platform's available resources each time it runs, without finishing.
Loop-latch mode	Mode in which Mementos places energy checks at loop latches.
Function-return mode	Mode in which Mementos places energy checks after call instructions.
Timer-aided mode	Mode in which Mementos performs energy checks only when a hardware timer has raised a flag.
Lifecycle	(or power lifecycle) Time during which a transiently powered device can execute code. Mementos splits computations across multiple lifecycles.

Table 3.1. Terms used in our discussion of Mementos.

surement cost predominates over execution. To satisfy our goal of supporting a wider range of TPC 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, recurring control structures—namely loops and function calls—and to execute energy checks according to a preconfigured timer.

- 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 predeter-

mined 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 these 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 calls to Mementos functions in her program. In the same manner, she can similarly insert code to force checkpointing at a certain point in the program.

3.4.2 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 TPCs 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 and 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}) represented as a threshold ADC value. 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. (In practice, this assumption may result in underestimates of available energy because the harvester may still be working even during checkpointing.) 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 $\Delta t = C(V_{\max} - V_{\min})/I$. 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 3.5 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 non-volatile 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
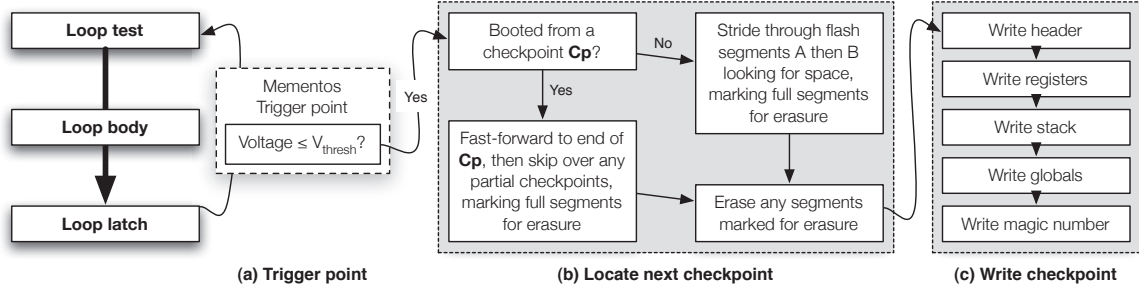
Figure 3.5. Overview of run-time checkpointing in Mementos. This diagram depicts the *loop-latch mode* in which Mementos instruments loop back-edges with energy checks that conditionally trigger checkpointing.

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.

3.4.3 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 3.5). 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 3.2 and below) complicate checkpoint management.

Most checkpointing systems are designed to run on multiprogrammed operating systems (e.g. the MPI checkpointing of Bronevetsky et al. [18]) or in hardware environments that support the issuance of commands by other devices (e.g. the sen-

sornet checkpointing of Österlind et al. [109]). 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 (and executed from) memory-mapped 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 [116]. 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

Stack size, S	Globals size, G	Contents of registers	Contents of stack	Contents of globals	End marker
1 byte	1 byte	15 x 2 = 30 bytes	S bytes	G bytes	2 bytes

30 + S + G bytes
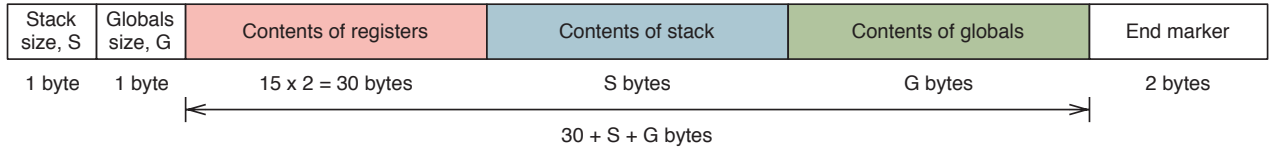
Figure 3.6. A state checkpoint in Mementos. The length of the entire checkpoint can be calculated from the two-byte header.

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. Figure 3.6 depicts a checkpoint as it is stored.

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 pose a design challenge. 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`[4] 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.

Unlike prior systems that rely on OS facilities to simply dump process memory to a file system (e.g. *libckpt* [115]), 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 successfully 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 erased 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

---

[4]Mnemonic: "Mementos, no trigger points!"

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).

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 3.6 works).

3.5   Implementation

Mementos formulates its program transformations as LLVM [88] 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 (at the time of this writing) a total of 912 lines of code including whitespace, comments and header files. Mementos's run-time library comprises an additional 843 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 3.6 details our use of this simulator for evaluation.

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. We originally considered static analysis to count post–code-generation instructions and simply tag basic blocks with energy estimates [122], 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.

3.6   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.

3.6.1 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 [138], 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 [49], 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.[5] 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).

Chapter 2 describes our simulation efforts in detail.

3.6.1.1 Test cases

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

---

[5]We refer the reader to Horowitz and Hill [74] for a detailed discussion of capacitor behavior.

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 3.4), 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 3.6.1.3).

3.6.1.2 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 3.2.

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 3.2. Metrics for evaluation.

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 3.3 gives results for all three test cases. For the instrumented variants, 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.

Instrumentation Strategy	crc / %M	sense / %M	rsa64 / %¡
Uninstrumented	575,315 / —	157,635 / —	70,218 / —
Loop latches	619,450 / 6.9	284,795 / 44.3	303,250 / 76.2
Function 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 3.3. 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, when its instrumentation runs but never results in a state checkpoint to nonvolatile memory.

3.6.1.3 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 3.4), 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 3.4 shows results obtained in oracle mode under *variable voltage* (rather than the fixed high voltage of Table 3.3). The cycle counts and lifecycle counts in the table should be considered near-lower bounds[6] on those metrics for instrumented versions.

3.6.1.4 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

---

[6]These lifecycle counts are from a version of the simulator that counted cycles correctly but occasionally repeated lifecycles unnecessarily; they are therefore not hard lower bounds.

	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 3.4. 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 $24.5\pm1.3\%$; for `sense`, $49.1\pm11.0\%$; for `rsa64`, $56.6\pm5.6\%$.

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 3.4), 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
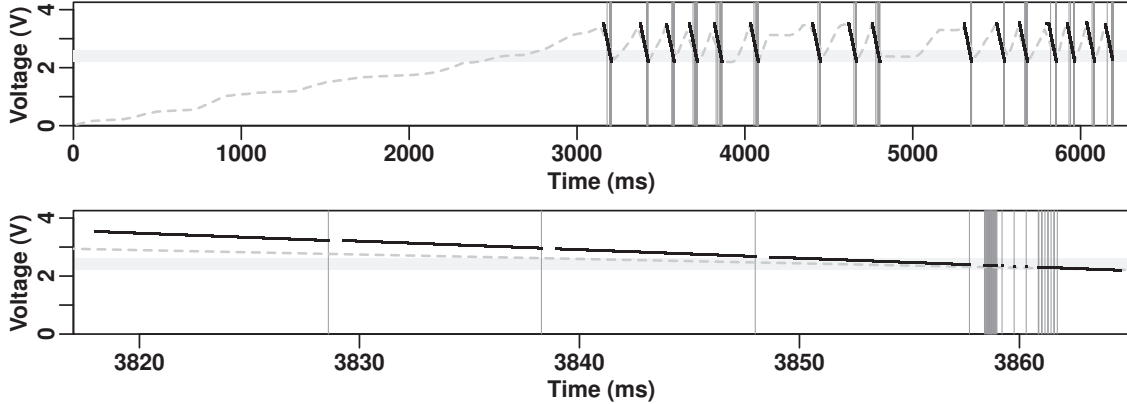
Figure 3.7. Simulated voltage versus time as Mementos spreads the execution of the `crc` test case across 17 power lifecycles (16 resets) against a voltage trace (#9). The bottom plot highlights a single power lifecycle from the top plot.

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

Figure 3.7 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 3.5 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 repre-

55

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)
3.0	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 3.5. 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_{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.

sentative 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 3.5 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 3.5 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 3.5 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 $V_{\mathrm{thresh}} = 2.6$ 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 $V_{\mathrm{thresh}} =$ 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 $V_{\mathrm{thresh}} = 3.0$ 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 3.5 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.

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.

3.6.2   Mementos on Hardware

Our evaluation of Mementos focuses on simulation for several reasons, summarized below and described in more detail in Chapter 2:

- Simulation enables reproducible runs;

- Simulation gives full visibility into machine state, something the programmers of larger computers may take for granted; and

- Simulation allows experimentation with different machine parameters, to avoid "overfitting" the system to a specific hardware platform.

We have verified that Mementos works (i.e., spreads computations over multiple lifecycles) on a variety of hardware, including standalone experimenter boards such as the Olimex MSP-H1232 and computational RFIDs such as the DL WISP (revision 4.1) [138] and the UMass Moo [175] (revision 1.1). For some devices, we had to make small hardware changes to make Mementos work. On the WISP, for example, 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, but the power consumption of the 2.8 V regulator came to

dominate the WISP's overall power consumption at run time. For the Moo, we used an off-chip flash accessible via a serial bus, at greater energy cost than writing to on-chip flash. Future TPCs may benefit from designing with checkpointing to on-chip nonvolatile memory in mind.

To support future explorations of Mementos on hardware, we added to Mementos a simple logging mechanism that uses a set of general-purpose I/O pins to convey bit values; toggling these pins at run time allows for coarse-grained observation of Mementos's activity.

3.6.3 Extension: Adaptive Checkpointing

As the rest of this section illustrates in detail, compile-time tuning of Mementos's parameters can significantly change its behavior. We have designed and implemented a scheme by which Mementos can adapt its behavior at run time based on its measurement of certain metrics.

In the non-adaptive version of Mementos, there are two parameters that the programmer must choose. One is the voltage threshold value V_{thresh} and the other is the instrumentation strategy. Since the appropriate instrumentation strategy is unlikely to change at run time for a given program, this technique for making Mementos adaptive focuses on adjusting the value of the V_{thresh} value at run time.

3.6.3.1 Minimizing Failed Checkpoints

As mentioned earlier in this chapter, Mementos does not append state checkpoints to failed checkpoints in its reserved memory segments. This strategy helps it avoid mistaking partially checkpointed state for valid checkpoints, but because it invalidates the flash segment for further writes, it can result in excessive flash erasures if failed checkpoints are commonplace.

Stack size, S	Globals size, G	Generation #	V_{thresh} value	Contents of registers	Contents of stack	Contents of globals	End marker
1 byte	1 byte	**2 bytes**	**2 bytes**	15 x 2 = 30 bytes	S bytes	G bytes	2 bytes

30 + S + G bytes
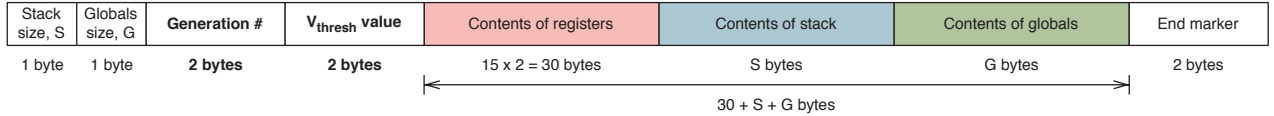
Figure 3.8. A state checkpoint in the *adaptive* version of Mementos. As in the non-adaptive version, the length of the entire checkpoint can be calculated from the two-byte header. Mementos uses an additional four bytes of metadata (shown in bold) to decide whether to adjust its checkpointing voltage threshold V_{thresh} up or down.

To avoid executing time- and energy-intensive flash erasures at the beginning of lifecycles, an extension to Mementos includes two pieces of extra metadata in each checkpoint header (depicted in Figure 3.8):

- The current value of V_{thresh} at the time the checkpoint was saved; and

- A monotonically increasing *generation number* that changes once per lifecycle.

Accordingly, instead of using a threshold the programmer must choose, this version of Mementos chooses a starting value for V_{thresh} and adjusts it up or down as necessary.

The adaptive version of Mementos uses this metadata at run time as follows. When walking its reserved flash segments at boot time in search of a restorable checkpoint at boot, Mementos notes the generation number in each checkpoint number, then chooses a new generation number that is one greater than the maximum generation number it found. New checkpoints written in the current lifecycle will therefore be distinguishable from checkpoints written in previous lifecycles. Also during its search for a restorable checkpoint, it decides whether to adjust V_{thresh} up or down:

- If multiple checkpoints are from the *same* previous lifecycle, as indicated by identical generation numbers, there is an opportunity to start checkpointing later in the current lifecycle. Mementos notes the V_{thresh} value stored in those checkpoints and lowers it by a small amount (0.1 volts) for the current lifecycle.

60

- If there is an *incomplete* checkpoint that is the only checkpoint from its lifecycle—assuming the incomplete checkpoint at least contains the generation-number and V_{thresh} portion of its header, which is written before any volatile state—then there is an opportunity to start checkpointing earlier in the current lifecycle, to give the checkpointing operation more time to complete. Mementos notes the V_{thresh} value stored in the incomplete checkpoint and raises it by a small amount (0.1 volts) for the current lifecycle.

Both of these strategies implicitly assume that information about one lifecycle is relevant to later lifecycles. This may be true if the signal powering the TPC is periodic or if energy availability is trending in one direction (e.g., a CRFID is moving away from a reader).

3.6.3.2   Results

These results highlight two test cases to evaluate adaptive checkpointing against the results in Section 3.6.1.4.

1. The `sense` test case exhibited the greatest fraction of wasted cycles and failed to complete at all under many fixed choices of V_{thresh}. We use this test case to evaluate the amount by which adaptive checkpointing can reduce the amount of wasted work, and its ability to compensate for ill-suited values of V_{thresh}.

2. The `crc` test case was the longest running of the three test cases, so we use this test case to evaluate the performance of adaptive checkpointing against the best-case performance with a checkpointing oracle that always checkpoints at the last practicable moment.

Table 3.5 in Section 3.6.1.4 gave waste and overhead figures for the `sense` test case against a voltage trace (#9 of our 10 traces). Table 3.6 evaluates the same test case under the voltage-threshold adjustment scheme described in this section.

(For simplicity of explanation, we focus on the loop-latch instrumented version of the `sense` test case.) The results suggest improvements in two key metrics. Under non-adaptive checkpointing, the `sense` test case exhibits wildly different performance against different values of V_{thresh}. Under adaptive checkpointing, we used the same values of V_{thresh} as initial values and allowed the run-time adaptations to adjust this threshold up and down. The number of lifecycles, number of CPU cycles, and fraction of wasted cycles decreased over the range of V_{thresh} values, so much so that the overhead of Mementos dominated the total number of CPU cycles to complete the workload. (The `sense` test case is structured as many iterations of a short loop.)

V_{thresh}	Non-adaptive	Adaptive
	Lifecycles (Cycles, % waste, % Mementos)	
2.4 V	—	3 (295,573 / 3.85 / 78.14)
2.6	89 (5,235,399 / 89.4 / 64.2)	5 (694,795 / 8.39 / 84.75)
2.8	13 (708,641 / 10.5 / 71.3)	4 (460,458 / 0.40 / 87.52)
3.0	46 (2,852,111 / 20.4 / 81.6)	3 (218,031 / 6.47 / 87.54)
3.2	—	3 (248,497 / 1.27 / 89.48)
3.4	—	3 (281,456 / 8.10 / 90.94)

Table 3.6. For the `sense` test case with loop-latch instrumentation, against the same voltage trace used for Table 3.5, adaptive checkpointing reduces the fraction of wasted cycles and the number of lifecycles. For the *Adaptive* runs, the V_{thresh} values in the left column were initial values rather than fixed values.

Table 3.4 in Section 3.6.1.2 evaluates the run time of the `crc` test case under a *checkpointing oracle* that finds the last practicable time to checkpoint in each lifecycle. Under adaptive checkpointing, the `crc` test case completed within roughly a factor of two of the oracle's predicted best-case performance in the worst case, and within 8% in the best case.

Energy prediction for adaptive checkpointing is an open problem. Any adaptive checkpointing mechanism that does not explicitly predict future energy availability is left to make decisions based on its historical performance—i.e., checkpoints and meta-data from current and previous lifecycles. An implicit assumption of this approach is

	Oracle Prediction	**Adaptive Checkpointing**
Trace	*Cycles (lifecycles)*	
Decay-only	621,501 (8)	669,909 (3)
#1	685,555 (15)	1,372,900 (10)
#2	685,150 (15)	1,382,685 (10)
#3	685,801 (15)	718,898 (6)
#4	685,641 (15)	1,391,161 (10)
#5	685,096 (15)	728,264 (6)
#6	685,099 (15)	725,866 (6)
#7	683,884 (15)	1,397,314 (10)
#8	685,005 (15)	732,225 (6)
#9	685,608 (15)	1,391,328 (10)
#10	685,045 (15)	1,382,879 (10)

Table 3.7. Comparison of oracle-mode predictions and adaptive-checkpointing performance for the `crc` test case. (Note that the lifecycle counts in the left column may be overestimates, as mentioned in Section 3.6.1.3.)

that future energy harvesting will resemble past and present energy harvesting. If it does not, then the adaptations may not be helpful.

Fortunately, some energy-harvesting modalities *do* offer reliable, predictable power. At close range, even RF energy can be predictable in a static environment after applying a low-pass filter to remove unpredictable high-frequency noise.

We have not designed a run-time energy-prediction model for Mementos 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.

3.7   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.

3.7.1 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 [9, 104]. 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 allow TPCs to outsource long-running computations to a more powerful device (e.g. a Linux-based RFID reader), rather than attempt to execute them locally on the TPC. While this approach removes many impediments, it imposes others. For example, to save power, CRFIDs use low-throughput radio communication that could make outsourcing time intensive. In security-sensitive applications, outsourcing cryptographic operations may violate security requirements, and existing techniques for outsourcing computation to untrusted infrastructure [57] still require a nontrivial amount of work on the client. Harvesting modalities that exploit environmental phenomena, such as solar or heat, may not have access to any other devices.

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.

3.7.2 Future Hardware

Moore's Law does not have an analogue for batteries [110], 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 [21] might extend a system's lifetime; however, alternatives like phase-change memory (PCM), magneto-resistive RAM (MRAM) and ferroelectric 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.

A factor that complicates Mementos's use of flash memory is that segment erasure, an operation that Mementos uses in checkpoint maintenance, causes irreversible wear to flash cells. It is well known that flash cells can tolerate only 10,000 to 1 million erasures before becoming unusable. To mitigate the effect of its bundle management scheme on flash lifetime, Mementos could be extended to use information coding schemes to allow rewrites without erasures [21]. Future RFID-scale devices might provide nonvolatile memory in the form of phase-change memory (PCM), magneto-resistive RAM (MRAM) or ferroelectic RAM (FeRAM), all of which import fewer

65

complications and are more forgiving with respect to erasure, but flash remains the most widespread form of nonvolatile memory in use today.

3.7.3 Future Work

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.

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.

Reducing checkpoint sizes has been a concern for previous checkpointing systems; past approaches have included memory exclusion [114] 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 [81] 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.

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.

3.8 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.

We borrow our definition of *checkpointing* from Bernstein et al. [13], 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 [103, 95, 86], 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 [13, 94] or file systems [131, 153].

Plank et al. [115] 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 [90, 92]

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 [152] 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. [18] 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.

Also on the topic of checkpointing state on larger-scale computers, Narayanan and Hodson use NVRAM-backed DRAM to endow a server-class computer with *whole-system persistence* (WSP) [106]. Their system implements a "flush-on-fail" policy whereby the system responds to power failures by checkpointing all volatile state to NVRAM, using only the residual capacitance of the server's power supply to accomplish this task. WSP performs favorably against *nonvolatile heaps* that implement "flush-on-commit" policies [38, 165] to provide object persistence; the primary source of the performance advantage is the delayed flushing behavior. The key difference between WSP and Mementos, other than scale, is that WSP triggers flushes as a side effect of power failures, whereas Mementos polls its energy supply. With appropriate hardware support, a CRFID or other TPC could simply trigger Mementos checkpointing via an interrupt handler, matching WSP's behavior. The notion of NVRAM transparently backing DRAM is harder to match to low-power microcontrollers using

69

SRAM and separate memory-mapped flash, but it may be beneficial to evaluate such an architectural change to support transiently powered computation. Another minor difference is that WSP implements state checkpointing, but not yet state recovery; this latter task requires special BIOS support and is presumably more difficult than Mementos-style state recovery on a microcontroller.

An alternative approach to program-state recovery is to partition programs into pieces that are idempotent, i.e., free of side effects, and then simply re-execute those pieces at run time to recover state. de Kruijf et al. identify idempotent regions through static analysis [41]. Such a strategy is appealing on machines that are constantly powered and must, for example, occasionally recover from device driver faults, but under transient power where power loss is the common case, constant re-execution may result in Sisyphean tasks—which Mementos aims to avoid.

Recent work in sensor networks considers the problem of whole-network checkpointing [109], using MSPsim for experimentation on continuously-powered sensor networks running the Contiki OS [46]. 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 [35], based on TinyOS [89], 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 TinyN-

ode [45], 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.

Sensor networking research has also led to developments in cooperative checkpointing. Using neighbor nodes to store state information [174] is not an option for CRFIDs because they lack the ability to initiate conversations. Checkpointing as a "macroprogramming" primitive [61] is an appealing concept, but no macroprogramming platform for CRFIDs currently exists.

Specific to RFID-scale devices, Buettner et al. [22, 23] 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. [31] 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. [36] and Gummeson et al. [62] 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. [136] 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.

Tseng et al. explore the behavior of NAND flash memory under power failures [162], arriving at several surprising conclusions. Specifically, they observe nonlinear decrease in error rates with write time, corruption of data other than the data being written, and increasing error rates over time in cells to which writes were interrupted. While these findings may not apply to the NOR flash found on board MSP430 and similar microcontrollers, they highlight the fact that flash memories—both NAND and NOR—often exhibit behavior that catches application designers by surprise. An operational goal of Mementos is to finish writing to flash memory before a power failure occurs, somewhat alleviating these concerns.

Buettner et al. integrate a voltage-aware task scheduler into the firmware of a WISP [24]. Given a measured voltage level, their scheduler selects a task from pre-defined 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.

Reddi et al. observe that designers often conservatively design to accommodate large deviations in operating current within microprocessors—so-called *voltage emergencies* that can cause timing problems—but that these conservative designs result in excessive power draw during idle periods [126]. They develop a system that learns "signatures" of voltage emergencies at run time, then applies throttling to decrease power consumption at critical junctures. Under the observation that a few code points cause most of the emergencies, Reddi et al. develop an extension to restructure code [127] at run time, dynamically adjusting scheduling to slow the issue rate during stalls. Mementos would benefit from an ability to predict power failures, but unfortunately, the power problems Mementos hedges against are unpredictable, as opposed to the deterministic voltage emergencies Reddi et al. address.

Accurate predictions of program run time could allow Mementos to predict the energy use of all or part of a program, then to tag program segments with metadata to aid in checkpointing decisions. Seshia and Kotker describe GameTime, a tool that uses game-theoretic analysis of program behavior to predict worst-case run times of a variety of programs [143]. This approach involves program profiling and may be compatible with the modified MSPsim described in this thesis, but we have yet to implement it.

3.9   Summary

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.

CHAPTER 4

COMMUNICATION AND STORAGE UNDER TRANSIENT POWER[1]

Research involving low-energy computing systems has long treated radio as an energy-hungry resource to be used sparingly. For systems that combine a microcontroller and an active (powered) radio, this assumption is reasonable—radio transmission can easily require as least 17 times more power than the microcontroller's fully operational active mode [52]. However, for computational RFIDs that harvest RF energy and lack active radios, there is a key difference in one area: *communication consumes less energy than persistent local storage*. Data comes to tags "for free" along with the RF waves that carry power; transmitting back to a reader involves modulating a single transistor. Writing and erasing on-chip flash memory is comparatively expensive, leading to the question: can off-tag storage provide a usable alternative to on-tag nonvolatile storage?

This chapter describes a system that uses radio as a resource to amplify the storage capabilities of computational RFIDs. The main idea of this chapter is that a CRFID can securely use radio-based off-tag storage as a less energy-intensive alternative to local, flash-based storage. The smaller energy requirements of radio allow the CRFID either to devote more energy to computation or to accomplish the same tasks using less energy, which may translate into a longer operating range or a longer period of active computation.

---

[1]This chapter first appeared as Salajegheh et al. in 2009 [135]; it appears here with minor modifications.

To mitigate the threats posed by untrustworthy RFID readers storing state on behalf of CRFIDs, the system uses established cryptographic mechanisms to protect against readers that could attempt to violate the confidentiality, authenticity, integrity, and freshness of the data on a CRFID. However, the cryptographic overhead threatens to eliminate the energy advantage of remote storage. Thus, the main challenge is to design an energy-saving remote storage system that provides security under the constraints of passive RFID systems.

This chapter uses computational state checkpointing as an example of an application that benefits from our radio-based storage technique. Our system, *Cryptographic Computational Continuation Passing* (CCCP), enables CRFIDs to perform sophisticated computations despite limited energy and continual interruptions of power that lead to complete loss of the contents of RAM. CCCP extends the Mementos architecture [124] for execution checkpointing by securely storing a CRFID's computational state on the untrusted RFID reader infrastructure that powers the CRFID, thereby making program execution on CRFIDs robust against loss of power. The design of CCCP is motivated by (1) a desire to minimize the amount of energy devoted to flash memory writes and (2) the observation that a CRFID's backscatter transmission is surprisingly efficient compared to alternatives such as active radio (like that found in motes) or flash memory writes.

This chapter's contribution is the synthesis of several existing ideas with techniques that are specifically applicable to computational RFIDs:

- We describe the design and implementation of CCCP, a remote storage protocol that suits the characteristics and constraints of CRFIDs and is secure under a reasonable threat model. We show how this protocol can be used in the contexts of execution checkpointing and external data storage on an untrusted RFID reader infrastructure (Sections 4.2, 4.3).

- Motivated by a desire to save energy when storing CCCP's numeric counters to nonvolatile memory, we introduce *hole punching* (Section 4.2.4.4), a unary encoding technique that allows a counter stored in flash memory to be updated economically, minimizing energy- and time-intensive flash erase operations. For a CRFID, less-frequent flash erasure means more energy available for computation.

Since CCCP involves communication with a potentially untrustworthy RFID reader, it must ensure the integrity, confidentiality, and data freshness of checkpointed messages. For message **integrity**, CCCP employs UMAC [14], a Message Authentication Code (MAC) scheme based on universal hash functions (UHF) that involves the application of a cryptographically secure pseudorandom pad. Remotely stored messages in CCCP are encrypted for **confidentiality** using a simple stream cipher. For data **freshness**, CCCP employs a monotonically increasing counter to detect superseded data. CCCP's frequent use of key material motivates the use of opportunistic precomputation: when a CRFID is receiving abundant energy, CCCP generates and stores keystream bits in flash memory for later consumption. CCCP maintains a small amount of its own state in local nonvolatile memory, including a counter that must be updated during checkpoint operations when energy may be low. To minimize the energy required to update the counter, CCCP employs hole punching.

Conventional passive RFID tags perform rudimentary computation, often in extremely tight real-time constraints using nonprogrammable finite state machines [1], but CRFIDs offer true general-purpose computational capabilities, broadening the range of their possible applications (Section 4.5). Although CRFIDs offer more flexibility, they present challenging resource constraints. While sensor motes, which rely on batteries for power, often have an active lifetime measured in weeks or months, a CRFID may be able to compute for less than a second given a burst of energy, and may receive such bursts in quick succession—putting CRFIDs in an entirely different

class with regard to energy constraints. Moreover, although CRFIDs have a small amount of flash memory available as nonvolatile storage, writing to this flash memory is energy intensive (Section 4.1.1).

CCCP extends the execution checkpointing system Mementos [124] by adding remote, rather than local flash-based, storage capabilities to CRFIDs. While systems such as Mementos investigate how to effectively store checkpoints locally in trusted flash memory to achieve computational progress on CRFIDs despite power interruptions, CCCP focuses on using external, *untrusted* resources to increase tag storage capacity in an energy-efficient manner with reasonable security guarantees.

4.1 Backscattering CRFIDs

Computational RFIDs use backscatter communication in which they use a small transistor to modulate their reflection of a more-powerful device's radio waves. Several key observations motivate the development of secure remote storage for CRFIDs.

The first observation is that frequent loss of power may interrupt computation. The CRFID model posits computing devices that are primarily powered by RF energy harvesting, a mechanism that is naturally finicky because of its dependence on physical conditions. Any change to a CRFID's physical situation—such as its position or the introduction of an occluding body—may affect its ability to harvest energy. Existing systems that use RF harvesting typically counteract the effect of physical conditions by placing stringent requirements on use. For example, an RFID transit card reader presented with a card may behave in an undefined way unless the card is within 1 cm for at least 300 ms, parameters designed to ensure that the card's computation finishes while it is still near the reader. CRFID applications may preclude such a strategy: programs on general-purpose CRFIDs do not offer convenient execution time horizons, and communication distances are beyond the tag's control. Without any guarantees of energy availability, it may be unreasonable to mandate that pro-
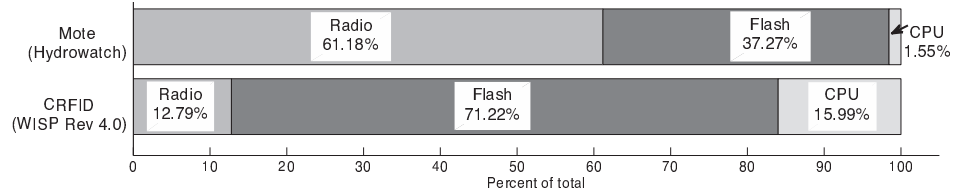
Figure 4.1. Per-component maximum power consumption of two embedded devices. Radio communication on the WISP requires less power than writes to flash memory. The relative magnitudes of the power requirements means that a sensor mote favors shifting storage workloads to local flash memory instead of remote storage via radio, while a computational RFID favors radio over flash. The numbers for the mote are calculated based on the current consumption numbers given by Fonseca et al. [52]. For the CRFID, we measured three operations (radio transmit, flash write, and register-to-register move) for a 128-byte payload.

grams running on CRFIDs complete within a single energy lifecycle. As an extension of the Mementos system [124], CCCP aims to address the problem of suspending and resuming computations to facilitate spreading work across multiple energy lifecycles.

The second observation is that storing remotely may require less energy than storing locally. Some amount of onboard nonvolatile memory exists on a CRFID, so an obvious approach to suspension and resumption is simply to use this local memory for state storage. However, to implement nonvolatile storage, current microcontrollers use flash memory, which imports several undesirable properties. While reading from flash consumes energy comparable to reading from volatile RAM, the other two flash operations—writing and erasing—require orders of magnitude more energy per datum (Table 4.3). Our measurements of a CRFID prototype reveal that the energy consumption of storing a datum locally in flash can in fact exceed the energy consumption of transmitting the same datum via backscatter communication.

Figure 4.1 illustrates the difference between flash and radio storage on a CRFID and shows how the relationship is different on a sensor mote. The figure helps explain why designers of mote-based systems choose to minimize radio communication; simi-

larly, it justifies our exploration of radio-based storage as an alternative to flash-based storage on CRFIDs.

It should be noted that CCCP, although its primary data storage mechanism is the communication link between CRFIDs and readers, still requires *some* flash writes during storage operations: CCCP maintains a counter in flash to ensure that it does not reuse key material. However, because CCCP employs hole punching (Section 4.2.4.4) to maintain the counter, the amount of data written for counter updates is small compared to the amount of data that can be stored at once—small enough not to obviate the energy advantage of radio-based storage—and counter updates do not frequently necessitate erasures.

EPC Gen 2 RFID readers are typically not standalone devices. Rather, they are connected to networks or other systems (for, e.g., control or logging) that can offer computing resources such as storage. The benefit to CRFIDs that communicate with such a reader infrastructure is access to effectively limitless storage. Several kilobytes of onboard flash memory is minuscule compared to the potentially vast amount of storage available to networked RFID readers. While unlimited external storage is not obviously helpful for saving computational state—a CRFID cannot save or restore more state than it can hold locally—its usefulness as general-purpose long-term storage is analogous to the usefulness of networked storage for PCs.

RFID protocols allow arbitrary payloads. While the EPC Gen 2 protocol imposes constraints on the transmissions between RFID tags and RFID readers—for example, the maximum upstream data rate from tag to reader is 640 Kbps [48]—it also offers sufficient flexibility that CCCP can be implemented on top. In particular, the Gen 2 protocol permits a reader to issue a *Read* command to which a tag can respond with an arbitrary amount of data. Previous versions of the EPC RFID standard mandated a small response size that would have imposed severe communication overhead on large upstream transmissions.

CRFID is not married to EPC Gen 2 as an underlying protocol, but the existence of a widespread RFID reader infrastructure and the availability of commodity reader hardware makes for easy prototyping. The *backscatter anything-to-tag* protocol (BAT) [105] offers an appealing alternative for CRFID communication that is better suited to arbitrary payloads than Gen 2, but we do not evaluate it in this chapter because of gaps in its implementation.

4.1.1 Challenges From Energy Scarcity

Several energy-related considerations limit the resources available for computation on CRFIDs, limiting the utility of CRFIDs as a general-purpose computing platform. Ransford et al. [122] discuss the difficulty of effectively utilizing a storage capacitor and enumerate the drawbacks of using capacitors for energy storage; Buettner et al. [23] discuss how energy limitations bear on the deployment of a CRFID-based system. Two key design features of CRFIDs pose energy challenges to a system like CCCP: first, the voltage and current requirements of flash memory constrain the design of flash-bearing CRFIDs and limit the portion of a CRFID's energy lifecycle that is usable for computation. Second, a CRFID's reliance on energy harvesting and backscatter communication means that a CRFID cannot compute or communicate without reader contact.

Unfortunately, flash memory imposes limitations. Microcontrollers that incorporate flash typically have separate threshold voltages: one threshold for computation, and a higher threshold for flash writes and erases. Because of this difference, flash writes cannot be executed at arbitrary times during computation on a CRFID; they require sufficient voltage on the storage capacitor. Without a constant supply of energy, capacitor voltage declines with time and computation, so waiting until the end of a computation to record its output to nonvolatile memory may be risky.

The size of a CRFID's storage capacitor imposes another basic limitation. Flash writes, which owe their durability to a process that effects significant physical changes, require more current and time (and therefore energy) than much simpler RAM or register writes. Per-datum measurements show that, on a WISP's microcontroller, writing to flash consumes roughly 400 times as much energy as writing to a register [122]. Such outflow from the storage capacitor can dramatically shorten the device's energy lifecycles.

Another challenge is that of non-autonomous operation. Backscatter communication involves modulating an antenna's impedance to reflect radio waves—an operation that, for the sender, involves merely toggling a transistor to transmit binary data. Such communication cannot occur without a signal to reflect; CRFIDs, like other passive RFIDs, are therefore constrained to communicate only when a reader within range is transmitting. Computation may occur during times of radio silence, but only if sufficient energy remains in the CRFID's storage capacitor. Unlike battery-powered platforms that can operate autonomously between beacon messages from other entities, a CRFID may completely lose power and volatile state between interactions with an RFID reader. Our experience shows that, lacking a source of harvestable energy, the storage capacitor on a WISP can support roughly one second of steady computation before its voltage falls below the microcontroller's operating threshold. (For perspective, note that a single checksum operation over $2\,\mathrm{KB}$ takes $575\,\mathrm{ms}$ on an MSP430 microcontroller running at $1\,\mathrm{MHz}$ [124].) Such limitations constrain the design space of applications that can run on CRFIDs. For example, without autonomy, an application cannot plan to perform an action at a specific time in the future.

A final challenge is the unsteady energy supply. For CRFIDs, the supply of energy can be unsteady and unpredictable, especially under changing physical conditions. RFID readers may not broadcast continuously or even at regular intervals, and they do not promise any particular energy delivery schedule to tags. In our experiments,

81

even within inches of an RFID reader that emitted RF energy at a steady known rate, the voltage on a CRFID's storage capacitor did not appear qualitatively easy to predict despite the fixed conditions. A CRFID's storage capacitor must buffer a potentially unsteady supply of RF energy without the ability to predict future energy availability.

4.2 Design of CCCP

CCCP's primary design goal is to furnish computational RFIDs with a mechanism for secure outsourced storage that facilitates the suspension and resumption of programs. This section describes how CCCP is designed to meet that goal and several others. Refer to Section 4.3 for a discussion of CCCP's implementation, and refer to Section 4.4 for an evaluation of CCCP's design choices and security; in particular, Section 4.4.3.1 discusses the overhead imposed by cryptographic operations.

Design goal	Approach
Computational progress	Communication of checkpoints via radio to untrusted RFID readers
Security: authentication, integrity	UHF-based MAC
Security: data freshness	Key non-reuse; counter stored by hole punching in nonvolatile memory
Security: confidentiality	Symmetric encryption with keystream precomputation

Table 4.1. CCCP's design goals and techniques for accomplishing each of them.

Given a chunk of serialized computational state on a CRFID, CCCP sends the state to the reader infrastructure for storage. (CCCP is designed to work independently of the state serialization method, and does not prescribe a specific method.) In a subsequent energy lifecycle, an RFID reader that establishes communication with the tag sends back the state, CCCP performs appropriate checks, and the CRFID resumes computation where it left off. CCCP provides several operating modes that allow an application designer to increase security—by adding authentication alone,

82

or authentication and encryption—at the cost of additional per-checkpoint energy consumption. Table 4.1 describes how CCCP meets each of the goals discussed in this section.

4.2.1 Design Goal: Computational Progress on CRFIDs

CCCP remotely checkpoints computational state to make long-running operations robust against power loss—i.e., to enable their *computational progress*, defined as change of computational state toward a goal (e.g., the completion of a loop). While CRFIDs are able to finish short computations in a small number of energy lifecycles (e.g., symmetric-key challenge-response protocols [31, 69]), the challenges described in Section 4.1.1 make it difficult for a CRFID to guarantee the computational progress of longer-running computations.

If a CRFID loses power before it completes a computation, all volatile state involved in the computation is lost and must be recomputed in the next cycle. If energy availability is similarly inadequate in subsequent cycles, the CRFID may never obtain enough energy to finish its computation or even to checkpoint its state to flash memory. Following Mementos [124], we refer to such vexatious computations as *Sisyphean tasks*. (Sisyphus was condemned to roll a large stone up a hill, but was doomed to drop the stone and repeat hopelessly forever [73].) A major goal of CCCP is to prevent tasks from becoming Sisyphean by shifting energy use away from flash operations and toward less energy-intensive radio communication.

4.2.2 Checkpointing Strategies: Local vs. Remote

We consider two strategies for the nonvolatile storage of serialized checkpointed state. The first, writing the state to flash memory, involves finding an appropriately sized region of erased flash memory or creating one via erase operations. The second strategy, using CCCP, requires a CRFID to perform zero or more crypto-

graphic operations (depending on the operating mode) and then transmit the result via backscatter communication.

The obvious advantage of flash memory is that its proximity to the CRFID makes it readily accessible. On-chip flash has the further advantage that it may be inaccessible to an attacker. However, the operating requirements of flash are onerous in many situations. With unlimited energy, a CRFID could use flash freely and avoid the complexity of a radio protocol such as CCCP. Unfortunately, energy is limited in ways described elsewhere in this chapter, and several disadvantages of flash memory diminish its appeal as a store for checkpointed state. The most obvious disadvantage is an imbalance between the requirements for reading and writing. Write and erase operations require more time, energy, voltage and current per bit than reading (Table 4.3); additionally, the minimum voltage and current requirements are higher. For example, in the case of the MSP430F2274, read operations are supported at the microcontroller's minimum operating voltage of $1.8\,\text{V}$, but write and erase operations require $2.2\,\text{V}$. Finally, flash memory (both NOR and NAND types) generally imposes the requirement that memory segments be erased before they are written: if a bit acquires a zero value, the entire segment that contains it must be erased for that bit to return to its default value of 1. Aside from burdening the application programmer with inconvenience, erase-before-write semantics complicate considerations of energy requirements. These disadvantages are minor afflictions for higher-powered systems, but they pose serious threats to the utility of flash memory on CRFIDs.

Backscatter transmission, since it involves modulating only a single transistor to encode data, requires significantly less energy than transmission via active radio. In fact, our measurements (Figure 4.4) show that backscatter transmission of an authenticated, encrypted state checkpoint (plus a small amount of bookkeeping in flash) can require less energy than exclusively writing to flash memory, even after including the energy cost of encrypting and hashing the checkpointed state. Because

84

of its consistent behavior throughout the microcontroller's operating voltage range, backscatter transmission is an especially attractive option when the CRFID receives radio contact frequently but cannot harvest energy efficiently, in which case writing to flash may be infeasible because of insufficient energy in the storage capacitor. These circumstances may occur far from the reader, or in the presence of radio occlusions, or when a computation uses energy quickly as soon as the CRFID wakes up.

Despite its advantages over flash storage and active radio, CCCP's reliance on backscatter transmission has drawbacks. Bitrate limitations in the EPC Gen 2 protocol cause CCCP's transmissions to require up to twice as much time per datum as flash storage on some workloads. The best choice of storage strategy depends on an application's ability to tolerate delay and the necessity of saving energy.

4.2.3 Threat Model

We define CCCP's threat model as a superset of the attacks that typically threaten RFID systems [78]. The most obvious way an attacker can disrupt the operation of a CRFID is to starve it of energy by jamming, interrupting, or simply never providing RF energy for the CRFID to harvest. Because they depend entirely on harvestable energy, CRFIDs cannot defend against such denial-of-service (DoS) attacks, so we consider these attacks as a problem to be dealt with at a higher system level. We instead focus on two types of attacks that a CRFID can use its resources to address: (1) active and passive radio attacks and (2) attacks by an untrusted storage facility.

An adversary may attempt to:

- Eavesdrop on radio communication in both directions between a CRFID and reader.
- Masquerade as a legitimate RFID reader in order to collect checkpointed state from CRFIDs. Because CRFIDs do not trust reader infrastructure, such an attack should allow an attacker to collect only ciphertext.

- Masquerade as a legitimate RFID reader in order to send corrupted data or old data (e.g., a previous computational state) to the CRFID. Such invalid data should not trick the CRFID into executing arbitrary or inappropriate code.

- Masquerade as a specific legitimate CRFID in order to retrieve that CRFID's stored state from the reader. This state should be useless without access to the keystream material that encrypted it—keystream material that is stored in the legitimate owner's nonvolatile memory and never transmitted.

We additionally assume that an adversary cannot physically inspect the contents of a CRFID's memory.

4.2.4 Secure Storage in CCCP

Because computational RFIDs depend on RFID readers for energy—if a CRFID is awake, there is probably a reader nearby—the reader infrastructure is a natural choice for storing information. But a reader trusted to provide energy should not necessarily be trusted with sensitive information such as checkpointed state.

CCCP involves communication with untrusted reader infrastructure, so we establish several security goals:

- Authenticity: a CRFID that stores information on external infrastructure will eventually attempt to retrieve that information, and the authenticity of that information must be cryptographically guaranteed. Under CCCP, the only party that ever needs to verify the authenticity of a CRFID's stored information is the CRFID itself.

- Integrity: an untrusted reader may attempt to impede a CRFID's computational progress by providing data from which the CRFID cannot resume computation (e.g., random junk). While CCCP cannot prevent a denial of service attack in which a reader provides only junk, it guarantees that CRFIDs will compute only on data they recognize as their own.

- Data freshness: just as a reader can provide corrupted data instead of usable data, it can replay old state in an attempt to hinder the progress of a computation. Under CCCP, a CRFID recognizes and rejects state that has been superseded.

- Confidentiality: in certain applications, the leaking of intermediate computational state might be a critical security flaw. For other applications, confidentiality may not be necessary. CCCP offers a configurable level of protection for application designers.

4.2.4.1 Keystream Precomputation

Because CCCP's threat model assumes a powerful adversary that can intercept all transmissions, CCCP never reuses keystream material when encrypting data or computing MACs. We use CCCP's refreshable pool of pseudorandom bits (a circular buffer in the CRFID's nonvolatile memory) as a cryptographic keystream to provide confidentiality and authentication.

CCCP stores keystream material on the CRFID because we assume that the CRFID trusts only itself; a CRFID cannot extract trustworthy keystream material from a reader it does not trust, nor from any observable external phenomenon (which, in our threat model, an attacker would be able to observe equally well). Because a CRFID can reserve only finite storage for storage of keystream material, the material must be periodically refreshed. CCCP opportunistically refreshes the keystream material with pseudorandom bits, following Algorithm 3.

To provide unique keystream bits to cryptographic operations (encryption and MAC), CCCP uses an existing implementation [31] of the RC5 block cipher [128] in counter mode to generate pseudorandom bits and store them to flash. The choice of a block cipher in counter mode means that the resulting MAC and ciphertext are secure against a computationally bounded adversary [17]. A stream cipher would work equally well in principle, but in implementing CCCP, we found that those under

consideration required a large amount of internal read-write state. For example, the stream cipher ARC4 requires at least 256 bytes of RAM [140], whereas RC5 requires only an 8-byte counter. The RC5 key schedule is preloaded into flash memory the first time the device is programmed, and the keystream materials are generated during periods of excess energy (or *power seasons*; see Section 4.2.5). One such period of excess energy is the CRFID's initial programming, at which time the entire keystream buffer is filled with keystream bits. To avoid reusing keystream bits, CCCP maintains several variables in nonvolatile memory (the aforementioned "bookkeeping"). Table 4.2 summarizes the variables CCCP stores in nonvolatile memory.

Variable	Description
chk_ctr	Counter representing the number of checkpoints completed; used to calculate the location of the first unused keystream material; updated each time keystream material is consumed; unary representation
kstr_end	Pointer to the end of the last chunk of unused keystream bits in keystream memory; updated during key refreshment
rc5counter	Incrementing counter used as an input to RC5 while filling keystream memory with pseudorandom data; updated during key refreshment

Table 4.2. Variables CCCP stores in nonvolatile memory.

4.2.4.2   UHF-based MAC for Authentication and Integrity

CCCP uses a MAC scheme based on universal hash functions (UHF) [28] to provide authentication and integrity. CCCP constructs the MAC by first hashing the message and then XORing the 80-bit hash with a precomputed cryptographic keystream. Because of the resource constraints of CRFIDs, it is critical to use a scheme that consumes minimal energy, and according to recent literature [14, 50], UHF-based MACs are potentially an order of magnitude faster than MACs based on cryptographic hash functions. We chose UMAC [14] as the UHF-based MAC function after evaluating several alternatives. Our experiments on WISP (Revision 4.0) CR-

FIDs determined that UMAC takes on average $18.4 \, \mathrm{ms}$ and requires $28.8 \, \mu\mathrm{J}$ of energy given a 64-byte input.

4.2.4.3 Stream Cipher for Confidentiality

To provide confidentiality, a CRFID simply XORs its computational state with a precomputed cryptographic keystream. This encryption scheme is low-cost in terms of computation and energy, but it relies on using each keystream bit at most once. CCCP ensures that the encryption and MAC functions never reuse keystream bits by keeping track of the beginning and end of fresh keystream material in flash memory. The keystream pool is represented as a circular buffer. The address of the first unused keystream material is derived from the value of `chk_ctr` (Table 4.2), and the last unused keystream material ends just before the address pointed to by `kstr_end`.

If the application using CCCP demands confidentiality at all times, then if CCCP cannot satisfy a request for unused keystream bits, it pauses its work to generate more keystream bits. This behavior is inspired by that of the blocking `random` device in Linux [65].

4.2.4.4 Hole Punching for Counters Stored in Flash

To avoid reusing keystream material, CCCP maintains a counter (`chk_ctr`) from which the address of the first unused keystream bits can be derived. The counter is stored in flash memory because it is used for state restoration after power loss. However, incrementing a counter stored in binary representation always requires changing a 0 bit into a 1 bit (Figure 4.2(a)). On segmented flash memories, changing a single bit to 1 requires the erasure (setting to 1) of the entire segment that contains it—at least 128 bytes on the MSP430F2274—before the new value can be written. An additional cost that varies among flash cells is that they wear out with repeated erasure and writing [66].

$00000111_2 \;\; (=7_{10})$ $...11110000000_1 \;\; (=7_{10})$

$\downarrow$

$11111111_2 \;\; \text{(erase)}$

$\downarrow$ $\downarrow$

$00001000_2 \;\; (=8_{10})$ $...11100000000_1 \;\; (=8_{10})$

(a) Binary counter   (b) Unary counter
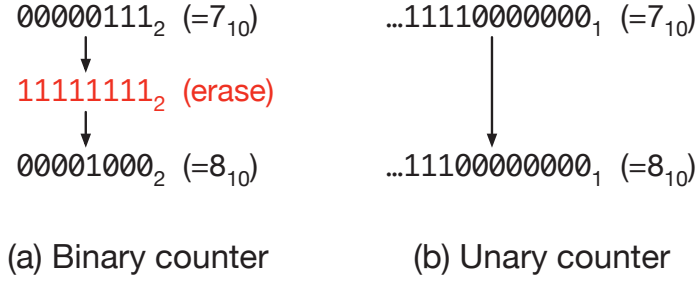
Figure 4.2. Illustration of hole punching. While incrementing a binary counter (a) in flash memory may require an energy-intensive erase operation, complemented unary representation ((b), with the number of zeros, or "holes," representing the counter value) allows for incrementing without erasure at a cost of space efficiency.

To avoid energy-intensive erasures and minimize the energy cost of writing counter updates, CCCP represents `chk_ctr` in complemented unary instead of binary. CCCP interprets the value of such a counter as the number of 0 bits therein. Because 1 bits can be changed to 0 bits without erasure, incrementing a counter requires a relatively small write, with erasures necessary only if the unary counter must be extended into unerased memory. We call this technique *hole punching* after the visual effect of turning 1 digits into 0 digits. Since `chk_ctr` is simply incremented at each remote checkpoint, updating the counter generally requires writing only a single word. Table 4.3 illustrates the energy cost of erasing an entire segment and the energy cost of writing a single word.

Operation	Seg. erase	Write	Read	Write
Size (bytes)	128	128	128	2
Energy (μJ)	46.81	56.97	0.64	0.96

Table 4.3. Comparison of energy required for flash operations on an MSP430F2274. Hole punching often allows CCCP to use a single-word write (2 bytes on the MSP430) instead of a segment erase when incrementing a complemented unary counter.

To minimize the length of the unary `chk_ctr`'s representation and to facilitate simple computation of offsets, CCCP assumes a fixed size for checkpointed state;

90

in practice an application designer can choose an appropriate value for the fixed checkpoint size.

4.2.4.5 Extension for Long-Term Storage

Under CCCP, readers can act not only as outsourced storage for computational state, but also as long-term external storage. Because of their ultra-low-power microcontrollers, CRFIDs are likely to have only a small amount of flash available for data storage. Moreover, since flash operations are energy intensive, depending exclusively on flash memory as a storage medium is undesirable. CCCP could enable a CRFID to instead use the reader infrastructure as an external storage facility with effectively limitless space.

Long-term storage requires a different key management strategy than checkpointing data. With a temporary checkpointing system, the CRFID needs access only to the keystream material used to prepare the last checkpoint sent to a reader. However, in the case of long-term storage, the CRFID may require access to all of the data it has ever stored on the reader and therefore must remember all of the cryptographic keys from those stores. To avoid this unrealistic requirement, an extension to CCCP allows the CRFID to generate keys on demand when long-term storage is required.

There are two operations that CCCP can provide to a CRFID application for this purpose:

- To satisfy a STORE(*data*) request, CCCP provides a keystream generator in the form of a block cipher in counter mode; this requires a monotonically increasing counter in addition to CCCP's chk_ctr. CCCP XORs the given *data* with the generated keystream and then constructs a MAC, then sends the ciphertext and MAC to the reader for storage. CCCP then sends the counter value back to the application.

91

- To satisfy a RETRIEVE(*index*) request, CCCP asks the RFID reader for the data at the given index. CCCP then regenerates the same keystream it used to encrypt the data by passing the *index* to the block cipher. Finally, CCCP verifies the MAC provided by the reader and returns the decrypted data to the application.

4.2.5 Power Seasons

If a CRFID could predict future energy availability, then it would be able to schedule its generation of keystream bits and ensure that it never exhausted its supply of pseudorandomness during normal operation. However, because CRFIDs lack autonomy and cannot depend on RFID reader infrastructure to provide a steady energy supply, we roughly classify the energy availability scenarios a CRFID faces into two *seasons*. We assume that the general case is a *winter* season, in which a CRFID cannot consistently harvest enough energy to perform all of its tasks. During winter, the CRFID must focus on minimizing checkpoints and wasted energy. The other season is *summer*, during which harvested energy is plentiful and the CRFID can afford to perform energy-intensive operations such as precomputation and storage of keystream material for later use.

CCCP can identify a summer season if one of two conditions is true. First, the CRFID may find itself awake with no computations left to complete, for example after it has finished and processed a sensor reading. Second, the CRFID may find itself communicating with an entity that does not understand CCCP and simply provides harvestable energy.

4.3 Implementation

The components of CCCP span two environments: CRFIDs and RFID readers. On a CRFID, CCCP accepts data from an application and uses the CRFID's

backscatter mechanism to ship the data to a reader. The reader (which we consider as an RFID reader plus a controlling computer) is programmed to participate in the CCCP protocol and return computational state where necessary. This section describes the CRFID-side components, the reader-side components, and the protocol that ties them together.

The CRFID side of CCCP is implemented in the C programming language on WISP (Revision 4.0) prototypes. At its core are three primary routines, which we present in pseudocode: CHECKPOINT (Algorithm 1), RESUME (Algorithm 2), and KEY-REFRESH (Algorithm 3). CHECKPOINT and RESUME refer to a counter called chk_ctr from which CCCP derives the address of the first unused keystream material. For routines that require radio communication, we borrow radio code from Intel's WISP firmware version 1.4. Note that, since a CRFID cannot assume that a reader is listening at an arbitrary time, the TRANSMIT subroutine waits for a signal indicating that the CRFID has received a go-ahead message from the reader.

The RFID reader side of CCCP consists only of code to drive the reader appropriately for communication events. Because of the Gen 2 protocol's complexity, we have not completely implemented the reader side of the CCCP protocol. Rather than write a large amount of code for the reader, we chose to use simple control programs for the reader and inspect the exchanged messages manually, a strategy that allowed us to concentrate on the more resource-constrained CRFID side of the system while avoiding porting applications from one proprietary reader to another. (The WISP is nominally compatible with only the Alien ALR-9800 and Impinj Speedway readers; we chose to use a desktop PC to program these readers for the sake of simplicity and portability.) A full implementation of the reader side would properly parse each message received from the CRFID and manage storage for checkpointed state.

Algorithm 1 The *Checkpoint* routine encrypts, MACs, and transmits a fixed-size (*STATE_SIZE*, selected by the application designer) chunk of computational state. $\langle A, B \rangle$ means the concatenation of A and B with a delimiter in between. 80 bits is the fixed output size of *NH*, the hash function used by UMAC. For arithmetic simplicity, this pseudocode treats the *keystream* pool as an infinite array.

function CHECKPOINT(*state, keystream, counter*)
    ▷ *How much keystream material will be used in this invocation?*
    $size \leftarrow STATE\_SIZE + Length(\langle state, counter \rangle) + 80$ bits

    $k \leftarrow counter \times size$ ▷ *keystream[k] holds unused keystream material*
    $counter \leftarrow counter + 1$ ▷ *Update counter in nonvolatile memory*

    ▷ *Encrypt state by XORing with keystream material*
    $C \leftarrow state \oplus keystream[k \ldots k + STATE\_SIZE - 1]$
    $k \leftarrow k + STATE\_SIZE$ ▷ *. . . and advance k*

    ▷ *Hash the encrypted state*
    $H \leftarrow NH(\langle C, k \rangle, keystream[k \ldots k + Length(\langle C, k \rangle) - 1])$
    $k \leftarrow k + Length(\langle C, k \rangle)$ ▷ *. . . and advance k*

    ▷ *Construct an 80-bit MAC*
    $M \leftarrow H \oplus keystream[k \ldots k + Length(H) - 1]$

    $Transmit(C, M)$ ▷ *Will block until a reader is detected*
end function

4.3.1 Communication Protocol

The CRFID model places a number of restrictions on communication. The only communication hardware on a CRFID is a backscatter circuit involving an antenna and a modulating transistor; an active radio would require significantly more energy. Since backscatter simply reflects an incoming carrier signal, a prerequisite for communication is that the reader emits an appropriate carrier signal. In our experiments, we used two different EPC Gen 2-compatible RFID readers that are readily available as off-the-shelf products; we used no nonstandard reader hardware or antennas.

CCCP's communication protocol is based on primitives provided by the EPC Gen 2 RFID protocol (the RFID protocol the WISP understands). Specifically, CCCP makes use of three EPC Gen 2 commands:

Algorithm 2 The *Resume* routine receives an encrypted checkpoint C and a message authentication code M from a reader, then restores the computational state of the CRFID if the received data pass an authenticity test. *counter* is the value stored in nonvolatile memory at the beginning of *Checkpoint* (Algorithm 1). We assume that, since k and *counter* are both numbers, their in-memory representations have the same length. As in Algorithm 1, this pseudocode treats the *keystream* pool as an infinite array for arithmetic simplicity. $\langle A, B \rangle$ means the concatenation of A and B with a delimiter in between. 80 bits is the fixed output size of *NH*, the hash function used by UMAC.

---

function RESUME($C, M, keystream, counter$)
   ▷ *Find the first unused keystream material, then backtrack to find the keystream material* CHECKPOINT *used to hash and MAC the ciphertext*

   $size = Length(C) + Length(\langle C, counter \rangle) + 80 \text{ bits}$
   $k \leftarrow counter \times size$
   $k \leftarrow k - (Length(\langle C, k \rangle) + 80 \text{ bits})$

   $H \leftarrow NH(\langle C, k \rangle, keystream[k \ldots k + Length(\langle C, k \rangle) - 1])$ ▷ *Compute the ciphertext's hash*
   $k \leftarrow k + Length(\langle C, k \rangle)$ ▷ *. . . and advance k to point to the MAC*

   if   then$M = H \oplus keystream[k \ldots k + Length(H) - 1]$ ▷ *If the MAC is OK, then. . .*
      $k \leftarrow k - (Length(C) + Length(\langle C, k \rangle))$ ▷ *backtrack further . . .*
      $state = C \oplus keystream[k \ldots k + Length(C) - 1]$ ▷ *and decrypt C to yield state*
      $Restore - State(state)$
   else
      ▷ *Do nothing*
   end if
end function

- A reader issues a *Query* command to a specific tag (in our case, a CRFID). The *Query* command comprises a 4-tuple: $\langle action, membank, pointer, length \rangle$. While a conventional RFID tag may require reasonable values for all four tuple members, a CRFID need examine only the fourth member to learn the maximum reply length the reader will accept. The reader can use the other three fields to encode meta-information such as whether the reader wants to offer checkpointed state to the CRFID.

Algorithm 3 The KEY-REFRESH algorithm replaces used keystream material with new keystream material in nonvolatile memory. Unlike in CHECKPOINT and RESUME, this pseudocode treats the *keystream* pool as a fixed-size circular buffer. This allows us to treat keystream material between k and $kstr\_end$ as unused, and the rest—between $kstr\_end$ and k—as used. This pseudocode omits two subtleties for simplicity: first, the routine must not erase keystream material that is waiting to be used by RESUME. Second, because flash erasure affects entire segments at once, the ERASE-MEMORY-RANGE routine must sometimes restore data that should not have been erased.

> **function** KEY-REFRESH($keystream, kstr\_end, chk\_ctr, rc5counter$)
>   ▷ *Find the first unused keystream material in the circular keystream buffer*
> $size \leftarrow STATE\_SIZE + (STATE\_SIZE + Length(\langle null, chk\_ctr \rangle)) + 80$ bits
> $k \leftarrow chk\_ctr \times chkpt\_size$ (mod $Length(keystream)/size$)
>
>   ▷ *Erase all used keystream memory, then write pseudorandom data to it*
>   ERASE-MEMORY-RANGE($keystream[kstr\_end \ldots k]$)
> $i \leftarrow kstr\_end$
>
>   **while** $i < k$ **do**
> $rc5counter \leftarrow rc5counter + 1$ ▷ *Update counter in NVRAM*
> $keystream[i] \leftarrow \text{RC5}(rc5counter - 1)$ ▷ *Write keystream into NVRAM*
> $kstr\_end \leftarrow i + 1$ ▷ *Update kstr_end in NVRAM*
> $i \leftarrow i + 1$
>   **end while**
> **end function**

- A reader issues a *Read* command to a specific tag to request an arbitrary amount of data from an RFID tag's memory. A CRFID can respond to a coordinated *Read* command with a chunk of checkpointed state.

- A reader issues a *Write* command to send data for storage in a specific tag's memory. Because RFID tags tend to have even fewer resources than CRFIDs, *Write* commands transmit only a small amount (16 bits) of data. A CRFID can request a series of *Write* commands from the reader to retrieve checkpointed state, then reassemble the results in memory and restore its state from the checkpoint.

Figure 4.3 gives an overview of CCCP's message types and their ordering. CCCP does not require protocol changes to the EPC Gen 2 standard, but it requires that an RFID reader be controlled by an application that understands CCCP. While a
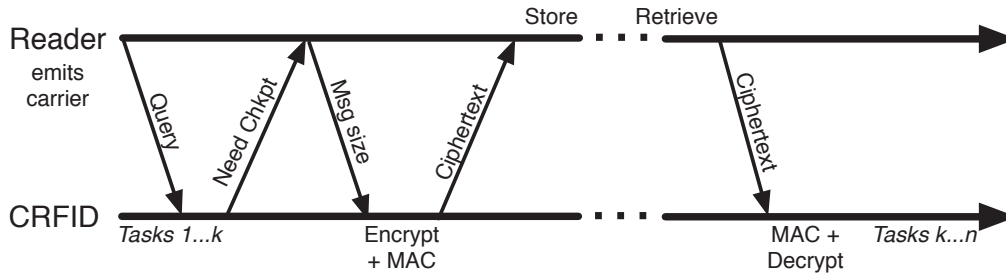
Figure 4.3. Application-level view of the CCCP protocol. The CRFID sends a request to checkpoint state while in the presence of a reader, and the reader specifies the maximum size of each message. The CRFID then prepares the checkpoint and transmits it in a series of appropriately sized messages. The reader stores the checkpoint data for later retrieval by the CRFID. All messages from the reader to the CRFID also supply power to the CRFID if the latter is within range.

proprietary radio protocol for CCCP could be more efficient than one built atop an existing RFID protocol, a goal of CCCP—inherited from the design goals of the WISP CRFID—is to maintain compatibility with existing RFID readers.

4.4 CCCP Evaluation

This section justifies our design choices and offers evidence for our previous claims. We evaluate the security properties of four distinct checkpointing strategies—three based on CCCP's radio transmission and one on local flash storage—and describe how CCCP provides data integrity with or without confidentiality. We describe our experimental setup and methods, then provide empirical evidence that CCCP's radio-based checkpointing requires less energy per checkpoint than a flash-based strategy. Finally, we characterize the overhead incurred by CCCP's cryptographic operations in terms of both the energy and the keystream material that they consume.

4.4.1 Security Semantics

CCCP trades the physical security of local storage for the energy savings of remote storage, but its use of radio communications introduces different security properties.

We consider CCCP's four operating modes in increasing order of cryptographic complexity. Note that the algorithm listings (Algorithms 1–3) describe the most computationally intensive operating mode; the other modes involve subsets of its operations.

- Under CCCP's threat model, storing checkpointed state only in local flash memory is the most secure option, since it involves no radio transmission at all. However, for reasons detailed elsewhere in this chapter, writing to flash memory is not always possible or desirable. We call the flash-only approach *Mementos* after the system [122, 124] that inspired CCCP.
- In a mode called *CCCP/NoSec*, a CRFID sends computational state in plaintext. Under CCCP's threat model, CCCP/NoSec allows an attacker to intercept computational state and trivially recover the information it contains.
- In a mode called *CCCP/Auth*, the CRFID computes a message authentication code (MAC), attaches it to plaintext computational state, and transmits both. To trick a CRFID into accepting illegitimate state, an attacker must craft a message that incorporates a MAC that the CRFID can verify. However, since CCCP's MAC routine incorporates keystream material that is local to the CRFID, the attacker must guess the contents of a chunk of the CRFID's keystream memory, which requires brute force under our threat model.
- In a mode called *CCCP/AuthConf*, CCCP encrypts computational state, computes a MAC, and transmits both (Algorithm 1). As with CCCP/Auth, an attacker who wants to trick a CRFID into accepting illegitimate state must find a hash collision; however, part of her colliding input must be a valid *encrypted* computational state from which the CRFID would be able to resume. Since CCCP does not reuse keystream material, the attacker is limited to brute-force search to find such an encrypted state.

4.4.2 Experimental Setup & Methods

We used a consistent experimental setup to obtain timing and energy measurements for a prototype CRFID. We programmed a WISP with a task (e.g., a flash write) and set a GPIO pin to toggle immediately before and after the task. We then charged the WISP's capacitor to 4.5 V using a DC power supply, disconnected the power supply so that the storage capacitor was the only source of energy for the WISP, and observed the task's execution and storage capacitor's voltage on an oscilloscope. We delivered energy directly from a DC power supply when taking measurements because the alternative, providing an RF energy supply, results in unpredictable and unsteady charge accumulation, making it difficult to shut off the energy supply at a precise capacitor voltage.

After watching the GPIO pin signal the beginning and end of the task, we calculated the task's duration and the corresponding change in the storage capacitor's voltage. When an operation completed too quickly to observe clearly on the oscilloscope, we repeated it in an unrolled loop and divided our measurements by the number of repetitions. Finally, we calculated per-bit energy values by subtracting the baseline energy consumption of the WISP with its MSP430 microcontroller in the LPM3 low-power (sleep) mode. We subtract the WISP's baseline energy consumption in order to discount the effects of omnipresent consumers such as RAM and CPU clocks. For all measurements that we present, we give the average of five trials.

4.4.3 Performance

Figure 4.4 shows that, for data sizes greater than 16 bytes, a checkpoint operation under CCCP/NoSec requires less energy than a checkpoint to flash. Under CCCP/AuthConf, which adds encryption and MAC operations, a similar threshold exists between 64 and 128 bytes. Checkpointing via flash has an additional cost: if the checkpointing mechanism needs to overwrite existing data (e.g., old checkpoints)
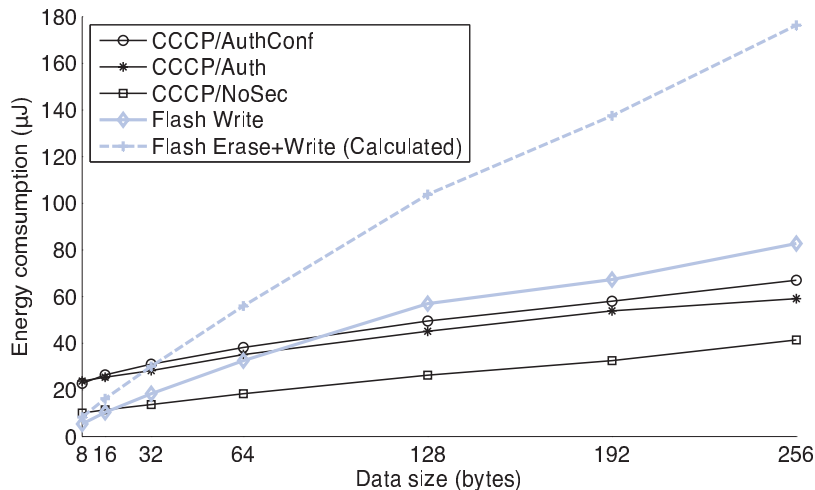
Figure 4.4. Energy consumption measurements from a WISP (Revision 4.0) proto-
type for all considered checkpointing strategies. Under our experimental method, we
are unable to execute flash writes larger than 256 bytes on current hardware because
larger data sizes exhaust the maximum amount of energy available in a single energy
lifecycle. The average and maximum percent error of the measurements are 5.85%
and 14.08% respectively.

in flash memory, it must erase the corresponding flash segments and potentially re-
place whatever data it did not overwrite. Even if a flash write does not necessitate
an immediate erasure, it makes less space available in the flash memory and therefore
increases the probability that a long-running application will eventually need to erase
the data it wrote—that is, it incurs an *energy debt*. In the ideal case, an applica-
tion can pay its energy debt easily if erasures happen to occur only when energy is
abundant—i.e., in summer power seasons. However, since CCCP is designed to ad-
dress scenarios in which energy availability fluctuates, we consider the case in which
each write incurs an energy debt. Factoring in debt, we characterize the energy cost
of a write of size `dsize` as

$$\text{Cost}^*(\text{write}(\texttt{dsize})) = \left(\text{Cost}(\text{seg. erase}) \times \frac{\texttt{dsize}}{\text{Size(seg.)}}\right) + \text{Cost}(\text{write}(\texttt{dsize})).$$

100

In practice, because some erasures will likely occur in summer power seasons and some in winter power seasons, the energy cost of a flash write of size `dsize` falls between Cost(write(`dsize`)) (the ideal cost) and Cost*(write(`dsize`)) (the worst-case cost), inclusive.

The energy measurements we present in this chapter (e.g., in Figure 4.4) fail in some cases to strongly support the hypothesis that radio-based checkpointing is consistently less energy intensive than flash-based checkpointing. The imbalance is due to a missed opportunity for optimization on the WISP prototype. The transistor used for backscatter modulation on the WISP (Revision 4.0) draws $500\,\mu$W of power, far more than is typical of a comparable mechanism on a conventional RFID tag. Alien's Higgs 3, a conventional RFID tag, draws only $15.8\,\mu$W of power [4] (total) during operation—an order of magnitude difference that supports an alternative design choice for future CRFIDs.

4.4.3.1   System Overhead

An application on a CRFID can balance energy consumption against security by choosing one of CCCP's operating modes:

- CCCP/NoSec imposes the least overhead because it does not encrypt data or compute a MAC; it requires no computation and consumes no keystream material. However, CCCP/NoSec imposes a time overhead to receive computational state from a reader at power-up and to transmit new state at checkpoint time.
- CCCP/Auth avoids encryption overhead (like CCCP/NoSec) but requires time, energy, and keystream bits to compute a MAC over the plaintext checkpoint. However, it requires no energy or keystream bits for encryption because it does not encrypt the plaintext checkpoint.
- CCCP/AuthConf offers the most security, since it adds confidentiality to CCCP/Auth, but the extra security comes at the expense of time, energy, and keystream bits.

In this mode, CCCP encrypts the computational state before computing a MAC and transmitting both. It requires as much keystream material as the size of the state plus a constant amount for authentication.

4.5   Applications

The outsourced memory introduced by CCCP expands the design space for applications on a computational RFID. This section offers some illustrative example applications.

One application for CRFIDs is low-maintenance sensing. Consider a *cold-chain monitoring* application for pharmaceutical supplies, in which a CRFID carries an attached temperature sensor and stores in flash memory a temperature reading each time it is scanned. To prevent exhaustion of its flash memory, the CRFID periodically computes aggregate statistics on, then discards, stored readings. Some statistical computations (e.g., computation of quartiles) require memory-intensive manipulation of the data set. If the flash memory on the CRFID considerably exceeds the size of RAM, computation of such statistics would require many writes to flash, an energy-intensive operation. An alternative is to use outsourced memory for the computation. (In the case of cold-chain monitoring, maintaining privacy of harvested data with respect to the reader may be unessential, but the *integrity* of the statistical computation is important.)

Another application is *RFID sensor networks* (RSNs), described in recent work [23] as networks that combine RFID reader infrastructure with sensor-equipped computational RFIDs. RSNs do not simply replace traditional sensor networks because of several limitations. First, they require an infrastructure of readers that provide power to sensor nodes. Second, they are constrained by the distances (several meters) at which CRFIDs currently operate. Third, because RFID communication is asymmetric, the nodes of an RSN cannot exchange information with each other except

102

through a more powerful reader. However, there are applications for which short-range networks of batteryless sensors would be appropriate; Yeager et al. offer several examples [170].

Computational RFIDs may also act like smartcards. Some passive RFID tags are capable of executing strong cryptographic primitives. For example, various models of the Mifare DESfire can perform triple-DES or AES, while other RFID devices can compute elliptic-curve and RSA signatures, such as the RF360 introduced by Texas Instruments [159]. The RF360 is designed to allow public-key authentication in RFID-enabled identification documents, such as e-passports.

The RF360 incorporates an MSP430, but also includes a cryptographic co-processor, and is designed to operate at relatively short range as a high-frequency, ISO 14443 device. As we show in this chapter, CCCP creates the possibility of a more lightweight device. Such a "CCCP smartcard" has two notable benefits: (1) a CCCP smartcard eliminates the cost of cryptography-specific hardware; and (2) a CCCP smartcard can operate in a mode compatible with EPC Gen 2 and achieve read ranges beyond those of a high-frequency device like the RF360.

Some smartcards are capable of performing biometric authentication—generally fingerprint verification. Match-on-card, i.e., verification of the validity of a fingerprint through computation exclusively within the smartcard, has long stood as a technical challenge. The U.S. National Institute of Standards and Technology (NIST) recently conducted an evaluation of a range of such algorithms in contactless cards [39]. CCCP is a promising tool for expanding the class of radio devices for which match-on-card is feasible. While CCCP does not follow a strict match-on-card paradigm—given that it outsources data to a reader—it nonetheless provides comparable security assurances under the same threat model.

Finally, CRFIDs may participate in work-sharing networks that distribute computation over a large area. CCCP permits a computational RFID to use external

103

memory via an RFID reader. It can support an even broader design space if we use CCCP instead for secure outsourcing not of memory, but of *computational tasks.*

Trusted platform modules (TPMs) [11, 161, 102] offer support for such outsourcing. A TPM is a hardware device, standard in the CPUs of modern PCs and servers, that can provide a secure attestation to the software configuration of the computing platform on which it operates. Briefly stated, an attestation takes the form of a digital signature on a digest of the software components loaded onto the device. (An attestation does not provide assurance against hardware tampering or subversion of running software.)

A computational RFID can in principle make use of a TPM-enabled reader—or platform communicating with the reader—to gain access to a more powerful external computer. The process for such use of a TPM is subtle. The operations of verifying a TPM attestation and creating a secure session are both cryptographic operations that require computationally intensive modular exponentiation. Hence the computational outsourcing process requires CCCP as a bootstrapping mechanism.

4.6 Related Work

CCCP is closely related to Mementos [124] in that both systems provide checkpointing of program execution on CRFIDs. Whereas Mementos relies purely on flash memory and focuses on finding optimal checkpoint frequencies via static and dynamic analysis, CCCP relies primarily on untrusted remote storage via radio and focuses on low-power cryptographic protections to ensure that remotely stored data is as secure as if it were stored locally.

Several systems share CCCP's goal of exploiting properties of RFID systems to enhance security and privacy. For instance, Shamir's SQUASH hash algorithm [144] exploits the underutilized radio link between a tag and a reader to reduce the amount of cryptographic computation necessary on a tag. While number-theoretic hash func-

tions typically require significant computational resources for modular arithmetic, the SQUASH function eliminates costly modular reductions and produces large (unreduced) hash outputs that a tag can send directly to a reader. Tags can thus use the SQUASH function to engage in secure challenge-response protocols with minimal computational resources on the tag. The scheme is provably as one-way as Rabin encryption [119]. Like SQUASH, CCCP exploits the relatively low cost of radio communication between a tag and a reader to increase security. While SQUASH increases radio communication to reduce computation, CCCP increases radio communication to reduce writes to flash memory.

CCCP uses cryptographic techniques from past work on secure file systems and secure content distribution. CFS [15], the SFS read-only file system [54], and Plutus [80] investigated how to provide secure storage layered on various degrees of untrusted infrastructure. While scalability and throughput are the main challenges in such file systems, CCCP primarily addresses energy and memory constraints. The semantics of CCCP storage are similar to the semantics of secure file systems. None of the systems explicitly and directly prevent denial of service. Storing information on untrusted RFID readers trades off the gain in storage capacity and energy conservation against the risk of losing data because of compromise or destruction of the external storage. To mitigate the risk against denial of service, CCCP could choose to replicate data as do secure file systems.

CCCP shares some goals with power-aware encryption systems such as that proposed by Chandramouli et al. [32]. Both systems are designed to consume little energy while offering the security of well-known cryptographic primitives, and both are motivated by a study of power profiling results, but they have different goals. Chandramouli et al. focus on deriving an energy-consumption model and establishing a relationship between energy consumption and security, and they offer an encryption scheme that might allow CCCP to consume less energy during its precomputation of

keystream bits. However, CCCP's opportunistic precomputation occurs during periods of abundant energy, when the choice of encryption scheme is not of the utmost importance. CCCP's precomputation allows it to use time- and energy-efficient XOR operations at checkpoint time, when energy is low; an alternative encryption scheme would have to save time or energy over simple XOR operations to be useful when energy is waning.

CCCP shares a number of properties with systems built for sensor networks. Storage-centric sensor networks [42, 101] have focused on reducing radio communication and increasing writes to flash memory to conserve energy. One of our motivating observations is that this relationship is inverted in the CRFID model: CCCP reduces writes to flash memory in favor of increasing radio communication. Performing cryptography is hard on both a CRFID and its elder cousin the sensor mote. Previous systems, such as SPINS [112] and TinySec [82] for sensor networks, have faced design challenges similar to CCCP's. SPINS and TinySec use RC5 because of its small code size and efficiency, but the battery-powered platform underlying these systems differs in fundamental ways from batteryless CRFIDs. For a side-by-side comparison of such embedded systems, see Table 1 of Chae et al. [31].

CCCP provides secure storage for CRFIDs, and CRFIDs are closely related to existing passively powered RFID tags conforming to the EPC Gen 2 standard [48]. At times the RFID and sensor world fuse together. Buettner et al. [23] propose *RFID sensor networks* (RSNs) as a replacement for wireless sensor networks in applications where batteries are inconvenient, and the authors describe RSNs built on WISP CR-FIDs. However, the RSN work does not consider remote storage options for CRFIDs.

4.7 Future Work

Avenues for future exploration include enhancements to the CCCP protocol. In particular, the protocol currently suffers from a potential atomicity problem. In

CHECKPOINT (Algorithm 1), chk_ctr is updated before the checkpointed state is transmitted, so that even if the transmission fails, chk_ctr will point to unused keystream material the next time CHECKPOINT runs. However, if CHECKPOINT updates the offset but terminates before transmission succeeds, then the next RESUME operation will see a value of chk_ctr from which its normal backtracking operation will not find the correct keystream material. CCCP cannot currently recover from such a mismatch.

An unacceptable solution is for CHECKPOINT to update chk_ctr *after* a successful transmission; such a strategy opens the possibility that, if power loss occurred between the transmission and the counter update, CCCP would reuse keystream material. A more reasonable solution (which we have not implemented) is to use a separate *commit bit* that is set in nonvolatile memory after both the chk_ctr update and the transmission; this solution avoids both problems mentioned above. Minimizing the energy cost of maintaining a commit bit is an opportunity for hardware optimization.

A number of implementation enhancements are also future work. For instance, shortfalls in over-the-air RFID protocols and a lack of drivers on the WISP make the restore procedure unnecessarily complicated and difficult to implement. We also plan to extend the borders of CCCP from checkpointing towards long-term storage as described in Section 4.2.4.5. Key management makes long-term storage more challenging than checkpointing. Another area for further investigation is modifying the checkpoint function to operate at lower voltages. Writing the counter value to flash memory restricts checkpoints to periods where the available energy can support at least one write to flash memory. Our future work seeks to circumvent these minimum voltages in order to accomplish secure remote storage for CRFIDs whenever their processors have sufficient energy to compute. Finally, for simplicity, CCCP's communication protocol currently addresses only the scenario in which a single tag

communicates with a single reader. We plan to discard that simplifying assumption during further testing in multi-reader infrastructures.

4.8 Summary

CRFIDs enable pervasive computing in places where batteries are difficult to maintain. However, the high energy necessary to erase and write to flash memory makes storage difficult without a constant energy source. CCCP extends Mementos [124] by exploiting the backscatter transmission common on passive RFID systems to remotely store checkpoints on an untrusted RFID reader infrastructure. CCCP protects data with UHF-based MACs, opportunistic precomputation of keystream material for symmetric cryptography, and hole punching to store a counter used to enforce data freshness. Our measurements of a prototype implementation of CCCP on the WISP tag shows that radio-based, remote checkpoints require less energy than local, flash-based checkpoints—despite the overhead of the cryptography to restore the security semantics of local, trusted storage. CCCP gives a CRFID increased storage capacity at low energy cost and enables long-running computations to make progress despite continual power interruptions that destroy the contents of RAM. Moreover, the abstraction provided by CCCP allows application developers to focus on computation rather than space, energy, and security management. Flash memory generally requires a coarse-grained, high-power erase operation before writing a new value. Our hole punching technique allows CCCP to partially reuse unerased flash memory, thus reducing the frequency with which flash memory must be erased.

CHAPTER 5

ZERO-POWER SECURITY FOR IMPLANTABLE MEDICAL DEVICES[1]

This chapter proposes using a transiently powered computer to address the problem of trust for an implantable medical device (IMD)—a deeply embedded, resource-limited device that bears similarities to other electronic devices that are installed in difficult-to-reach places. Starting with the observation that a certain IMD is vulnerable to resource-depletion attacks via its radio, we design *WISPer*, a TPC-based cryptographic authentication mechanism that external parties must convince of their authenticity before they are allowed to communicate with the IMD. WISPer's key feature that protects the IMD's resources is its implementation of a *bring-your-own-power* policy: a party that wishes to authenticate itself must provide WISPer's operating power. The power to perform the authentication does not come from the IMD's finite battery, and a malicious third party cannot keep the IMD awake with its attempts to communicate.

This chapter's contributions are:

- The design and implementation of WISPer, an authentication mechanism that uses a lightweight cryptographic challenge–response protocol implemented on a TPC to prevent an IMD from resource-depletion attacks.

---

[1]This chapter is adapted from papers by Halperin et al. [70] and Burleson et al. [27].

- The design and implementation of a *sensible key exchange* mechanism with which a TPC can convey a cryptographic key to a nearby device through a medium such as skin at a rate of 310 bits per second.

Additionally, we evaluate a modification to WISPer that adds a piezoelement to enable the wearer of a WISPer-enhanced device to detect security-sensitive events that WISPer signals via sound or vibration; the sound volume of WISPer was louder than normal conversation at a distance of 1 m.

Throughout this chapter, we refer to these applications of TPCs as *zero-power* techniques because they use harvested energy and require no power from the device they serve to augment.

5.1 Background

In many embedded systems, energy is a limited resource. In an *energy-depletion attack*, a type of denial-of-service (DoS) attack, an attacker can cause a system to consume energy in excess of normal operating energy, depleting the limited supply.

Several examples arise in the context of sensor networks, where battery-powered nodes communicate via radio and must strive to conserve energy to maximize their utility. In one kind of attack, intentionally mis-crafted packets result in excessive path lengths, even loops, in network routes [163], engaging nodes' energy-intensive radio subsystems. Another form of attack at the MAC layer—the *denial of sleep* attack—can reduce a sensor node's lifetime by orders of magnitude [125]. Alternatively, an attacker can plant software on a mobile device to induce inexplicable battery drain [84].

Nodes in sensor networks may be difficult to access for battery recharging or replacement (being located on treetops or endangered turtles [149]), making energy-depletion attacks an annoyance. However, more-serious consequences may occur in the context of *implantable medical devices* (IMDs) that are inside living tissue. Many

110

IMDs use nonrechargeable batteries because of their consistent, long-lasting energy-delivery performance and to hedge against human error (e.g., an infirm patient forgetting to, or being unable to, charge the battery) [98]. These nonrechargeable batteries have finite lifetimes measured in years, after which time the entire device must be surgically replaced. Recent work has demonstrated that an unauthorized party can keep an IMD's radio awake continuously, which poses a serious risk for the battery's—and therefore the device's—longevity [70, 111].

These vulnerabilities to energy-depletion attacks are symptomatic of a more general problem of *trust*: many resource-limited devices implicitly trust external entities to assign them work to do. Computational—e.g., cryptographic—methods of establishing trust are an appealing way to address this problem, but they require their own resources, opening another avenue for abuse.

5.2   Security and Privacy for Implanted Medical Devices

Implantable medical devices (IMDs) perform a variety of therapeutic or life-saving functions ranging from drug infusion and cardiac pacing to direct neurostimulation. Modern IMDs often contain electronic components that perform increasingly sophisticated sensing, computation, and actuation, in many cases without any patient interaction. IMDs have already improved medical outcomes for millions of patients; many more will benefit from future IMD technology treating a growing number of ailments. Many of these devices are in use worldwide; over 2.6 million cardiac (heart) devices were implanted in patients in the U.S. alone between 1990 and 2002 [97].

Because of their crucial roles in patient health, IMDs undergo rigorous evaluation to verify that they meet specific minimum safety and effectiveness requirements. However, *security* is a relatively new concern for regulatory bodies; bug-averse manufacturers have traditionally had little incentive to add security mechanisms that might cause problems or slow down regulatory approval. Perhaps not surprisingly in light
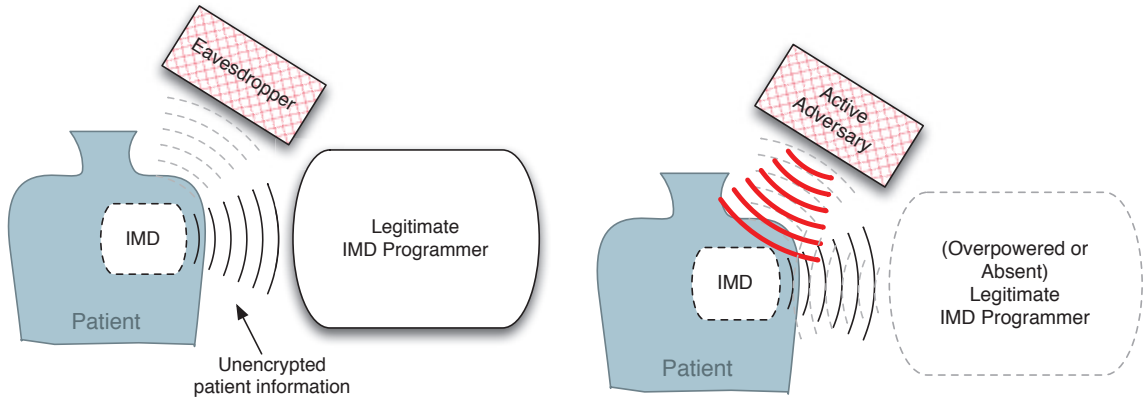
Figure 5.1. Two kinds of adversaries for IMD applications: a passive eavesdropper (left) and an active adversary with a radio (right).

of this situation, recent security research has demonstrated that some IMDs fail to meet appropriate expectations of security for critically important systems.

The key classes of IMD vulnerabilities researchers have identified are *control* vulnerabilities, in which an unauthorized person can gain control of an IMD's operation or even disable its therapeutic services, and *privacy* vulnerabilities, in which an IMD exposes patient data to an unauthorized party. Both kinds of vulnerabilities may be harmful to patients' health outcomes, and both kinds are avoidable.

5.2.1 Threat model

Threat modeling, which entails anticipating and characterizing potential threats, is a vital aspect of security design. With realistic models of adversaries, designers can assign appropriate priorities to addressing different threats.

The severity of vulnerabilities varies along with the sensitivity of the data or the consequences of actuation; there is no "one size fits all" threat model for IMDs. A non-actuating glucose sensor incurs different risks than a defibrillator that can deliver disruptive electrical shocks to a heart.

Adversaries are typically characterized according to their goals, their capabilities and the resources they possess. Security designers evaluate each threat by consider-

ing the value of the target and the amount of effort necessary to access it. Recent work analyzing IMD security and privacy has posited several classes of adversaries, described below and illustrated in Figure 5.1.

An *eavesdropper* who listens to an IMD's radio transmissions, but does not interfere with them, can often learn private information with minimal effort. Such a *passive adversary* may have access to an oscilloscope, software radio, directional antennas, and other listening equipment. Several studies have considered this type of adversary and demonstrated that eavesdropping on unencrypted communications could compromise patients' data privacy [70, 91, 111, 120, 130].

An *active adversary* extends the passive adversary's capabilities with the ability to generate radio transmissions addressed to the IMD, or to replay recorded control commands. Halperin et al. demonstrated that an active adversary with a programmable radio could control one model of implantable defibrillator by replaying messages—disabling programmed therapies or even delivering a shock intended to induce a fatal heart rhythm [70]. Jack and Li have demonstrated similar control over an insulin pump, including the ability to stop insulin delivery or inject excessive doses [130, 91].

Another adversarial capability is *binary analysis*, the ability to disassemble a system's software and in some cases completely understand its operation. By inspecting the Java-based configuration program supplied with his own insulin pump, researcher Jerome Radcliffe reverse-engineered the pump's packet structure, revealing that the pump failed to encrypt the medical data it transmitted or to adequately authenticate the components to one another [120]. In contrast to design-time static analysis of source code, a crucial practice that may expose flaws before devices are shipped [77], binary analysis involves inspecting *compiled* code; it can expose flaws in systems that erroneously depend on the supposed difficulty of reverse engineering to conceal private information.

In the context of medical conditions, it may be difficult to comprehend why a malicious person would seek to cause harm to patients receiving therapy, but unfortunately, it has happened in the past. For example, in 2008, malicious hackers defaced a webpage run by the nonprofit Epilepsy Foundation, replacing the page's content with flashing animations that induced migraines or seizures for some unsuspecting visitors [118]. Although we know of no reports of malicious attacks against IMDs "in the wild," it is important to address vulnerabilities before they become serious threats.

5.3 Zero-Power and Sensible Defenses for IMD Security and Privacy

Providing security and privacy on an IMD involves health risk factors and tight resource constraints. Traditional approaches could potentially introduce new hazards to patient safety. For instance, protecting an IMD with a cryptographic key may provide security, but the unavailability of a key could hinder treatment in emergency situations. Another risk to IMD availability is excessive power consumption by mechanisms other than those needed for the device's primary function. For instance, the energy cost of performing computation for cryptography or radio communication could directly compete with the energy demands of pacing and defibrillation. Effective mechanisms for security and privacy should not provide new avenues for an unauthorized person to drain a device's battery. For instance, spurious wake-ups or a cryptographic authentication process itself could cause a device to enter a state that consumes excessive amounts of energy (as in, e.g., the *sleep deprivation torture* attacks of Stajano and Anderson [151]).

Therefore, three goals guided our design of zero-power approaches for IMD security and privacy. First, an effective approach should either prevent or deter attacks by both malicious outsiders with custom equipment and insiders with commercial

114
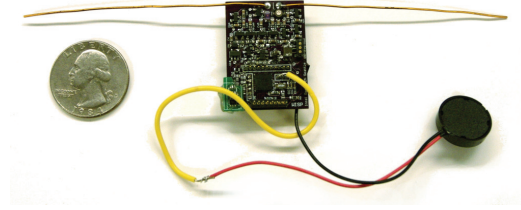
Figure 5.2. The WISP with attached piezo-element.

programmers. Because IMD therapies rely on long-lasting batteries, a second goal
is that security and privacy should draw no power from the primary battery, thus
preventing denial of service attacks on power. Third, security-sensitive events should
be effortlessly detectable by the patient. We must also ensure that new security
mechanisms do not introduce new failure modes.

We do not claim that our defenses are final designs that IMD manufacturers
should immediately incorporate into commercial IMDs. Rather, we believe that our
research establishes a potential foundation upon which others can create, evaluate,
and implement new defensive mechanisms for future IMD designs.

5.3.1   Detection: Zero-Power Notification for Patients

As earlier work notes [68], it may be possible to deter malicious activities by
making patients aware of those activities. Our zero-power notification alerts a patient
to potentially malicious activities both by insiders using commercial programmers
and by outsiders using custom attack hardware, thereby making patients effortlessly
aware of remote communications. On some modern ICDs, triggering a magnetic
switch inside the ICD causes the ICD to beep. Whether intentional or not, such
beeping represents a step towards the concept of patient awareness by way of audible
alerts. But beeps triggered by a magnet alone do not raise patient awareness for
RF-initiated actions, which our approach does.

Our prototype of zero-power notification, *WISPer*, wirelessly drives a piezo-element
that can audibly warn a patient of security-sensitive events. WISPer builds upon revi-

sion 1.0 of the Wireless Identification and Sensing Platform (WISP) [139], a postage-stamp-sized embedded system that contains RFID circuitry and a Texas Instruments MSP430F1232 microcontroller with 256 bytes of RAM and 8 KBytes of flash memory. The WISP harvests energy from a 915 MHz RF signal generated by the Alien ALR-9640 nanoscanner, a UHF RFID reader running the EPC Class 1 Gen 1 protocol. Although we prototyped at 915 MHz, it may be possible to create similar hardware that operates at the frequency of current ICD programmers. WISPer adds to the WISP's base code a 30-line C program that activates a piezo-element which we attached to the general-purpose I/O (GPIO) ports of the WISP. After WISPer receives a sequence of wireless requests from the RFID reader, it emits constant chirping, thereby informing the patient of the wireless interaction. A future version of WISPer could set a separate GPIO high after buzzing for a certain number of cycles, and the IMD could allow remote communications only after that GPIO is raised. WISPer satisfies our zero-power notification design constraints: it draws no energy from a battery and can issue alerts for all reprogramming activity.

Two measurements quantify the effectiveness of the WISPer prototype for zero-power notification. We used a sound level meter to measure Sound Pressure Level (SPL) with a reference pressure of 20 micropascals (the standard for above-water calculations). The buzzing volume peaked at 67 dB SPL from a distance of 1 m. For reference, a normal conversation is about 60 dB SPL and a vacuum cleaner at a distance of 3 meters is about 70 dB SPL [96]. We then placed our prototype in an environment designed to simulate implantation in a human (Figure 5.3; we called this experimental apparatus "Alice" after the protagonist of cryptographic protocol descriptions). We implanted WISPer beneath 1 cm (a standard ICD implantation depth) of bacon, with 4 cm of 85% lean ground beef packed underneath. Note that Section 5.3.4 suggests a more-appropriate setup for tissue simulation.

Figure 5.3. "Alice." To simulate implantation in a human, we placed WISPer in a bag containing bacon and ground beef (left and middle). **This method of tissue simulation is deprecated.** Section 5.3.4 suggests a preferable method of tissue simulation (right).

We took several readings at the surface of the bacon in order to ascertain the effects of obstruction by tissue. We measured 84 dB SPL of sound at the surface of the tissue, and subjectively were easily able to hear it from a meter away (more than the distance between standard ICD implantation sites and a patient's ear).

These tests of our prototype device suggest that its piezo-element is audible under reasonable simulations. Because malicious attackers may attempt their attacks in noisy, chaotic environments to vitiate auditory notification, and because some patients with ICDs may have limited hearing, we note that a piezo-element can be used to produce vibration instead of audible sound. In our experiments, the 4 kHz alert used was easily sensed by touch.

5.3.2 Prevention: Zero-Power Authentication

Our second defense implements a zero-power method that allows an IMD to verify that it is communicating with a real commercial programmer (and not an unauthorized software radio programmer).

The device implements a simple challenge-response protocol (Figure 5.5) based on RC5-32/12/16 [129]. In this model, all commercial programmers know a master key K_M, each IMD has an serial number or identity I, and each IMD has an IMD-specific key $K = f(K_M, I)$, where f is any cryptographically strong pseudorandom

function (such as AES). The value K_M should be stored in secure hardware on the programmers. The protocol works as follows. The programmer transmits a request to authenticate to WISPer. WISPer responds with its identity I and a nonce N. The programmer computes $K = f(K_M, I)$ to get the IMD-specific key and then returns the response $R = \text{RC5}(K, N)$ to WISPer. WISPer computes the same value and verifies the value it received from the programmer against its result. WISPer finally sets a GPIO high which, if attached to or built into a real IMD, would inform the IMD that WISPer successfully authenticated a programmer.

For the sake of simplicity, our prototype does not implement the full protocol. Namely, in our experiments we use a fixed nonce and assume that the programmer knows the nonce in advance. Using this simplified model, we experimentally verified that, upon receiving the programmer response R, WISPer was able to perform its own RC5 encryption and verify equality. We were able to run this subset of the protocol with complete reliability using only harvested energy. To lift from the simplified model to a real implementation of our protocol, we note that the nonce should appear random to an adversary. Since we, and others [31], show that it is possible to run RC5 on a WISP, a natural solution would be to generate the nonce with RC5 in counter mode. A better approach that would yield a truly random nonce is to exploit process variations and omnipresent thermal noise by extracting random bits from SRAM using the FERNS technique of Holcomb *et al.* [72]. Applying FERNS to 256 bytes of SRAM could yield 100 bits of true randomness each time the SRAM is powered up. Our work would benefit from an implementation of the memory-as-TRNG technique on the WISP.

We learned from our successful attacks that private data transmitted between our ICD and programmer are not encrypted. We propose that cryptography be added at least at critical junctures. Encryption of the entire conversation would be optimal—for example, a secure channel between programmer and ICD could prevent third-

party disclosure, replay, and many other attacks—but in the interest of modularity we consider in this chapter only defensive approaches that might be implemented with less extensive modifications to current ICD designs. Modularity aside, if we were to propose cryptographic extensions that required significant changes to ICD design, it would be necessary to consider the power cost of our proposed changes. Without detailed knowledge of the inner workings of ICDs, however, we cannot accurately assess the cost of adding cryptography to existing devices.

The tension between increased security and increased power consumption can be resolved by requiring successful zero-power authentication *before* the device switches to higher power consumption modes. Our prototype shows that this proposal is feasible for bootstrapping stronger (and possibly mutual) authentication methods. Our prototype harvests power from RF transmissions, performs a cryptographic authentication, and on successful authentication of a programmer, sets a GPIO high which, if connected to or built into a real ICD, would permit the ICD to participate in active RF communication and other higher-level protocols. This approach addresses the risk of *sleep deprivation torture* described by Stajano and Anderson [151].

This chapter does not address the well-known problem of key management. Using a shared secret (called K_m above) is reasonable for a prototype implementation, but a large-scale deployment of shared key material—in implanted devices, hospitals, clinics, ambulances, IMD programmers, and so on—may pose an unacceptable risk because of the ease with which an unauthorized party could decrypt transmissions upon obtaining the key material. (Though our recommendation of storing K_m in secure hardware does partially mitigate this risk under certain threat models.) The simple scheme described above also fails to address revocation of privilege and is therefore ill-suited to situations in which key material might be compromised, although the proposed system is still no less secure than the open-access model of conventional systems. An SKEYS [67] or key-regression [55] approach, with periodic updates of
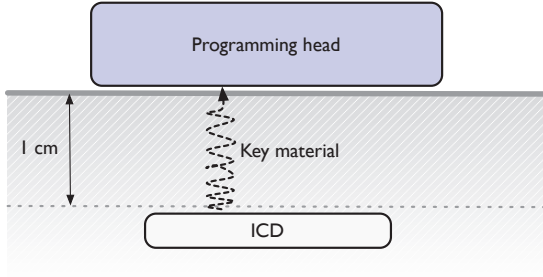
Figure 5.4. Zero-power sensible key exchange: a nonce is transmitted from the ICD to the programmer using acoustic waves. It can be clearly picked up only if the programmer is in contact with the patient's body near the implantation site, and can be used as the secret key in the authentication protocol from the previous section. (1 cm is a typical implantation depth. Diagram is not to scale.)
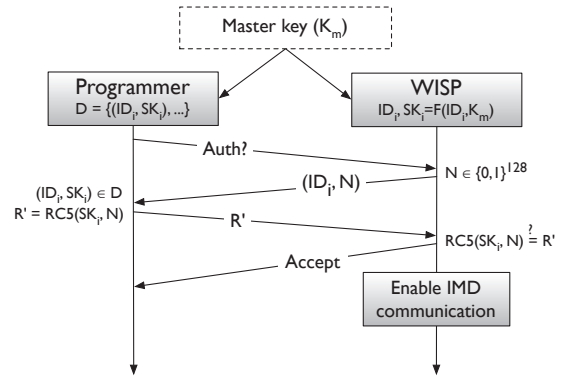
Figure 5.5. The protocol for communication between an ICD programmer and a zero-power authentication device (a WISP RFID tag, in the case of our prototype).

programmer keys, might mitigate the time-window in which an attacker can use compromised keys while also not significantly changing the overall model. Furthermore, the offline nature of the transactions that must be secured—imagine an ambulance reaching an ICD patient in a remote setting—further complicates the problem of key management and revocation.

In the context of medical devices, security-related design choices must balance security, privacy, safety, and efficacy [68]. An ideal key management scheme for this context, which we present as an important open problem, must provide security and support privacy without hindering the operation of medical devices that are already known to provide safe and effective treatments.

5.3.3 Zero-Power Sensible Key Exchange

We now present a key-distribution technique that complements both of our previous defensive techniques: distribution of a symmetric cryptographic key over a

human-perceptible sensory channel. The primary goal is to allow the patient to detect a key exchange while it occurs.

The programmer initiates our protocol by supplying an unmodulated RF carrier signal that could power the passive component of the IMD. The IMD then generates a random value to be used as a session key and broadcasts it as a modulated sound wave. The amplitude of this sound wave is such that it can be easily received and demodulated by a reader with a microphone in contact with the patient's body near the implantation site, but it cannot be heard over background noise at any appreciable distance from the patient, at least not without dedicated sensing equipment. The close proximity this enforces further ensures patient awareness and consent to the authentication attempt. Once key exchange has been performed, RF communication can safely occur over a longer range without fear of eavesdropping.

We implemented our key exchange mechanism on WISPer using as carrier frequency the same 4 kHz audible and tactile signal discussed above. To effect key exchange, we used the same modulation scheme currently in use by our reader (2-FSK). We achieved a baud rate of 310 Bd, permitting transmission of a 128-bit nonce in 0.4 s. The components performed key exchange without drawing power from a battery, and the exchange was clearly audible, measuring 75 dB SPL through a human hand. When the microphone was not in contact with the skin, the sound pickup was too low to be measured on our meter (< 60 dB SPL). In our *ad hoc* experiments, transmission of the key was easy to feel with the hand, but difficult to hear at a distance. While these preliminary measurements show the plausibility of making eavesdropping difficult, further work is necessary to illuminate the relationship between sound levels and the ability to eavesdrop. Furthermore, an adversary may attempt to eavesdrop on the electromagnetic emanations [87] of the electrical components that generate the sound rather than on the sound itself. Radio shielding in the form of a Faraday cage or use of non-electromagnetic, optical links between security-sensitive modules may

help to reduce these unintended emanations. An alternate approach for sensible key exchange might be for the programmer to transmit the key to the IMD over an audio channel, or for the final key to be derived from keys sent in both directions.

5.3.4 Note on Tissue Simulation

Evaluating hardware or software modifications to (or interference with) IMDs poses a challenge to researchers who usually design for open-air transmission. The ideal evaluation environment is human flesh, but human subjects may be unwilling to receive prototype implantable devices, and arranging for human testing typically requires extensive paperwork and arduous board approval. Animal subjects (e.g., pigs) may anatomically resemble humans, but they are difficult to obtain, house, and maintain, and implanting devices in animals poses ethical concerns. Cadaverous and other nonliving tissue such as store-bought beef and bacon [69, 58] is not a widely accepted substitute electromagnetically for live tissue. In cardiologist parlance, "Dead meat don't beat."

Fortunately, conforming to electromagnetic compatibility (EMC) standards that *do* provide a reasonable tissue-simulation environment is within reason for nonmedical researchers. The ANSI/AAMI PC69 standard [5, 6] defines a testing setup for EMC testing of cardiac devices. Work incorporating a PC69-compliant setup includes investigations of pacemaker interference by RFID readers [142], MP3 players [10], walk-through metal detectors [79], and cellphones [133]. The U.S. Food and Drug Administration (FDA), the agency that regulates medical devices, uses a calibrated saline-bath *phantom* (tissue simulation environment) to study RF interference with IMDs. Seidman et al. refer to the FDA's preferred phantom as "a modified version of the ANSI/AAMI PC69 Standard" [142].

Because it electromagnetically approximates a human torso, a PC69-compliant saline bath is a suitable environment for testing electronic devices that are to be

embedded in tissue. Based on the description by Seidman et al., we built our own calibrated saline bath prototype (shown in Figure 5.3) with approximately $30 of parts from a hardware store, a scale, distilled water and table salt.[2]

Simulation with a standardized setup offers a controlled environment with a minimum number of confounding variables. Such an environment is designed to produce reproducible results, which is a major challenge for open-air and *in vivo* experimentation. This reproducibility and experimental control allow for comparison with analytical models and reasoning about how the system under test will behave when implanted.

5.4   Related Work

In addition to the work mentioned at the beginning of this chapter, other work has studied the problem of security for IMDs.

Halperin et al. propose a set of design goals and research directions for security and privacy mechanisms for IMDs, noting the tensions that pose design and operational challenges [68]. They mention battery depletion as a specific threat and propose shifting computation to external resources. A follow-up paper (on which this chapter is based) expands these ideas by implementing several attacks and the WISPer defense [69].

Paul et al. point out challenges and solutions that apply especially to insulin pumps [111], one kind of IMD that has both internal (to the body) and external components, including a user interface for patients. They suggest that energy-depletion risks can be mitigated with a patient-facing button that allows the radio to be active for a small amount of time. This approach may work even for IMDs that are fully implanted, provided that a button mechanism on an implanted device can be made

---

[2]A manuscript describing experiments with this PC69-compliant saline bath is in preparation.

biocompatible and that pressing a button through tissue would not damage the tissue. The techniques in this chapter address the same problem in a different manner, albeit one that does not require user interaction—which may allow them to generalize other devices that are not in direct contact with people.

Authentication with an energy-harvesting gatekeeper device like WISPer is most appealing when the time and energy cost of cryptography are small enough that the interaction is fast and unintrusive. Performing cryptographic authentication on a TPC necessitates the use of ciphers that are lightweight—i.e., can fit in limited memory and execute quickly enough at the slow clock rates that support low-power operation. Kerckhof et al. ably summarize and compare a variety of ciphers from this perspective [83].

5.5 Summary

Implantable medical devices (IMDs), like many other kinds of embedded devices, suffer from resource limitations that make excessive computation a concern. This chapter presents a medley of mechanisms designed to protect an inaccessible, resource-limited device like an IMD from radio-based resource-depletion attacks. These "zero-power" mechanisms include WISPer, a TPC-based cryptographic authentication system that requires external parties to pay the power cost of computation, and a key-exchange mechanism that works through biological tissue.

CHAPTER 6

AUGMENTING MEDICAL DEVICES WITH TRANSIENTLY POWERED COMMUNICATION

This chapter presents the design and implementation of Noradio,[1] a transiently powered communication system for an implantable medical device (IMD). Noradio aims to address the risk of excessive power consumption by an IMD's radio subsystem— an increasingly appealing optimization opportunity as IMDs become more networked to improve patient monitoring.

6.1 Implantable Medical Devices and Power

Power consumption is a critical design consideration for embedded devices [53], particularly those that are unavailable for maintenance or battery recharging. Many implantable medical devices (IMDs) perform life-supporting functions from nonrechargeable batteries.[2] Excessive power consumption may put an IMD's longevity in jeopardy. (Chapter 5 describes a particularly alarming kind of *intentional* excessive power consumption.)

Thanks to advances in home monitoring and automatic electronic therapy delivery, the trend in healthcare is toward ever more energy use, data collection, and device connectivity, even for fully implanted devices. For example, all leading makers of

---

[1]Apologies to Marcel Molina, Jr. (`@noradio` on and at Twitter).

[2]IMDs that are fully implanted in the body, such as pacemakers and implantable cardiac defibrillators (ICDs), use nonrechargeable batteries with specially formulated chemistry that are designed to support up to 10 years of implantation [98]. Doctors typically surgically replace such an IMD after a predetermined time period, or when its *elective replacement indicator* (ERI) triggers an alarm.

implantable cardiac defibrillators offer at-home monitoring and periodic data uploads to clinical databases, which enables doctors to respond to clinically relevant signs of trouble before they become severe [40].

Communication interfaces for IMDs improve treatment outcomes by giving doctors advance warning, but they raise a design issue: radios are energy hungry. Radio communication often dominates embedded systems' power consumption during times when the radio is on (either listening or sending). For example, Fonseca et al. profiled a Hydrowatch mote [47] and measured the TI CC2420 2.4 GHz radio's current consumption to be $39.4\times$ that of the TI MSP430F1611 microcontroller in its highest-power "active" computing mode [53].

According to a 2004 survey article, "a cardiac pacemaker" at the time of that publication "uses half of its battery power for cardiac stimulation and the other half for housekeeping tasks such as monitoring and data logging" [98]. According to a semiconductor manufacturer, the radio-communication portion of a pacemaker's power budget is approximately 15% [134]. As the trend toward greater connectivity continues, the total fraction of battery energy used for radio communication is likely to increase.

This chapter explores the idea of removing the radio from an IMD altogether. We call our platform for evaluating this idea *Noradio*. Noradio extends the design of WISPer—an authentication mechanism for IMDs, described in Chapter 5—by replacing the radio subsystem of an IMD with a computational RFID (CRFID) that communicates via backscatter (reflective) radiation without using any battery power. Noradio's overall design goal is to reduce the amount of energy used for IMD communications. To evaluate Noradio, our quantitative goal is to measure the difference in energy consumption when a CRFID replaces an active radio. We build versions of Noradio both *with* and *without* an active radio circuit, measuring the energy consumption of each during communication.

6.1.1 Contribution

This chapter presents Noradio, a proof-of-concept prototype for a TPC-augmented embedded system, as well as an evaluation of Noradio's energy consumption. The results validate our approach, suggesting that replacing a radio with a CRFID reduces the energy required for communication.

The key design challenge of implementing Noradio is translating the synchronous communications of a system built to use an active radio—one that can transmit at any time—to the asynchronous communications that are feasible with a CRFID that lacks an active radio.

Although this work explores the repercussions of removing an active radio component from a specific kind of device, our techniques can generalize to other kinds of devices that currently use active radio components.

6.2 Emulating a Pacemaker

A cardiac pacemaker continuously monitors a patient's heartbeat, periodically issuing small pulses of electricity to stimulate heart tissue to beat at a steady rhythm [167]. The body includes a natural electrical pacing system to perform that function, so electromechanical devices implanted in patients are also called *artificial* pacemakers; we refer to them here as simply *pacemakers*.

The designs of commercially viable IMDs such as pacemakers are closely held trade secrets. Various academic projects, notably those responding to McMaster University's Pacemaker Challenge [148], have aimed to build functionally valid models of pacemakers that benefit from state-of-the-art techniques in formal methods. To aid in this effort, Boston Scientific has released functional specifications for the behavior of a previously available model of pacemaker [16]. In contrast to this formal approach, for the design of our pacemaker emulator we abstract away most of the clinical functions in order to focus on the communication subsystem.

Noradio performs both sensing and "pacing"[3] activities in addition to communication. In cardiology nomenclature, Noradio implements *VVO* mode, indicating single-channel pacing (the first "V"), single-channel sensing (the second "V"), and no inhibition of pacing based on the sensed activity. VVO mode is not used in clinical settings, since ill-timed pacing pulses can trigger unsafe heart conditions [166], but for Noradio it serves as a suitable test mode that avoids the need to attach a cardiac simulator.

To implement both versions of Noradio, we chose a Texas Instruments MSP-EXP430F5438 evaluation board. This board features a TI MSP430F5438A microcontroller with 16 KB of SRAM and 256 KB of flash memory. Key to our design goal, this evaluation board bears a slot for a removable radio daughterboard. We use a TI CC2500 radio daughterboard that operates in the 2.4 GHz industrial, scientific, and medical (ISM) band. A typical pacemaker might use a custom purpose-built CMOS radio consuming approximately 5 mA of current when sending or receiving, and three to four orders of magnitude less when doing neither—which is its normal state, since designers target low duty cycles to conserve energy [134]. The CC2500 radio consumes approximately three times as much current as this custom radio while transmitting and receiving, and within a factor of two while sleeping [157]. Pacemakers may also use a separate wakeup circuit in the 2.4 GHz range (in which regulations allow higher transmit power) [134]. Table 6.1 compares our emulated pacemaker to typical pacemaker specifications.

Typical pacemaker	Our emulator	Notes
< 10 KB RAM*	16 KB SRAM	
< 100 KB NVRAM*	256 KB flash memory	Contains program memory
(varies)	8 MHz CPU clock rate	
405 MHz + 2.4 GHz radios†	2.4 GHz radio	Removable daughterboard

Table 6.1. Comparison of our emulator to typical commercial pacemakers. Because the designs of cardiac devices are closely held secrets, the entries marked with ⋆ are estimates based on a 1999 book on real-time systems [44]. The radio specification (marked with †) is based on public information from a semiconductor manufacturer that serves IMD manufacturers [134].

6.2.1 Baseline: Radio-Based Pacemaker Emulator

We prototyped a simple clinical communication system wherein our emulated pacemaker periodically sends telemetry[4] to a collecting node that emulates a clinical programmer (the pacemaker-adjustment device in a doctor's office). We designed a simple peer-to-peer telemetry mechanism based on TI's SimpliciTI protocol, version 1.2.0 [158]. SimpliciTI is a simple packet-based protocol stack that implements link and network layers. Noradio's telemetry mechanism operates as follows (also see the block diagram in Figure 6.2):

- Every 200 ms, the emulated pacemaker uses its onboard 12-bit analog-to-digital converter (ADC) to sample the voltage on one of its analog inputs (resulting in a 5 Hz sample rate, enough to capture low-frequency components of a 1–2 Hz heartbeat).

- Immediately after sensing, the emulated pacemaker pulses a general-purpose input/output (GPIO) pin for 0.5 ms (following the Boston Scientific pacemaker

---

[3]We did not test Noradio on tissue; doing so would not prove anything about the energy required for communication.

[4]Transliterated, *remote measurements*—sensed values sent from one place to another.
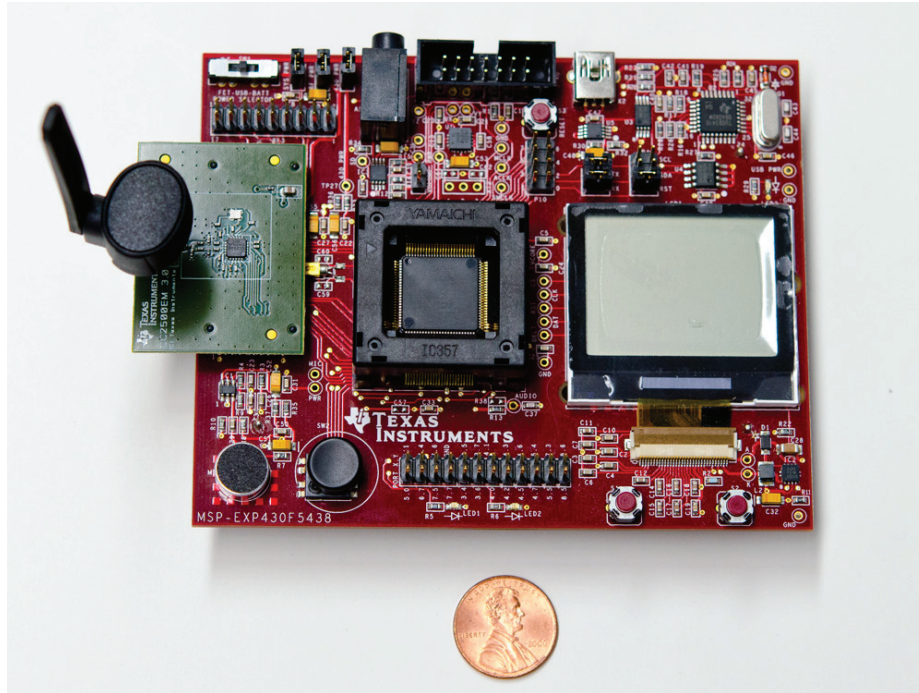
Figure 6.1. Photo of an MSP-EXP430F5438 board with a CC2500 2.4 GHz daughterboard implementing the radio-equipped version of our emulated pacemaker.

specification [16] for on-demand pacing). The pin is connected to a $1\,\mathrm{k\Omega}$ resistor to approximate heart tissue [141].

- After the output pulse, the emulated pacemaker assembles a packet containing the 12-bit ADC value and transmits it via radio to the clinical programmer emulator.

- The clinical programmer emulator prints each received reading to a PC screen via its serial port.

The clinical programmer emulator can also send commands to the emulated pacemaker via the same protocol. These command packets trigger interrupts on the emulated pacemaker, which updates its state to obey the commands and sends an acknowledgement packet.
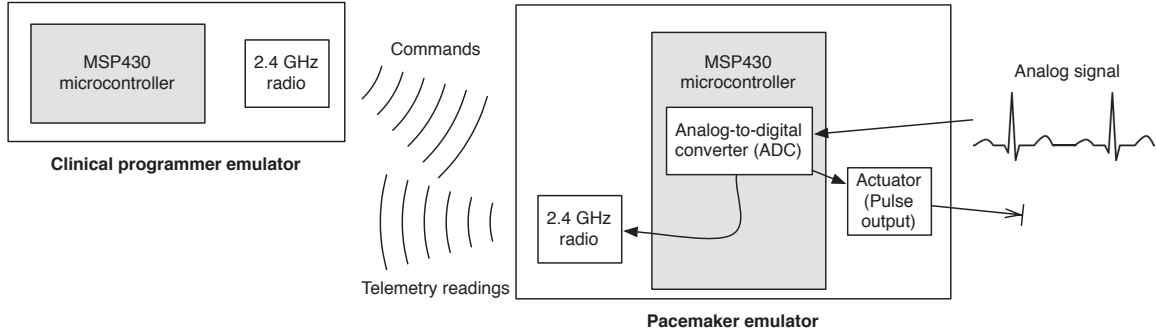
Figure 6.2. Block diagram of the *radio-equipped* version of an emulated pacemaker. The pacemaker reads an analog signal (e.g., a heartbeat) and transmits a representation of it to an emulated clinical programmer.

6.2.2 CRFID-Augmented, Radioless Pacemaker Emulator (Noradio)

We built a radioless version of our pacemaker emulator in order to evaluate the difference in energy consumption. This implementation effort required removing the radio daughterboard, choosing an appropriate interface to connect Noradio to a UMass Moo CRFID, and rewriting the synchronous communication protocol as an asynchronous protocol. Figure 6.3 depicts our Noradio prototype.

Unlike the radio-equipped version, Noradio does not have a "live" telemetry mode, because a CRFID cannot transmit information at arbitrary times. Instead, the CRFID can operate only when it receives enough power from a nearby RFID reader. Accordingly, Noradio saves samples in its RAM and emits them to the Moo only when the Moo indicates that it is ready to transmit readings. This process works as follows (also see Figure 6.4 for a block diagram of Noradio):

- As before, every 200 ms, Noradio uses its onboard 12-bit analog-to-digital converter (ADC) to sample the voltage on one of its analog inputs.

- Also as before, Noradio pulses a general-purpose input/output (GPIO) pin for 0.5 ms. The pin is connected to a 1 kΩ resistor.
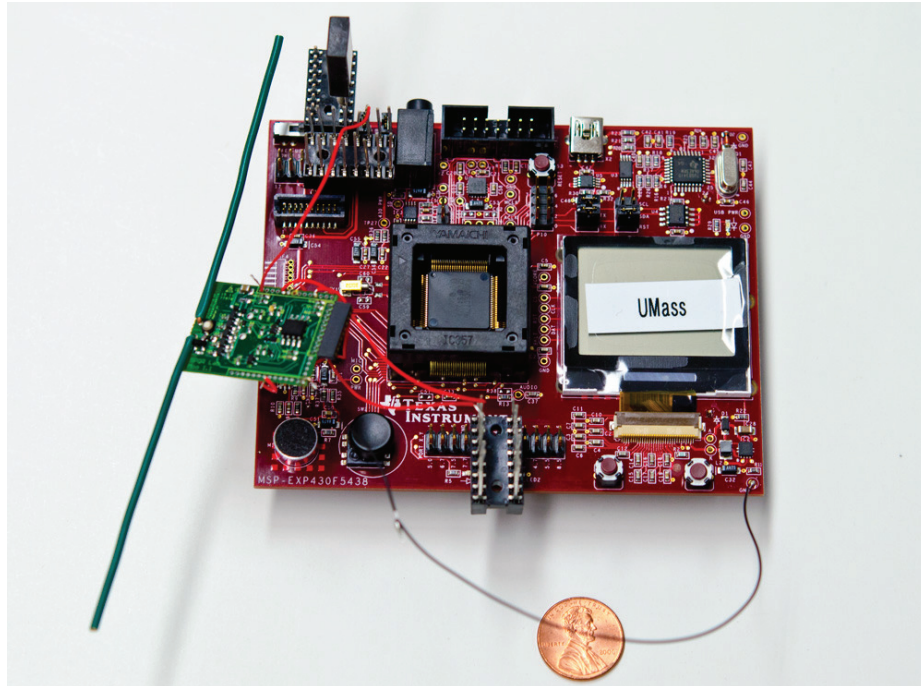
Figure 6.3. Noradio, the Moo-augmented version of the emulated pacemaker.

- Instead of transmitting the sensed value immediately, Noradio saves it in a circular buffer.

- When the Moo receives a query from an RFID reader, it raises a GPIO pin that is connected to GPIO on Noradio via a wire. This triggers an interrupt on Noradio.

- Responding to the interrupt, Noradio drains the circular buffer of sensed values to the Moo via the Serial Peripheral Interface (SPI) bus, one 16-bit word at a time.

- The Moo packs these sensed values into the *EPC* field of its replies to the RFID reader. (Flit, a bulk-transfer protocol for CRFIDs [63], uses the same field to carry bulk messages.)
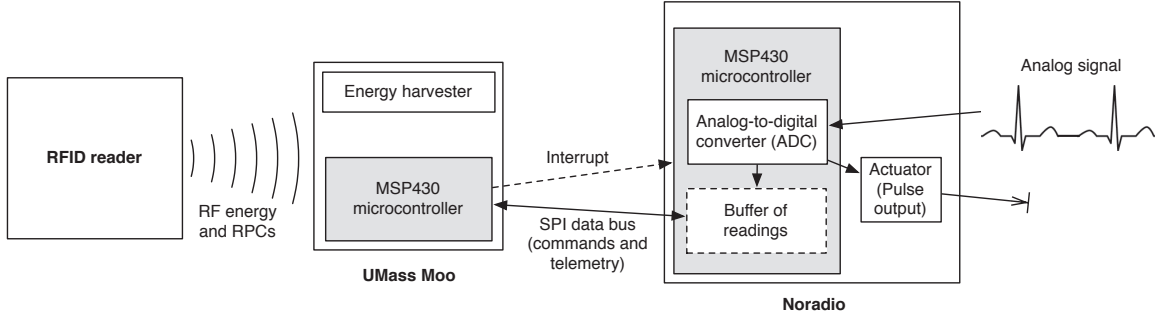
Figure 6.4. Block diagram of the *radioless* version of the Noradio prototype. When an RFID reader provides power, a UMass Moo CRFID triggers an interrupt on the Noradio board's CPU, causing it to read commands from the Moo. In response to an appropriate command, Noradio drains its buffer of stored ADC readings over a data bus to the Moo.

Lacking a reliable communication mechanism for reader-to-CRFID communication,[5] we did not implement a mechanism by which the RFID reader can issue commands to Noradio via the Moo. A future reliable protocol for reader-to-CRFID data transmission, such as BAT [105], would serve as an appropriate mechanism to convey such commands.

6.3   Evaluation

The goal of our evaluation is to compare the energy use of the emulated pacemaker with and without a radio board.

We replaced the "system power" jumper on each MSP-EXP430F5438 board with a 1.4 Ω sense resistor, then attached the high-impedance probes of a Tektronix MSO5204 oscilloscope to the legs of the sense resistor to measure the entire board's current consumption (via Ohm's Law, $I = V_{\text{observed}}/R = V_{\text{observed}}/1.4\ \Omega$).

---

[5]We noticed unpredictable data corruption when attempting to carry data in user-definable fields of reader-to-tag EPC messages with our Impinj Speedway RFID readers.

For each version of the system—the radio-equipped version and Noradio—we measured current consumption as follows. First, we turned on the MSP-EXP430F5438 board (thereby starting the pacemaker emulation) and did not initiate telemetry communications. We verified with the oscilloscope that the board was emitting pacing pulses. We observed the current consumption in this "unlinked" (not sending telemetry) state over a 5-second window. Next, we initiated telemetry communications. For the radio-equipped emulated pacemaker, we turned on the emulated clinical programmer and observed that the link state changed (via a blink of an on-board LED). For Noradio, the CRFID-equipped version, we queried the attached Moo CRFID with an Impinj Speedway Revolution RFID reader emitting queries at 30 Hz at its maximum transmit power of 30 dB and a distance of 50 cm. We verified on the PC controlling the RFID reader that the reader was receiving the values sensed on the Noradio board.

We ran each scenario five times, computing the average current consumption from each oscilloscope trace; the aggregate results appear in Table 6.2.

	Radio	Noradio
Unlinked	24.97 ± 0.26 mA	19.92 ± 0.08 mA
Linked	36.28 ± 0.15 mA	20.30 ± 0.14 mA

Table 6.2. Current consumption of the radio-equipped emulated pacemaker versus that of Noradio, when not communicating with the emulated clinical programmer (*Unlinked* row) and when communicating with it (*Linked* row).

According to a public presentation by a pacemaker manufacturer, a typical pacemaker battery contains about 1.5 ampere–hours of energy [150]. If the current consumption of the pacemaker were as measured above, the battery life from full charge to complete discharge would be $t = E/I = 1.5 \,\mathrm{Ah}/I$ (Table 6.3).

Noradio's energy savings come mainly from the removal of the active radio. A pacemaker, like our radio-equipped emulated pacemaker, includes wakeup circuitry that listens for radio communication. Noradio removes the need to listen for commu-

134

	Radio	Noradio
Unlinked	60.07 h	75.30 h
Linked	41.34 h	73.70 h

Table 6.3. Projected battery life of both the radio-equipped and Noradio versions of the emulated pacemaker with a 1.5 Ah pacemaker battery.

nication, instead saving sensed data locally until an interrupt arrives from a CRFID. The energy cost of transmitting bits via the SPI bus, as Noradio does, is also lower than the energy cost of transmitting them via radio.

1.5 Ah pacemaker batteries are, of course, designed to accommodate infrequent radio communications and low-power on-board ASICs instead of general-purpose microcontrollers like the MSP430. However, the prototype's results for telemetry in Table 6.2 and Table 6.3 suggest that performing communication via an attached CRFID may offer significant energy savings that can extend battery lifetime.

6.4   Related Work

Recent work on "leadless" pacemakers—which differ from conventional pacemaker designs in that they do not require long, difficult to implant, potentially brittle wires to run into the heart—has resulted in a new crop of smaller pacemakers that sit directly on cardiac tissue.

Medtronic has publicized its work on a leadless pacemaker the size of a grain of rice, noting that they were still in search of a way to power it [146]. A forthcoming leadless pacemaker from EBR Systems features a similar form factor for the device that delivers pacing; it harvests energy from ultrasound transmissions emitted by a larger device that is implanted in the chest [60]. Details on the device and its power properties—e.g., whether the ultrasound channel provides both power *and* data—are scant, but it is reasonable to expect that the wireless interface will be powered via the ultrasound transmissions as well. Noradio is meant to serve as a proof of concept to

illustrate that the radio functions of a pacemaker, such as telemetry, can be handled via augmentation with a CRFID. Removing components that are energy hungry may help designers further miniaturize these new IMDs. Alternatively, it may allow the same devices to last longer before replacement.

In addition to the ultrasound-powered leadless pacemaker mentioned above, recent research has revisited the idea that IMDs can be powered from *outside* the body. Kim et al. performed a numerical study of power transfer from outside the body to a cardiac implant and determined that a pacemaker could be powered via a 1 mm coil antenna [85, 154]. The same group designed a system to power a locomotive implant that can swim through blood or other fluids, propelling itself using Lorentz forces while harvesting energy from an external supply [113].

Wong et al. design an integrated pacemaker circuit that requires only 8 μW in operation [168]—minuscule in comparison to the milliwatts a typical "low-power" radio requires (e.g., the CC2500 radio we used in the Noradio prototype requires at least 20 mW to transmit [157]—three orders of magnitude more than Wong et al.'s pacemaker chip).

6.5   Future Work

The results in Section 6.3 are only a preliminary indication that removing the active radio from an emulated pacemaker may reduce overall energy consumption and extend battery life. While we strove to meet the functional parameters of a pacemaker outlined in public documents [16], many fundamental design details differ. Real implantable cardiac devices commonly use custom low-power components—from chips to batteries and even capacitors—that the manufacturers make in their own facilities. We used off-the-shelf hardware with general-purpose components such as a programmable microcontroller and radio, which partially explains the vastly different projected battery life of our emulated pacemakers (Table 6.3). However, our choice

to use off-the-shelf components offers an advantage: it resembles low-cost consumer devices that are made en masse and require frequent battery changes. Noradio's approach to reducing operating energy may apply to these devices.

A sample rate of 5 Hz suffices to capture such metrics as pulse rate (in a normal range) accurately, but it is not sufficiently high to capture all of the physiologically interesting signals that an ECG can convey; a better rate is 20 Hz to capture signals up to 10 Hz [37]. An informative experiment would be to increase the sample rate and measure the energy consumption as the sample rate increased. The analog-to-digital converter on our evaluation boards' MSP430F5438A chips can sample at up to 200 kHz, suggesting that a small increase in sample rate would have a negligible effect on energy consumption from the ADC; however, the energy cost of transmitting more frequently via the radio would likely be noticeable. This difference may have implications for the simple sample-and-transmit strategy that Noradio currently uses.

6.6 Summary

Implantable medical devices (IMDs) are becoming increasingly connected to systems outside the body, resulting in proactive patient monitoring that improves therapeutic outcomes. IMDs that are fully implanted are usually powered by nonrechargeable batteries and use a radio to communicate with clinical systems and at-home monitoring setups. The increasing connectivity of these devices results in more radio use, which results in greater demand on the battery. This chapter asks: what if we eliminated the radio's portion of an IMD's energy budget?

We develop a system called Noradio to evaluate the idea of replacing an active radio with a transiently powered auxiliary device, namely a UMass Moo computational RFID, that provides asynchronous communication. We implement two versions of an emulated pacemaker: one with an active radio, and one without (hence "Noradio"), and keep all other parts of the system identical. Using CRFID-augmented

communication for telemetry resulted in a 44% reduction of the entire board's current consumption versus the radio-equipped version. Using similar techniques on a real pacemaker may significantly reduce the 15% portion of the battery's power that is budgeted for telemetry.

Noradio serves as a proof of concept to illustrate that augmenting parts of a system with transiently powered computers can result in energy savings.

CHAPTER 7

CONCLUSIONS

The key idea of this thesis is that software techniques can make energy harvesting a practicable way to power computing devices. These techniques will make energy harvesting a viable means to perpetually power some embedded devices that currently use batteries. In the long term, the same principles may be applied to make computing technology more robust in areas of the world that lack a reliable or accessible power infrastructure.

This thesis presented Mementos, a system that automatically spreads long-running computations over multiple bursts of execution whenever there is sufficient energy. I evaluated Mementos in a trace-driven simulator that augments an existing cycle-accurate simulator with a notion of energy. Observing that operating solely on harvested energy introduces implicit trust that is undesirable, I described CCCP, a system that safely outsources storage to the potentially malicious harvesting in-frastructure. I described WISPer, an authentication and notification mechanism to protect patients who use implantable medical devices. Finally, I presented Noradio, a prototype emulated pacemaker that saves energy by replacing synchronous radio communications with asynchronous TPC-augmented communications.

Opportunities for future work include: exploring the architectural and operating-system implications of operating on harvested energy; building new hardware that implements the TPC model; and designing ubiquitous-computing environments that deeply embed TPCs and perform nontrivial computing tasks in a user's environment.

BIBLIOGRAPHY

[1] S. A. Ahson and M. Ilyas, editors. *RFID Handbook: Applications, Technology, Security, and Privacy*. CRC Press, 2008.

[2] A. R. Alameldeen and D. A. Wood. Addressing workload variability in architectural simulations. *IEEE Micro*, 23(6):94–98, Nov. 2003.

[3] A. R. Alameldeen and D. A. Wood. Variability in architectural simulations of multi-threaded workloads. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, HPCA '03, pages 7–18, Washington, DC, USA, Feb. 2003. IEEE Computer Society.

[4] Alien Technology. Product Overview: Higgs-3 EPC Class 1 Gen 2 RFID Tag IC, July 2008.

[5] American National Standards Institute and Association for the Advancement of Medical Instrumentation. *EMC test protocols for implantable cardiac pacemakers and implantable cardioverter defibrillators—ANSI/AAMI PC69:2000*, May 2000.

[6] American National Standards Institute and Association for the Advancement of Medical Instrumentation. *EMC test protocols for implantable cardiac pacemakers and implantable cardioverter defibrillators—ANSI/AAMI PC69:2007*, Apr. 2007.

[7] Y. Ammar, A. Buhrig, M. Marzencki, B. Charlot, S. Basrour, K. Matou, and M. Renaudin. Wireless sensor network node with asynchronous architecture and vibration harvesting micro power generator. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-Aware Services: Usages and Technologies*, sOc-EUSAI '05, pages 287–292, New York, NY, USA, 2005. ACM.

[8] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, Feb. 2002.

[9] 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.

[10] H. Bassen. Low frequency magnetic emissions and resulting induced voltages in a pacemaker by iPod portable music players. *Biomedical engineering online*, 7(7), 2008.

[11] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vTPM: virtualizing the trusted platform module. In *Proceedings of the 15th USENIX Security Symposium*. USENIX Association, 2006.

[12] J. Bernstein, M. Gurfinkel, X. Li, J. Walters, Y. Shapira, and M. Talmor. Electronic circuit reliability modeling. *Microelectronics Reliability*, 46(12):1957–1979, Dec. 2006.

[13] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[14] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. In *CRYPTO*, pages 216–233. Springer-Verlag, 1999.

[15] M. Blaze. A cryptographic file system for UNIX. In *1st ACM Conference on Communications and Computing Security*, pages 9–16, Nov. 1993.

[16] Boston Scientific. PACEMAKER System Specification. `http://sqrl.mcmaster.ca/_SQRLDocuments/PACEMAKER.pdf`, Jan. 2007. Retrieved October 22, 2012.

[17] G. Brassard. On computationally secure authentication tags requiring short secret shared keys. In *CRYPTO*, pages 79–86, 1982.

[18] 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.

[19] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, Nov. 2000.

[20] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, May 2000.

[21] 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.

[22] M. Buettner. *Backscatter Protocols and Energy-Efficient Computing for RF-Powered Devices*. PhD thesis, University of Washington, 2012.

[23] 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.

[24] 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.

[25] 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.

[26] D. Burger, T. M. Austin, and S. Bennett. Evaluation future microprocessors: the SimpleScalar tool set. Technical Report 1308, University of Wisconsin–Madison, July 1996.

[27] W. P. Burleson, S. S. Clark, B. Ransford, and K. Fu. Design challenges for secure implantable medical devices. In *Proceedings of the 49th Design Automation Conference*, DAC '12, June 2012. Invited paper.

[28] L. Carter and M. Wegman. Universal hash functions. In *Journal of Computer and System Sciences*, pages 143–154. Elsevier, 1979.

[29] Academy of Achievement Interview: Johnny Cash. Audio interview, 1993. Online at `http://www.achievement.org/autodoc/printmember/cas0int-1`; retrieved January 8, 2013.

[30] H.-J. Chae, M. Salajegheh, D. J. Yeager, J. R. Smith, and K. Fu. *Wirelessly Powered Sensor Networks and Computational RFID*, chapter Maximalist Cryptography and Computation on the WISP UHF RFID Tag. Springer, 2012. To appear. An earlier version based on older WISP technology appeared in RFID-Sec07.

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

[32] R. Chandramouli, S. Bapatla, K. P. Subbalakshmi, and R. N. Uma. Battery power-aware encryption. *ACM Trans. Inf. Syst. Secur.*, 9(2):162–180, 2006.

[33] J. Chen, M. Dubois, and P. Stenström. Simwattch: Integrating complete-system and user-level performance and power simulators. *IEEE Micro*, 27(4):34–48, July 2007.

[34] R. Y. Chen, M. J. Irwin, and R. S. Bajwa. Architecture-level power estimation and design experiments. *ACM Trans. Des. Autom. Electron. Syst.*, 6(1):50–66, Jan. 2001.

[35] 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.

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

[37] G. D. Clifford. *Advanced Methods and Tools for ECG Data Analysis*, chapter ECG Statistics, Noise, Artifacts, and Missing Data. Artech House Publishers, 2006.

[38] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson. Nv-heaps: making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XVI)*, pages 105–118, Mar. 2011.

[39] D. Cooper, H. Dang, P. Lee, W. MacGregor, and K. Mehta. *Secure Biometric Match-on-Card Feasibility Report*, 2007.

[40] G. Crossley, A. Boyle, B. J. Berman, T. Alsheikh, B. C. Sodowick, S. A. John-son, and H. S. Vitense. Patient and caregiver burden of following implantable cardioverter defibrillators. *Heart Rhythm*, 5(5):S259, May 2008.

[41] M. de Kruijf, K. Sankaralingam, and S. Jha. Static analysis and compiler design for idempotent processing. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 475–486, June 2012.

[42] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy. Re-thinking data manage-ment for storage-centric sensor networks. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR)*, Jan. 2007.

[43] B. Dolan. Proteus receives patent for ingestible sen-sor. mobihealthnews, `http://mobihealthnews.com/11923/proteus-receives-patent-for-ingestible-sensor/`, Retrieved April 20, 2012.

[44] B. P. Douglass. *Doing Hard Time: Developing Real-Time Systems With UML, Objects, Frameworks, and Patterns*. Addison-Wesley Professional, 1999.

[45] H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler. TinyNode: a com-prehensive 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.

[46] 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.

[47] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. E. Culler. A building block approach to sensornet systems. In *Proc. 6th International Conference on Embedded Networked Sensor Systems*, SenSys '08, pages 267–280, Nov. 2008.

[48] EPCglobal. EPC Radio-Frequency Identity Protocols, Class-1 Generation-2 UHF RFID. http://www.epcglobalinc.org/standards/uhfc1g2/, 2008.

[49] 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.

[50] M. Etzel, S. Patel, and Z. Ramzan. Square hash: Fast message authentication via optimized universal hash functions. In *In Proc. CRYPTO 99, Lecture Notes in Computer Science*, pages 234–251. Springer-Verlag, 1999.

[51] FeS$_2$: A full-system execution-driven simulator for x86. http://fes2.cs.uiuc.edu/, Retrieved October 12, 2012.

[52] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *8th USENIX Symposium of Operating Systems Design and Implementation (OSDI'08)*, pages 323–328, 2008.

[53] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proc. 8th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, Dec. 2008.

[54] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. *ACM Transactions on Computer Systems*, 20(1):1–24, Feb. 2002.

[55] K. Fu, S. Kamara, and T. Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. In *Proceedings of the Symposium on Network and Distributed Systems Security*, Feb. 2006.

[56] The gem5 simulator system. http://gem5.org/, Retrieved October 12, 2012.

[57] 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.

[58] S. Gollakota, H. Hassanieh, B. Ransford, D. Katabi, and K. Fu. They can hear your heartbeats: Non-invasive security for implanted medical devices. In *Proceedings of ACM SIGCOMM*, Aug. 2011.

[59] M. Gorlatova, R. Margolies, J. Sarik, G. Stanje, J. Zhu, B. Vigraham, M. Szczodrak, L. Carloni, P. Kinget, I. Kymissis, and G. Zussman. Energy harvesting active networked tags (EnHANTs): Prototyping and experimentation. In *Proceedings of IEEE Infocom*, Apr. 2013.

[60] D. Graham-Rowe. New pacemaker needs no wires. `http://www.technologyreview.com/news/426164/new-pacemaker-needs-no-wires/`. Retrieved December 3, 2013.

[61] R. Gummadi, N. Kothari, T. Millstein, and R. Govindan. Declarative failure recovery for sensor networks. In *Aspect-Oriented Software Development*, 2007.

[62] J. Gummeson, S. S. Clark, K. Fu, and D. Ganesan. On the limits of effective micro-energy harvesting on mobile CRFID sensors. In *Proceedings of 8th Annual ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys 2010)*, pages 195–208, June 2010.

[63] J. Gummeson, P. Zhang, and D. Ganesan. Flit: A bulk transmission protocol for RFID-scale sensors. In *Proceedings of the 10th International Conference on Mobile Systems, Applications and Services (MobiSys 2012)*, June 2012.

[64] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. T. Kandemir, T. Li, and L. K. John. Using complete machine simulation for software power estimation: The softwatt approach. In *HPCA*, pages 141–150. IEEE Computer Society, Feb. 2002.

[65] Z. Gutterman, B. Pinkas, and T. Reinman. Analysis of the Linux random number generator. In *IEEE Symposium on Security and Privacy*, pages 371–385. IEEE Computer Society, 2006.

[66] S. Haddad, C. Chang, B. Swaminathan, and J. Lien. Degradations due to hole trapping in flash memory cells. In *Electron Device Letters*, pages 117–119. IEEE, Mar. 1989.

[67] N. M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed Systems Security*, Feb. 1994.

[68] D. Halperin, T. S. Heydt-Benjamin, K. Fu, T. Kohno, and W. H. Maisel. Security and privacy for implantable medical devices. *IEEE Pervasive Computing, Special Issue on Implantable Electronics*, 7(1):30–39, Jan. 2008.

[69] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the 29th Annual IEEE Symposium on Security and Privacy*, pages 129–142, May 2008.

[70] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the 29th IEEE Symposium on Security and Privacy*, May 2008.

[71] C. A. Hewitt, A. B. Kaiser, S. Roth, M. Craps, R. Czerw, and D. L. Carroll. Multilayered carbon nanotube/polymer composite based thermoelectric fabrics. *Nano Letters*, 12(3):1307–1310, 2012.

[72] D. E. Holcomb, W. P. Burleson, and K. Fu. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In *Proceedings of the Conference on RFID Security*, July 2007.

[73] Homer. *Odyssey*, volume XI. ca. 750 B.C.

[74] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, 1989.

[75] W. Huang, K. Sankaranarayanan, K. Skadron, R. J. Ribando, and M. R. Stan. Accurate, pre-RTL temperature-aware design using a parameterized, geometric thermal model. *IEEE Transactions on Computers*, 57(9):1277–1288, Sept. 2008.

[76] IMDb.com, Inc. Memento (2000). `http://www.imdb.com/title/tt0209144/`, Retrieved May 2, 2012.

[77] R. P. Jetley, P. L. Jones, and P. Anderson. Static analysis of medical device software using CodeSonar. In *Proceedings of the 2008 Workshop on Static Analysis*, SAW '08, pages 22–29. ACM, 2008.

[78] A. Juels. RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, Feb. 2006.

[79] W. Kainz, J. Casamento, P. Ruggera, D. Chan, and D. Witters. Implantable cardiac pacemaker electromagnetic compatibility testing in a novel security system simulator. *IEEE Transactions on Biomedical Engineering*, 52(3):520–530, 2005.

[80] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proc. USENIX Conference on File and Storage Technologies*, pages 29–42, San Francisco, CA, Dec. 2003.

[81] S. F. Kaplan. *Compressed Caching and Modern Virtual Memory Simulation*. PhD thesis, University of Texas at Austin, Dec. 1999.

[82] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[83] S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, and F.-X. Standaert. Towards green cryptography: A comparison of lightweight ciphers from the energy viewpoint. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 390–407. Springer Berlin / Heidelberg, 2012.

[84] H. Kim, K. Shin, and P. Pillai. MODELZ: Monitoring, detection, and analysis of energy-greedy anomalies in mobile handsets. *IEEE Transactions on Mobile Computing*, 10(7):968–981, July 2011.

[85] S. Kim, J. S. Ho, L. Y. Chen, and A. S. Y. Poon. Wireless power transfer to a cardiac implant. *Applied Physics Letters*, 101(7):073701, 2012.

[86] 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.

[87] M. Kuhn. *Compromising Emanations: Eavesdropping Risks of Computer Displays*. PhD thesis, Computer Laboratory, University of Cambridge, Wolfson College, 2003.

[88] 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.

[89] 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.

[90] 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.

[91] C. Li, A. Raghunathan, and N. K. Jha. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *Proceedings of the 13th IEEE International Conference on e-Health Networking, Applications, and Services*, Healthcom '11, June 2011.

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

[93] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, June 2003.

[94] 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.

[95] 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.

[96] Maine State Planning Office. Noise technical assistance bulletin #4. `http://www.maine.gov/spo/landuse/docs/NoiseTABulletin.pdf`, May 2000. Last viewed November 9, 2007.

[97] W. H. Maisel, M. Moynahan, B. D. Zuckerman, T. P. Gross, O. H. Tovar, D.-B. Tillman, and D. B. Schultz. Pacemaker and ICD generator malfunctions: Analysis of Food and Drug Administration annual reports. *Journal of the American Medical Association*, 295(16):1901–1906, Apr. 2006.

[98] V. S. Mallela, V. Ilankumaran, and N. S. Rao. Trends in cardiac pacemaker batteries. *Indian Pacing and Electrophysiology Journal*, 4(4):201–212, Oct. 2004.

[99] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 33(4):92–99, Nov. 2005.

[100] G. Massobrio and P. Antognetti. *Semiconductor device modeling with SPICE*. McGraw-Hill, 1993.

[101] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. CAPSULE: An energy-optimized object storage system for memory-constrained sensor devices. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2006.

[102] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: an execution infrastructure for TCB minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 315–328, New York, NY, USA, Apr. 2008. ACM.

[103] 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.

[104] 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.

[105] A. Molina-Markham, S. S. Clark, B. Ransford, and K. Fu. *Wirelessly Powered Sensor Networks and Computational RFID*, chapter BAT: Backscatter Anything-to-Tag Communication. Springer, 2012. To appear.

[106] D. Narayanan and O. Hodson. Whole-system persistence with non-volatile memories. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XVII)*, Mar. 2012.

148

[107] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden. Wishbone: Profile-based partitioning for sensornet applications. In J. Rexford and E. G. Sirer, editors, *NSDI*, pages 395–408. USENIX Association, 2009.

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

[109] 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.

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

[111] N. Paul, T. Kohno, and D. C. Klonoff. A review of the security of insulin pump infusion systems. *Journal of Diabetes Science and Technology*, 5(6):1557–1562, Nov. 2011.

[112] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept. 2002.

[113] D. Pivonka, A. Poon, and T. Meng. Locomotive micro-implant with active electromagnetic propulsion. In *Annual Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6404–6407, Sept. 2009.

[114] 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.

[115] 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.

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

[117] 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.

[118] K. Poulsen. Hackers assault epilepsy patients via computer. Wired.com, http://www.wired.com/politics/security/news/ 2008/03/epilepsy, Mar. 2008.

[119] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.

[120] J. Radcliffe. Hacking medical devices for fun and insulin: Breaking the human SCADA system. Black Hat Conference presentation slides, Aug. 2011.

[121] A. Rahmati, M. Salajegheh, D. Holcomb, J. Sorber, W. P. Burleson, and K. Fu. TARDIS: Time and remanence decay in SRAM to implement secure protocols on embedded devices without clocks. In *Proceedings of the 21st USENIX Security Symposium*, Security '12, Bellevue, WA, Aug. 2012.

[122] B. Ransford, S. 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)*, Dec. 2008.

[123] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on rfid-scale devices (technical report). Technical Report UM-CS-2010-060, Department of Computer Science, University of Massachusetts Amherst, Amherst, MA, Oct. 2010.

[124] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on RFID-scale devices. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '11, Newport Beach, CA, Mar. 2011.

[125] D. Raymond, R. Marchany, M. Brownfield, and S. Midkiff. Effects of denial-of-sleep attacks on wireless sensor network MAC protocols. *IEEE Transactions on Vehicular Technology*, 58(1):367–380, Jan. 2009.

[126] V. Reddi, M. Gupta, G. Holloway, M. Smith, G.-Y. Wei, and D. Brooks. Predicting voltage droops using recurring program and microarchitectural event activity. *IEEE Micro*, 30(1):101–109, Jan. 2010.

[127] V. Reddi, M. Gupta, M. Smith, G. yeon Wei, D. Brooks, and S. Campanoni. Software-assisted hardware reliability: Abstracting circuit-level challenges to the software stack. In *Proceedings of the 46th Design Automation Conference*, DAC '09, pages 788–793, July 2009.

[128] R. L. Rivest. The RC5 encryption algorithm. *Dr Dobb's Journal—Software Tools for the Professional Programmer*, 20(1):146–149, 1995.

[129] R. L. Rivest. The RC5 encryption algorithm. In B. Preneel, editor, *Fast Software Encryption*, pages 86–96. Springer, 1995. (Proceedings Second Int'l Workshop, Dec. 1994, Leuven, Belgium).

[130] P. Roberts. Blind attack on wireless insulin pumps could deliver lethal dose. Threatpost (blog post), http://threatpost.com/en_us/blogs/blind-attack-wireless- insulin-pumps-could-deliver-lethal-dose-102711, Oct. 2011.

[131] 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.

[132] RSA Laboratories. How fast is the RSA algorithm? `http://www.rsa.com/rsalabs/node.asp?id=2215`, Retrieved January 2, 2013.

[133] P. S. Ruggera, D. M. Witters, and H. I. Bassen. In vitro testing of pacemakers for digital cellular phone electromagnetic interference. *Biomedical Instrumentation & Technology*, 31(4):358–371, 1997.

[134] D. Sagan. Meeting the challenge of ultra low power communication. Presentation slides, online at `http://stf.ucsd.edu/presentations/2007/2007-08%20STF%20-%20Zarlink%20ULP%20transceivers.pdf`, Aug. 2007.

[135] M. Salajegheh, S. S. Clark, B. Ransford, K. Fu, and A. Juels. CCCP: secure remote storage for computational RFIDs. In *Proceedings of the 18th USENIX Security Symposium*, pages 215–230, Montreal, Canada, Aug. 2009.

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

[137] M. Salajegheh, Y. Wang, A. A. Jiang, E. Learned-Miller, and K. Fu. Half-wits: Software techniques for low-voltage probabilistic storage on microcontrollers with NOR flash memory. *ACM Transactions on Embedded Computing Systems, Special Issue on Probabilistic Embedded Computing*, 2012. Preprint available upon request.

[138] 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.

[139] 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, Nov. 2008.

[140] B. Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[141] H. P. Schwan and C. F. Kay. Specific resistance of body tissues. *Circulation Research*, 1956(4):664–670, 1956.

[142] S. J. Seidman, P. S. Ruggera, R. G. Brockman, B. Lewis, and M. J. Shein. Electromagnetic compatibility of pacemakers and implantable cardiac defibrillators exposed to RFID readers. In *International Journal on Radio Frequency Identification Technology and Applications*, 2007.

[143] S. A. Seshia and A. Rakhlin. Quantitative analysis of systems using game-theoretic learning. *ACM Transactions on Embedded Computing Systems*, 11(S2):55, Aug. 2012.

[144] A. Shamir. SQUASH—a new MAC with provable security properties for highly constrained devices such as RFID tags. In *Proceedings of the 15th International Workshop on Fast Software Encryption (FSE)*, pages 144–157. Springer-Verlag, 2008.

[145] Wind River Simics. `http://www.windriver.com/products/simics/`, Retrieved October 12, 2012.

[146] E. Singer. A pacemaker the size of a Tic Tac. `http://www.technologyreview.com/news/423134/a-pacemaker-the-size-of-a-tic-tac/`. Retrieved December 3, 2012.

[147] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture*, ISCA '03, pages 2–13. IEEE Computer Society, June 2003.

[148] Software Quality Research Lab. Pacemaker formal methods challenge. `http://sqrl.mcmaster.ca/pacemaker.htm`, Apr. 2007. Retrieved October 22, 2012.

[149] 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.

[150] St. Jude Medical. The electrical management of cardiac rhythm disorders—Bradycardia—Electrical concepts. `http://www.sjmprofessional.com/Media/DownloadResource.aspx?id=6FF945AA-56B5-49DE-8787-A2E606D134E6`, Retrieved May 16, 2012. Presentation slides.

[151] F. Stajano and R. J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of Security Protocols, 7th International Workshop, LNCS 1796*, pages 172–194, 1999.

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

[153] 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.

[154] Y. Tanabe, H. Wong, S. Kim, J. S. Ho, and A. S. Y. Poon. Beam focused slot antenna for microchip implants. In *International Symposium on Antennas and Propagation (ISAP)*, pages 467–470, Nov. 2012.

[155] Texas Instruments, Inc. MSP430 Ultra-Low Power Microcontrollers. http://www.ti.com/msp430.

[156] Texas Instruments, Inc. MSP430F21x2 Mixed Signal Microcontroller (datasheet), Nov. 2007.

[157] Texas Instruments, Inc. CC2500 Low-Cost Low-Power 2.4 GHz RF Transceiver (datasheet), 2011.

[158] Texas Instruments, Inc. SimpliciTI compliant protocol stack. `http://www.ti.com/simpliciti`, Retrieved December 3, 2012.

[159] Texas Instruments Incorporated. http://www.ti.com/rfid/shtml/news-releases-11-12-07.shtml.

[160] 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.

[161] Trusted computing group. http://www.trustedcomputinggroup.org.

[162] H.-W. Tseng, L. M. Grupp, and S. Swanson. Understanding the impact of power loss on flash memory. In *Proc. 48th Design Automation Conference*, DAC '11, pages 35–40, June 2011.

[163] E. Y. Vasserman and N. Hopper. Vampire attacks: Draining life from wireless ad-hoc sensor networks. *IEEE Transactions on Mobile Computing*, 99, 2011.

[164] N. Villar and S. Hodges. The peppermill: a human-powered user interface device. In *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction*, TEI '10, pages 29–32, New York, NY, USA, 2010. ACM.

[165] H. Volos, A. J. Tack, and M. M. Swift. Mnemosyne: lightweight persistent memory. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XVI)*, pages 91–104, Mar. 2011.

[166] A. Wallace. Pacemakers for anesthesiologists made incredibly simple. `http://www.cardiacengineering.com/pacemakers-wallace.pdf`, Apr. 2008. Retrieved September 2, 2012.

[167] J. G. Webster, editor. *Design of Cardiac Pacemakers*. IEEE Press, 1995.

[168] L. Wong, S. Hossain, A. Ta, J. Edvinsson, D. Rivas, and H. Naas. A very low-power CMOS mixed-signal IC for implantable pacemaker applications. *IEEE Journal of Solid-State Circuits*, 39(12):2446–2456, Dec. 2004.

[169] 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.

[170] D. Yeager, P. Powledge, R. Prasad, D. Wetherall, and J. Smith. Wirelessly-Charged UHF Tags for Sensor Data Collection. In *IEEE Int'l Conference on RFID*, pages 320–327, 2008.

[171] 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.

[172] 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.

[173] L. Yerva, B. Campbell, A. Bansal, T. Schmid, and P. Dutta. Grafting energy-harvesting leaves onto the sensornet tree. In F. Zhao, A. Terzis, and K. White-house, editors, *IPSN*, pages 197–208. ACM, 2012.

[174] S. Yi, J. Heo, Y. Cho, and J. Hong. Adaptive mobile checkpointing facility for wireless sensor networks. In M. L. Gavrilova, O. Gervasi, V. Kumar, C. J. K. Tan, D. Taniar, A. Laganà, Y. Mun, and H. Choo, editors, *ICCSA (2)*, volume 3981 of *Lecture Notes in Computer Science*, pages 701–709. Springer, 2006.

[175] H. Zhang, J. Gummeson, B. Ransford, and K. Fu. Moo: A batteryless computational RFID and sensing platform. Technical Report UM-CS-2011-020, Department of Computer Science, University of Massachusetts Amherst, Amherst, MA, June 2011.

[176] H. Zhang, M. Salajegheh, K. Fu, and J. Sorber. Ekho: Bridging the gap between simulation and reality in tiny energy-harvesting sensors. In *Workshop on Power Aware Computing and Systems (HotPower 2011)*, Oct. 2011.