Dewdrop: An Energy-Aware Runtime for Computational RFID

Michael Buettner*, Ben Greenstein† and David Wetherall*†

University of Washington and Intel Labs Seattle†*

Abstract

Computational RFID (CRFID) tags embed sensing and computation into the physical world. The operation of the tags is limited by the RF energy that can be harvested from a nearby power source. We present a CRFID runtime, *Dewdrop*, that makes effective use of the harvested energy. *Dewdrop* treats iterative tasks as a scheduling problem to balance task demands with available energy, both of which vary over time. It adapts the start time of the next task iteration to consistently run well over a range of distances between tags and a power source, for different numbers of tags in the vicinity, and for light and heavy tasks. We have implemented *Dewdrop* on top of the WISP CRFID tag. Our experiments show that, compared to normal WISP operation, *Dewdrop* doubles the operating range for heavy tasks and significantly increases the task rate for tags receiving the least energy, all without decreasing the rate in other situations. Using offline testing, we find that *Dewdrop* runs tasks at better than 90% of the best rate possible.

1 Introduction

Computational RFID (CRFID) tags are an emerging technology in which sensing and computational abilities are added to traditional RFID tags. Passive UHF RFID tags run and transmit an identifier using energy gathered from the transmissions of nearby RFID readers; they are very small and have no battery or long-term energy store. This ability makes them widely useful in commercial settings to, for example, automate interactions with passports and drivers licenses, identify animals, and track retail goods in manufacturing and supply chains. The addition of sensing and computation with CRFIDs enables a broader range of sensing applications, including cold-chain monitoring, access control, embedded monitoring of bridges and planes, gestural interfaces, activity recognition, and non-intrusive physiological monitoring [2]. These and other applications depend on very small, long-lived nodes that can be deeply embedded into the physical environment in ways that go beyond sensor nodes and approach the original vision of “smart dust” [28].

The research agenda associated with CRFIDs is now becoming defined as the community uses prototype tags to experiment with applications [3, 6, 9]. A fundamental problem for these devices is the efficient use of energy. Energy is the scarce resource that limits the amount of computation that can be performed because it must be harvested at low rates from signals transmitted by readers meters away. Further, to remain physically small and to power-up quickly, CRFIDs have miniscule energy stores compared to sensor network nodes. For example, the energy store of the WISP [24] prototype tag is *eight* orders of magnitude smaller than the battery of the popular Telos sensor mote[18]. This means that CRFIDs will typically exhaust and recharge their energy stores many times a second. In turn, it means that runtimes for sensor networks are of little use for CRFIDs. Sensor node runtimes seek to keep long-term expenditures below long-term harvesting or to maximize node lifetimes measured in days [14]. In contrast, CRFID runtimes must take a short-term view to match lifetimes measured in milliseconds.

The problem we tackle in this paper is how CRFID tags can make efficient use of the available energy. The naive RFID power model on which CRFIDs are based is for the tag to turn on and run whenever it is powered by the reader. This approach works for traditional RFID tags because tag functionality is very simple (a state machine with memory) and can be run in the worst case at the limit of the energy harvesting range. However, CRFID tasks consume greater energy with more complicated tasks that use sensors and computation. By adopting the model of running whenever there is power, current CRFID designs reduce the range at which a CRFID tag functions and limit the kinds of tasks that can be run. Prior work has looked at tuning the CRFID hardware constants (e.g., capacitor sizes) to better match available energy to a specific task [8]. Instead, our approach is to view the need to match harvested energy to task consumption as a scheduling problem. We wake the tag out of deep sleep only when it is likely to execute a task efficiently. This enables devices to run a range of tasks efficiently without requiring hardware modications.

We present the design and evaluation of *Dewdrop*, an energy-aware runtime for CRFID tags. We have implemented *Dewdrop* on the Intel WISP tag, and have experimented by powering the tags using a commodity Impinj UHF RFID reader for a range of distances, number of competing tags, and light and heavy CRFID tasks. By waking tags at the right times, we find that we can run tasks where they previously could not run, and about as often as possible given the energy that the RF environment provides. Prior to our work, the WISP had an operating range sufficient for point demonstrations. With our runtime, it is possible to use a single RFID reader to track CRFID tags on everyday objects in a room with enough responsiveness for activity inference.

While *Dewdrop* is conceptually simple, we found a practical design difficult to achieve for several reasons. First, the energy needed to run a task and the input RF power both vary greatly over time due to factors such as non-deterministic protocols and reader frequency hopping. This hampers predictions of when to start the next task execution. Second, our intuition about energy storage as a simple reservoir proved wrong because a fixed amount of energy is more or less expensive to store depending on when it is gathered, and the rate at which it is consumed depends on when it is spent. This leads us to track other forms of waste. Finally, it is costly to gather the basic information needed to make scheduling decisions because CRFIDs are so energy impoverished. This required opportunistic measurement strategies and careful implementation.

We make three contributions. First, we formulate the task scheduling problem for CRFID tags with limited energy storage. Second, we present the design of a runtime that enables CRFID tags to adapt their behavior to best match task energy requirements to available energy over the factors that most affect efficiency. Third, we show by experimentation with the WISP tag and an Impinj RFID reader that our design is much more effective than prior techniques for real energy costs and RF conditions. *Dewdrop* doubles the operating range for heavyweight tasks as compared to the WISP hardware that runs tasks whenever there is power, and keeps overhead low to match the performance for lightweight tasks to which the WISP hardware is well suited.

The rest of this paper is organized as follows. We start with background in Section 2 and then define the task scheduling problem for CRFIDs in Section 3. We present the design of *Dewdrop* and its implementation in Sections 4 and 5. Our experimental evaluation is in Section 6. We follow with related work in Section 7 and conclude in Section 8.

2 Background

We begin with relevant background on computational RFID because it is an emerging research area.

CRFID tags and the WISP. CRFID tags combine RFID technology for energy harvesting and backscatter communication with computation and sensing. The prototype CRFID tag that we use is the Intel Wireless Identification and Sensing Platform (WISP) [24]. Other prototype CRFID tags exist [21, 30], but the WISP is the most widely used because it is available to the academic community.[1]

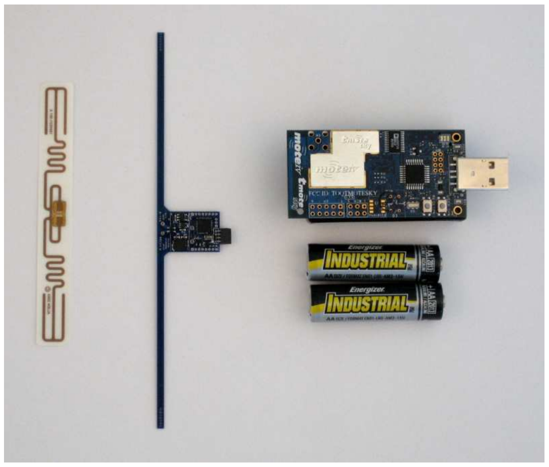
Figure 1: Gen 2 tag, Intel WISP, Telos mote.

Figure 1 shows the WISP in comparison to a Gen 2 UHF RFID tag and a Telos mote. Like an RFID tag, it is small, thin, and battery-free. It runs only when powered by energy harvested from an EPC Gen 2 RFID reader and communicates with the reader using a low-energy form of signaling called backscatter. The current WISP can harvest sufficient power to operate at up to 4m. As advances in processor and sensor technology continue to reduce power consumption, the range of WISP tags will increase accordingly.

Like a very low-end mote, the WISP is fully programmable, capable of running small programs, and equipped with sensors. The WISP runs programs written in C on an ultra-low power 16-bit MSP430 microcontroller and has 8K of flash memory, a 3D accelerometer, and temperature and light sensors.

However, unlike an RFID tag, the WISP consumes considerably more power when computing, communicating and sensing than can normally be harvested from the reader signal. Consequently, the WISP must duty cycle between a low-power sleep mode, in which the energy needed to run is gathered into a short-term energy buffer, and an active mode in which stored energy is consumed.

We expect future CRFID tags to be more capable than the WISP, but to remain very-low end devices, even compared to sensor nodes. As the power efficiency of the devices improves slowly over time, so too will the sensing and processing demands that are placed on them; thus, the disparity between harvestable power and operating power will remain.

CRFID Applications. CRFID tags and readers are enablers for ubiquitous computing applications that benefit

[1]See `wisp.wikispaces.com` for open-source WISP software and hardware designs. WISPs are in use at more than 30 universities.

from instrumentation on or as part of objects in the physical world. For example, the WISP has been used to prototype applications for gesture-based access control [6], cold chain monitoring [29], and activity recognition for eldercare [3].

We delve into the last scenario to give one example of a workload that *Dewdrop* is intended to support. The automatic recognition of the activities of elderly people can improve quality of life by helping elders remain in their own homes for longer with inexpensive care. It does this by tracking key indicators of well-being such as medication adherence, mobility and exercise, food and water intake, changes in routine, and safety [17]. The use of CRFIDs for activity recognition can deliver a solution that is inexpensive and non-intrusive. CRFIDs with accelerometers can be affixed to objects in an elder's home, and data gathered from the tags can be used to determine activity. This has advantages over existing solutions as it requires neither monitoring by cameras, which can invade privacy, nor on-body sensors, which can be inconvenient for elders. Additionally, this type of deployment would be difficult using motes because of their size and cost.

In earlier work, we prototyped such a system by tagging objects an elderly person normally interacts with—her medicine cabinet, tea kettle, teacup, toothbrush, etc.—with CRFIDs with onboard accelerometers [3]. RFID readers were placed out of sight in the ceiling. Each CRFID repeatedly sampled its accelerometer and transmitted its value to the readers. The readers detected tags that moved by looking for changes in those values, for instance, when a CRFID-tagged medicine bottle is picked up. Activities such as *preparing a meal* and *taking medicine* were then inferred from sequences of object use.

We built our earlier system using WISPs and found that the system worked, albeit with a smaller coverage region and lower response rates than we expected. This meant that we needed to deploy multiple readers per room, and even then some tags responded infrequently, which degraded activity inference. After some investigation, we determined that the WISPs were wasting much of the available energy. That discovery led to our work on *Dewdrop*.

3 Problem

Our goal is to run programs on CRFID tags in a way that makes the best use of the available energy, which in turn extends operational range and increases responsiveness. In this section, we formulate this goal as a scheduling problem and describe the key challenges.

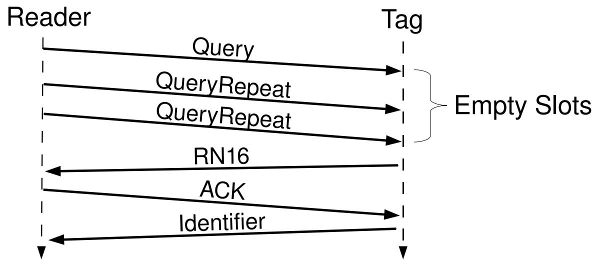

Figure 2: Example message exchange of a reader identifying a tag.

3.1 Task Model

In our setting, a reader powers one or more nearby tags and requests that they perform tasks. Tags may come and go from the range of a given reader as the RF environment changes or the tag or reader moves. In keeping with other CRFID and RFID applications, we assume that each CRFID tag repeatedly executes a single fixed operation as often as possible (e.g., reporting a sample), but from time to time may be retasked to perform a different operation (e.g., switch from sampling the accelerometer to measuring the light level). Additionally, tags in the deployment may be executing different tasks. As a tag considers only one type of task at a time, scheduling the order and execution of multiple tasks on a single tag is both unnecessary and out of scope.

We define a *task* to mean a short program that is run to completion without pause. While it may be possible to break some tasks into phases, the timing requirements of the tag hardware, the RFID protocol, and application requirements make it impractical to interrupt many tasks once they start. Due to the operating constraints of a tag, tasks are fairly inflexible and have limited functionality. They can support modest processing, e.g., for lightweight encryption, but generally consist of sensing and reporting operations. Even with this limited task diversity, tasks have very different power requirements. For example, measuring the light level consumes much less power than activating and sampling the accelerometer. We experiment with examples at the lower and higher ends of this spectrum later in the paper.

We assume that CRFID tags will be powered by a standard Gen 2 RFID reader, at least in the near future. This is likely, as it allows CRFID tags to take advantage of deployed and commodity infrastructure. Tasks often return a result to the reader. Contention between the transmissions of multiple tags is managed by the EPC Gen 2 MAC protocol [7] that is based on Framed Slotted Aloha [25]. To gather tag IDs, the reader transmits a *Query* command that indicates the number of slots in the frame. Tags then randomly choose a slot in which to reply, and transmit a 16-bit random number in their slot.

The reader *ACKs* this random number and the tag replies with a 96-bit identifier. An example of this exchange is shown in Figure 2, where no tag chooses the first two slots, and one tag responds in the third slot. Tags that collide in a slot are not *ACK*ed and respond again after the next *Query*. The reader iteratively modifies the frame size to best match the number of tags that are present. Sensor and other data is transferred on top of this protocol, either by overloading the identifier bits or using further commands that read and write tag memory. New MAC protocols specially designed for CRFIDs are also of interest, but we leave them to future work.

3.2 Task Scheduling Goal

Given that tags repetitively execute a task whenever possible and the reader power is not controlled by the tags, maximizing energy efficiency is equivalent to maximizing the rate at which tasks successfully complete. We use task completion rate, in terms of how many task iterations succeed over a given time period, as a metric to evaluate the performance of *Dewdrop* in the steady state. Since energy falls off with distance (at least as quickly as distance squared), we expect the completion rate to fall with distance. But, it should not fall more quickly than the available energy.

CRFID tags like the WISP collect the energy harvested from RF signals into a capacitor that matches the fluctuating input power to the steady output power needed to run the tag. Energy is harvested whenever a nearby reader is transmitting an RF signal. Like an RFID tag, the WISP hardware begins task execution whenever a fixed, hardware-defined power level that is sufficient to activate the tag is reached. Once a task iteration has started, it may either run to completion or fail if the CRFID tag runs out of energy first. We use this fixed, hardware approach as a baseline for comparison in our evaluation.

Dewdrop replaces the fixed, hardware approach with an adaptive software strategy. There is only one decision that a tag can make to improve energy efficiency: to defer the start of a task it could otherwise begin, sleeping until the energy store becomes more full. This is useful because the larger store of energy increases the chance that the task will run to completion. However, it is wasteful in terms of time and energy if the task would have succeeded anyway. The runtime's job is to decide when to run and when to sleep depending on the task and RF environment.

3.3 Challenge: Varying Task Needs

A good runtime will not start a task unless there is (likely) sufficient energy to complete it, as failing a task consumes energy without doing useful work. Yet whether a task will succeed is difficult to predict because task energy requirements vary greatly due to two main factors.

Different size tasks. The energy consumption of different tasks can vary widely depending on the sensors they use, the computation they perform, and their communication patterns. In our experiments, we consider a light task that simply takes an accelerometer reading, and a much heavier task that additionally uses the RFID communication protocol to send the accelerometer data to the reader by embedding it in the tag identifier. We refer to these as the SENSE and SENSETX tasks, respectively.

Non-deterministic tasks. Tasks may be non-deterministic, which causes their energy requirements to vary from execution to execution. An important source of non-determinism is the RFID MAC protocol. The number of messages that a tag must process to communicate with the reader depends on both the number of other tags present and the collisions that happen to occur. As a consequence of the way the protocol works, a tag that chooses to take part in a communication round must complete the transaction; it cannot sleep or it will lose synchronization with the reader. Other sources of non-determinism may come from sensor data itself, the timing of reader queries (which a tag cannot control or predict) or random numbers used in security protocols.

3.4 Challenge: Platform Inefficiencies

The variation in task energy requirements suggest that a better strategy might be to overestimate the task needs. For example, a tag could harvest energy until its buffer is completely full before executing a task. In this way, it would run with “a full tank” to avoid preventable failures and top off between tasks. Unfortunately, storing excess energy is wasteful due to platform characteristics.

Sublinear charging. CRFIDs use capacitors for energy storage as they are well suited to energy harvesting devices [12]. They charge quickly, recharge indefinitely, are small and inexpensive, and are non-toxic. However, capacitors store energy faster when they are close to empty than when nearly fully charged. This nonlinearity is fundamental to the way capacitors work. As the capacitor voltage, which increases with increasing charge, approaches the voltage supplied by the energy harvesting circuitry, the charging current decreases to zero. Thus, to increase the task rate, it makes sense to operate with a lightly charged capacitor.

Superlinear discharge. Regulating circuitry must adjust the supplied (input or stored) voltage to the operating voltage. Differences in voltage levels inevitably lead to some voltage-dependent conversion losses. For example, the WISP uses a linear regulator that sheds the voltage difference by dissipating heat, which wastes energy. Other techniques are possible but come with their own tradeoffs (e.g., switching regulators are more efficient but have greater leakage, don't work when the input voltage is near the target voltage, and are inefficient when they start up[2]). To minimize energy wasted while discharging, the tag again should operate with its capacitor at a minimal charge.

The exact inefficiencies will vary with the CRFID, but we believe that all real platforms will have these kinds of nonlinearities. The implication is that a quantum of energy may cost (or be worth) a different amount depending on when it is gathered (or spent), with excess energy being more wasteful.

3.5 Challenge: Varying Input Power

Even assuming that the tag runtime could accurately estimate tasks costs, it is difficult to know how long to sleep to store sufficient energy because the rate at which a tag harvests energy changes over time.

Widely varying input powers. RF power received at a tag decreases at least as fast as the square of its distance from the reader. In practice, this means that the available energy varies by more than an order of magnitude over useful ranges. Hardwiring tags to operate at the low end of the power scale wastes a significant opportunity at the high end of the scale, and restricting tags to operate at the high end of the scale limits operational range. Additionally, CRFIDs harvest energy even when the task is being executed. When the tag is close to a reader less energy will be drained from the energy store than when further from the reader. Consequently, when close to the reader, less energy needs to be stored before execution can begin.

Frequency selective fading. RFID systems operate in the 900MHz ISM band, so the reader must frequency hop every 400ms to obey FCC regulations. Multipath effects result in different frequencies being attenuated differently. This means that the received power at tags can vary widely over short time scales.

4 Design

We now develop the design of our energy-aware runtime, *Dewdrop*. The main scheduling decision is when to start the next task iteration. Starting too soon wastes energy when the tag runs out of power and the task fails. Starting too late collects excess energy, which is inefficient to both store and use. Our approach is to minimize both forms of waste. As we develop our design, we present microbenchmarks using the WISP to show the importance of the different factors we identified as challenges.

[2]This and other parts and design tradeoffs make the linear regulator the best choice for the WISP.

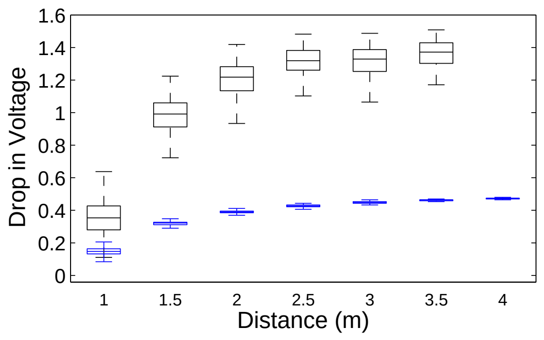

Figure 3: Voltage drop forSENSETX (upper black items) and SENSE (lower blue items).

4.1 Design Goals

From our problem formulation, the overarching goal of *Dewdrop* is to convert all available energy into completed task iterations. This goal is equivalent to two sub-goals that help to enable new applications:

Increased range. We want our runtime to execute a task at greater distances from the reader than the baseline WISP hardware. Each task should work from next to the reader out to the distance at which the tag can no longer harvest enough energy for the task.

Improved responsiveness. At all distances, we want to increase responsiveness compared to the baseline WISP hardware. We never want to noticeably decrease responsiveness.

Both goals are met by maximizing the task completion rate for a given task and distance from the reader. In practice, achieving them implies that we must meet two other goals:

Low overhead. The implementation of *Dewdrop* must be extremely lightweight. Operations such as checking the level of the energy store or calculating sleep periods consume scarce energy. Even a modest amount of overhead can easily negate the benefits of scheduling tasks.

Adaptation. Tags must operate well across a range of deployment scenarios. For example, they may be configured to run either heavy or lightweight tasks, and they must run their task efficiently both when near and far from a reader. Our performance sub-goals are stated across these factors, so *Dewdrop* must adapt to the environment at runtime.

4.2 Variation in Task Costs

To predict when to start a task, *Dewdrop* must estimate how much energy the task will need over and above the energy that will be harvested by the tag while it runs the task. This depends on the factors we previously identified: the task itself, other tags competing for the medium, the distance from the reader and the frequency on which the reader is transmitting, and the amount of energy already in the capacitor. All of these factors are fundamental. However, they may differ in magnitude with implications for system design. For example, if the energy needs depend mostly on the type of task, then each task could be profiled offline to characterize its fixed energy need.

To understand how much these factors matter in practice, we ran an experiment with the SENSE and SENSETX tasks running on a WISP. For the WISP, the energy consumption of a task can be measured by the drop in the voltage of the capacitor that acts as a short-term energy buffer[3]. Figure 3 shows this voltage drop as a function of distance for the two tasks. Box plots show the distributions over at least 300 task executions at each distance.

The SENSE task is deterministic. However, we see that the voltage drop is significantly larger when the tag is far from the reader than when it is close to the reader; it more than triples. This is because the input power from the reader varies by more than an order of magnitude. A second effect is that the variance is larger when the task is run close to the reader because the input power supplements stored energy and varies with the reader transmit frequency. At 1m this variance is approximately 0.3V compared to 0.1V at 4m.

Looking at the SENSETX task, the drop in voltage is almost three times larger than for SENSE. At 4m, the WISP cannot store sufficient energy to execute the task[4]. The variation is also higher at all distances because this task is non-deterministic. Its energy consumption depends on randomization in the Gen 2 MAC protocol, and the variation would be even greater if there were multiple WISPs (which we study as part of our evaluation).

These results imply that *Dewdrop* should adapt to both the task and the environment in which the tag is operating. Any fixed energy target at which to start a task will be either too low, causing the tag to fail at a distance when it could still run, or too high, causing the tag to run tasks more infrequently than it is capable of sustaining. A second implication is that it is likely not feasible to accurately estimate the energy needs of a particular task execution due to inherent variation. Instead, *Dewdrop* must adapt an estimate of energy needs that captures the effects of the distribution.

[3]The energy stored in a capacitor is calculated as $\frac{1}{2}CV^2$, where C is the capacitance and V is the measured voltage.

[4]To even run the task over a range of distances we needed to modify the baseline WISP behavior.

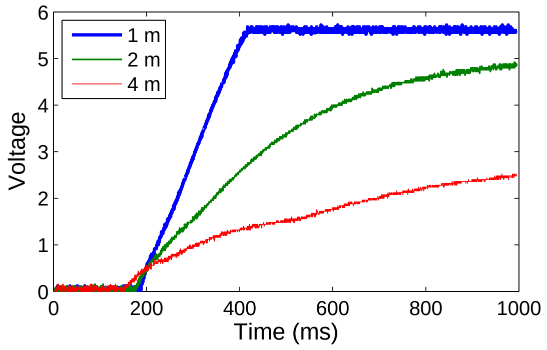

Figure 4: WISP capacitor voltage over time

4.3 Minimizing Wasted Energy

Sources of waste. Energy is wasted when the CRFID tag starts too early and fails to complete the task, or waits too long and inefficiently collects excess energy. How much energy is wasted in these cases depends on how CRFID tags convert reader energy into harvested energy and consume this energy.

To gain some insight, we performed a simple experiment by charging a WISP without running any task. Figure 4 shows the voltage of the WISP capacitor as it charges at different distances. (The RF source powers on at approximately 200 ms.) This is the expected behavior. A capacitor's charging rate decreases by a factor of e every RC seconds, where R and C are the resistance and capacitance of the RC circuit and e is the base of natural logarithms, and asymptotically approaches zero as the capacitor charges to the voltage of the power source.

This charging behavior has two implications. First, it shows the effects of distance. Far from the reader, the low received power limits the maximum energy that can be stored. At 4m the capacitor approaches only 2.75V, while at 1m it rises quickly to 5.8V (at which point an over-voltage protection circuit kicks in). This means that heavy tasks will not run as far from the reader as lightweight tasks no matter how long the tag sleeps.

The second implication is that, even for a fixed input power, it is inefficient to charge to a higher voltage than necessary. Because the rate at which energy accumulates in a capacitor decreases exponentially as it charges, storing excess energy wastes *time*. There is a penalty for charging too high and leaving spare energy in the capacitor. In a sense, that leftover energy was “cheaper” to store. This effect is magnified by the linear regulator of the WISP, which consumes more power when there is a higher charge on the capacitor.

To capture these factors, *Dewdrop* estimates waste in terms of time. This directly accounts for the energy consumed by a task, even if it fails, and also for how long it took to store that energy. While the details will differ, all platforms are likely to have nonlinearities with respect to storing and consuming energy that make it useful to measure waste in terms of time. For instance, capacitors are the natural choice for short-term energy storage, and all CRFIDs that use capacitors will have this kind of inefficiency.

Balancing sources of waste. Intuitively, starting tasks later, at a higher energy level, will decrease the time wasted due to tasks failing but increase the time wasted due to excess charging. Our goal is to minimize the total wasted time due to both causes. Since the energy cost of executing a task cannot be estimated precisely, *Dewdrop* aims to reduce the expected wasted time in the following manner. Let $P(fail|V_s)$ be the probability that the task will fail given a starting voltage level V_s. The runtime's job is to choose a V_s in the range $[V_0, V_{max}]$ that minimizes the wasted time:

$$\begin{aligned} t_{wasted}(V_s) &= P(fail|V_s)t_{under} \\ &+ (1 - P(fail|V_s))t_{over} \end{aligned}$$

where t_{under} is the time to charge back to V_s after a failure and t_{over} is the time spent overcharging, i.e., the time spent charging beyond the energy level that would have been sufficient. Note that this implies that some rate of failures may be desirable as charging high enough to assure success incurs a penalty that accumulates on every execution.

A naive approach to finding the V_s that minimizes wasted time would be to try every value of V_s. This is impractical, as the tag would need to examine a sufficiently long series of task execution attempts at each V_s to determine which had the best performance. Furthermore, this search would need to be repeated periodically as the RF environment and other factors change.

To avoid this search, we use our intuition that the two kinds of wasted time tradeoff against each other to find an approximate solution. Let P_f be the current task failure rate at a fixed starting voltage V_s and $T_{under} = P_f * t_{under}$ and $T_{over} = (1-P_f)*t_{over}$. If $T_{over} >> T_{under}$, then the runtime is too conservative; it could have chosen a lower V_s. If $T_{under} >> T_{over}$ then it is being too aggressive; V_s is too low and tasks are failing too often.

Dewdrop uses the heuristic that balancing the two sources of waste tends to minimize overall wasted time; this at least finds a reasonable operating point by ensuring that neither factor is a major source of inefficiency. Additionally, tracking and comparing the two sources of wasted time requires minimal computation which is key for any viable solution. The balance point can be found by slowly updating V_s to trade T_{under} against T_{over}. To do this, *Dewdrop* maintains separate estimates of T_{under} and T_{over} that are updated with an exponentially weighted moving average (with parameter α) each time a task executes depending on its success or failure. The two estimates are then compared, and the energy level V_s is adjusted by β in the direction that will balance the averages. That is, it is increased if more time is being wasted on failures than on charging too high.

More precisely, let V_e be the voltage at the end of running a task, and V_0 be the voltage at which the tag ceases to operate, and ϵ be a small voltage. A task succeeds if and only if $V_e \geq V_0 + \epsilon$. *Dewdrop* computes estimates and uses them to adjust the target energy level, V_s as follows:

$$T_{over} = \begin{cases} (1-\alpha)T_{over} + \alpha t_{over}, & \text{if } V_e \geq V_0 + \epsilon \\ (1-\alpha)T_{over}, & \text{if } V_e < V_0 + \epsilon \end{cases}$$

$$T_{under} = \begin{cases} (1-\alpha)T_{under}, & \text{if } V_e \geq V_0 + \epsilon \\ (1-\alpha)T_{under} + \alpha t_{under}, & \text{if } V_e < V_0 + \epsilon \end{cases}$$

$$V_s = \begin{cases} V_s - \beta, & \text{if } T_{over} > T_{under} \\ V_s + \beta, & \text{if } T_{under} > T_{over} \end{cases}$$

Of course, there are degenerate cases where this heuristic will fail, e.g., tasks that exhibit bimodal energy consumption where some executions consume a lot of energy and some executions consume very little. But, based on applications we have seen in the literature, our approach is a good fit and has the benefit of being both simple and efficient.

4.4 Charging to a Target Energy Level

Given a target energy level, the CRFID runtime must arrange for the task to begin execution when stored energy reaches that target. The baseline WISP uses hardware support in the form of a voltage supervisor to start execution when the capacitor voltage reaches a fixed level of 2V. Unfortunately, there are no designs for variable voltage supervisors that can be used in CRFIDs to the best of our knowledge.

Instead, *Dewdrop* uses a software polling approach to determine when the target energy level has been reached and execution should begin. It sleeps while energy is being harvested, and occasionally wakes up to sample the capacitor voltage using an analog to digital converter (ADC). This is a general strategy that can be used on most platforms regardless of how the target energy level is determined.

However, polling is difficult to achieve at low cost because charge times can vary over orders of magnitude and waking up and sampling the capacitor consumes precious energy. In our experiments with the WISP, we found that reaching a given threshold can take less than 10ms or 100s of ms depending on the input power. This variation, combined with the non-trival cost of waking up to take a sample, means that polling at any fixed interval is problematic. If the tag is close to the reader, a long interval means that the tag will store excess energy and miss opportunities to execute tasks. Conversely, if the tag is far from the reader, it will accumulate energy very gradually and pay a disproportionately greater overhead if the interval is short.

To gather energy over a large range of input powers and target voltages, *Dewdrop* uses an exponentially adapted polling interval. Specifically, let V_r be the voltage a tag has gained since it last woke up, and t be the current sleep interval. Then,

$$t_{next} = \begin{cases} 2t, & \text{if } V_s - V > 2V_r \\ t/2, & \text{if } V_s - V < V_r/2 \\ t, & \text{otherwise.} \end{cases}$$

This mechanism is very lightweight because it only involves shift operations to scale the polling interval, not multiply, divide, or floating point operations (which are not likely to be available in hardware). In our evaluation we find it to be responsive, sleeping for short amounts of time at high input power, and to have low overhead, gathering energy out to low input power levels.

5 Implementation

The WISP firmware is written in a mix of C and assembly, for timing sensitive operations. The code can be broken down into two main components: the *Dewdrop* runtime and task support. The *Dewdrop* runtime code must execute quickly and infrequently to reduce overhead. Task support includes the Gen 2 RFID communication protocol, which requires tags to respond to reader commands quickly, generally within 10s of microseconds. This section describes our implementation of a functioning prototype as it relates to these challenges.

5.1 WISP Hardware

The WISP draws approximately 600μA when the CPU is in active mode and 1.5μA when in a state-preserving sleep mode. By default, the WISP wakes up at a fixed power level; a voltage supervisor waits for sufficient power to operate (defined by its capacitor reaching 2V) and then triggers a hardware interrupt to wake the device. We use the term *HwFixed* to refer to this hardware method of waking up at a fixed voltage. *Dewdrop* disables this mechanism and instead uses a timer interrupt to wake the device.

The WISP stores energy in a 10μF capacitor and the voltage of the capacitor can be sampled via its analog to digital converter.[5] If the voltage of the capacitor drops below 1.5V, the WISP will black out and lose all state. We found that the time to fully charge the capacitor varied from 10s to 100s of milliseconds, depending on distance. Discharging a full capacitor to below 1.5V in the absence of a reader signal takes 10s of ms when active, but more than 8s when in sleep mode. Thus, the WISP can carry state across relatively long periods of reader inactivity by sleeping.

5.2 *Dewdrop*

Low power wake-up. *Dewdrop* puts the WISP into a deep sleep state for a specified period to gather energy, and the CPU is woken up by the timer interrupt. The process is repeated until the target wake-up voltage, V_s, is reached. This approximates the behavior of a hardware voltage supervisor, which wakes a device when a specified voltage is reached, but allows us to vary V_s. A potential drawback to this approach is an increased current draw due to keeping the crystal oscillator active to drive the timer, but in practice this increase is acceptably small (2 μA vs 1.5 μA with the crystal off).

Low cost voltage sampling. *Dewdrop* checks the capacitor voltage to see if enough energy has been stored to warrant starting a task, and goes back to sleep if not. The energy overhead of this polling approach is determined by the polling interval and how long the WISP must be awake for each sample. The per sample cost is directly proportional to how long the WISP must stay in active mode. Sampling the capacitor voltage should take 90μs according to the MSP430 data sheet instructions for using the ADC. However, we found that ADC values stabilized much faster—20μs including setup time—with sufficient accuracy (10mV). This shorter awake time drastically reduced the cost of voltage sampling.

Calculating the energy storage rate. *Dewdrop* also tracks how quickly energy is being stored, as it uses this information to adapt the sleep period and to calculate how much time is wasted overcharging. Our adaptive sleep function generally results in a series of sleep periods, where the WISP wakes up and checks its voltage, adjusts the sleep period, and returns to sleep. When a task completes, $V_e - V_o$ tells us how much energy is leftover. We use the last period's charging rate and the average charging rate over all periods to estimate how much time was wasted overcharging. When a task fails,

[5] A 10 μF capacitor is a reasonable trade-off between charge time (a smaller capacitor charges faster) and charge capacity.

$V_s - V_o$ tells us how much energy was wasted. We use the average charging rate to calculate the time wasted undercharging.

5.3 Task Support

Order of operations. The computation and sensing components of tasks must take place before or after communicating with the reader; the deadlines imposed by the Gen 2 protocol are too tight to interleave task processing and message handling. Therefore, in the SENSETX task, for example, the WISP samples the sensor immediately after waking up and then begins decoding reader commands and waiting for the next Query.

Detecting task failures. To avoid blacking out and losing state, the WISP needs to detect when task failures are imminent and then quickly enter sleep mode. In other words, if the voltage drops below $V_o + \epsilon$ (see Section 4), the task must be aborted. In future hardware revisions of the WISP, we would like to trigger an interrupt when a *minimum* voltage threshold is reached. In the meantime, we approximate this behavior by manually inserting calls to the voltage sampling function in the task code. We found that an ϵ of 0.15V was sufficient to protect against blackout. That is, if any voltage sample measures below 1.65V, the WISP will sleep and record a task failure.

Sampling the voltage during the communication phase proved difficult, but it was necessary because message processing is a major factor in energy consumption. The Gen 2 message timing constraints are such that the WISP does not have time to take a sample between messages without losing synchronization with the reader, even with a sampling time of only 20μs. However, we found that we could carefully schedule a voltage sample during the preamble of every reader command, so long as the inspection of the sample was deferred until after the command was decoded. As the WISP must be in active mode to accurately track the preamble, this approach amortizes the cost of keeping the CPU active for decoding. This strategy makes it possible for us to closely track the voltage of the capacitor at every reader command with essentially zero overhead.

Randomness. The Gen 2 MAC protocol requires that tags choose slots randomly. As a source of randomness, we sample the voltage in the capacitor once immediately when the WISP first powers up, and use this value as a seed for a pseudo-random number generator. The variance in this voltage sample, due to input power and noise in the ADC, gives us sufficient randomness. Alternatively, we could have used SRAM state as a random source, with similar efficiency [11].

5.4 Monitoring Support

Monitoring WISP state and operation for debugging and experimentation is difficult. Traditional methods for debugging embedded systems, such as a JTAG connection, would supply power to the WISP and change its behavior. Instead, we use a custom monitoring board we developed for debugging WISPs [19]. The board communicates with a PC via USB, attaches to the debug and other output pins of the WISP, but does not add to or consume energy harvested by the WISP. The monitor board can also sample the voltage in the WISP's capacitor. For our study, we instrument the WISP to toggle debug pins at key points in its operation, and the monitor board records what event happened and immediately samples the WISP capacitor to determine its voltage. This results in a trace of WISP operations from which we can determine task costs, and response rates even for tasks that do not communicate with the reader.

6 Evaluation

In this section, we evaluate *Dewdrop* experimentally. We show that our approach of balancing sources of waste generally achieves 90% of the best possible response rate for the SENSETX and SENSE tasks and across a wide range of RF environments. *Dewdrop* improves performance over the default WISP runtime, providing applications a benefit in terms of both improved coverage and higher response rates.

6.1 Experimental Setup

Our experiments were conducted using an Impinj Speedway RFID reader that continuously transmits energy and commands. This is the normal reader behavior. For experiments involving a single tag, the WISP was placed on a poster board 1m from the reader antenna and the output power was variably attenuated from 30dBm (1 Watt), the maximum allowed for "Gen 2" readers, to 18dBm. This method increases repeatability by limiting the multipath effects that would occur if we moved the WISPs. We present results in terms of an equivalent distance that is calculated using free-space propagation, as we find them to be more intuitive than results in terms of transmit power.

In all experiments, we ran *Dewdrop* and the default WISP hardware, which we call *HwFixed*, that starts tasks at a fixed energy level of 2.0V. *HwFixed* provides a baseline for comparison. When possible, we also report results for *Oracle* as the best result found from an exhaustive offline search of starting energy levels (at which the WISP wakes-up and starts a task) using 0.03V steps. We

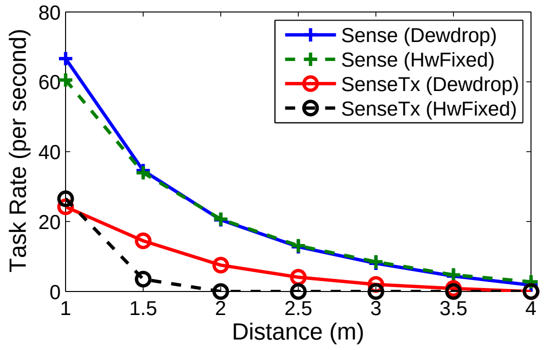

Figure 5: Response rates when using *Dewdrop* and the *HwFixed* runtimes.

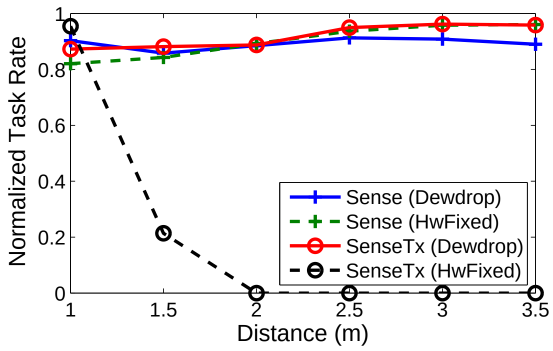

Figure 6: Response rates for *Dewdrop* and *HwFixed* compared to an oracle.

report results for both the SENSE and SENSETX tasks described in Section 4.2.

To evaluate our approach in a realistic deployment, complete with multipath effects, we deployed 11 WISPs with accelerometers on a 1.2m x .75m table of a model apartment at Intel Labs Seattle. This deployment is similar to that seen in [3], though we only consider a single workspace instead of the complete apartment. An RFID reader was installed in the ceiling and equipped with one antenna approximately 2m above the table pointing downwards. We configured the reader to run the SENSETX task to gather samples continuously for one minute. We performed three separate trials for each configuration to allow for variability from both the RF environment and communication protocol.

6.2 Using Energy More Effectively

***Dewdrop* performance.** We first assess how well *Dewdrop* performs compared to *HwFixed* for a single WISP.

Figure 5 compares the response rate of SENSE and SENSETX when using the two runtimes. *We find that the performance of* Dewdrop *consistently matches or exceeds that of* HwFixed. For the light SENSE task, the performance of *Dewdrop* closely matches that of *HwFixed*

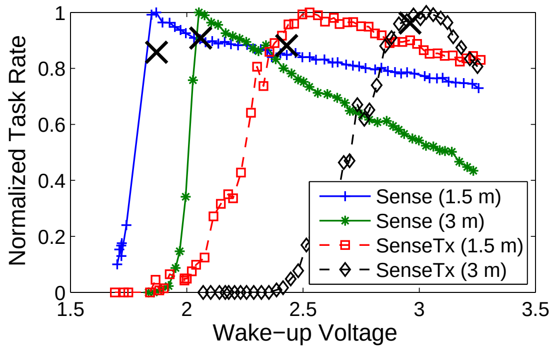

Figure 7: Response rates for both tasks at 1.5 and 3m. *X*'s indicate the operating point found by for *Dewdrop*.

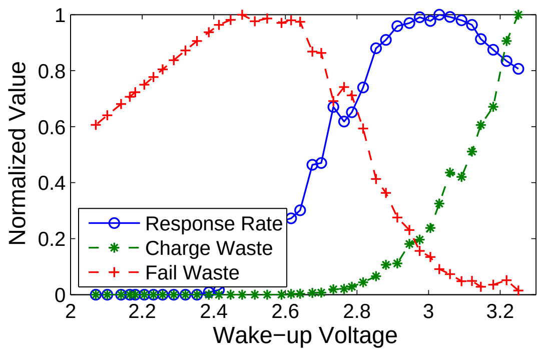

Figure 8: Response rate and wasted time for SENSE and SENSETX at 3m.

and actually performs better at 1m. This is because, at close range, the received power supplements stored energy enough to allow an energy level 0.2V below *HwFixed*'s fixed value.

In the case of the heavier SENSETX task, *Dewdrop*'s response rate decreases smoothly as reader power falls to 3.5m. *HwFixed* fails to execute the task beyond 1.5m. *Dewdrop* adapts to the higher energy requirements of this task, and stores more energy before beginning execution, whereas *HwFixed* does not. This improvement more than doubles the operating range of the tag.

To find an upper bound on how well *Dewdrop* could work, we compare to the *Oracle* results. Gathering this test data takes hours and is thus not a candidate for a practical CRFID runtime. Figure 6 again shows the response rates for the two tasks when using *HwFixed* and *Dewdrop*, but the rates are normalized by the best rates found using the *Oracle*. *We find that* Dewdrop *generally achieves better than 90% of the maximum rate seen by* Oracle *for both tasks.* Interestingly, *Oracle* always beat *HwFixed*. This means that the fixed 2 V energy level was never the best choice.

Evaluating *Dewdrop*'s choices. To understand why *Dewdrop* performs well, we looked at the starting energy

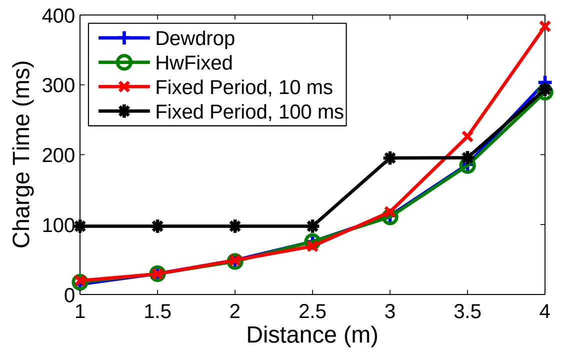

Figure 9: Charging time from 1.5V to 2V.

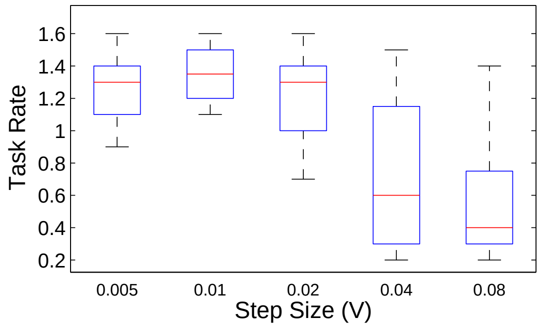

Figure 10: Effect of step size (β) on response rate for SENSETX at 3.5m.

levels it selects. *Dewdrop* must choose starting energy levels that are close to the best level found by the *Oracle* if it is to be efficient. To show that this is a non-trivial task, Figure 7 shows examples of response rate versus energy level curves. The figure is based on data from the *Oracle* for both tasks at 1.5 and 3m.

We see that the best starting energy level varies widely for different tasks and at different distances. For SENSE, the best energy level is 1.9V at 1.5m, when input power close to the reader supplements stored power, and 2.1V at 3m. Similarly, for SENSETX the best level varies from 2.5 to 3V over the same distance. These results emphasize that no fixed threshold will work either for all tasks or for all distances. For example, the best energy level for SENSETX at 3m is 3V. This level achieves only 50% of the maximum response rate for SENSE at the same distance. It is even worse if the best level for SENSE at 3m is chosen, as SENSETX cannot execute the task even once at 3m with an energy level of 2.1V.

The figure also shows the operating points found by *Dewdrop* marked with *X*s. *We see that our runtime finds points very close to the best energy level despite the differences between response curves.* Across all of our data the energy levels found by *Dewdrop* were within 0.1V of the best level found by *Oracle*.

To see how *Dewdrop* selects a good starting energy level, we looked at how it minimizes wasted time. We calculated the average wasted time per task due to failing and due to charging too high. Figure 8 shows this data, along with response rate, for an illustrative case of SENSE and SENSETX at 3m. The data are normalized by their maximum values. We see that as the starting energy level increases, the average wasted time due to failing generally decreases. (The waste is low at low wake-up thresholds despite tasks failing a greater fraction of attempts. This is because waste is computed in terms of time spent charging, and at low wake-up thresholds, very little time is spent charging.) Beyond 2.6V, waste from failed tasks decreases, as the task fails less often. Conversely, the wasted time from overcharging increases with the starting energy level because the energy is stored less efficiently at higher voltages.

Dewdrop seeks the intersection of the two waste curves, and uses the corresponding energy level. This appears to be a good strategy as the maximum response rate in the figure occurs near the intersection. Moreover, since the rates plateau around the maximum, *Dewdrop* can miss its mark by a fairly wide margin ($\pm$0.1V), without affecting performance significantly. Though the figure shows only a single example, *we found the energy level that equalized the two sources of waste generally achieved better than 95% of the maximum rate for both tasks at all distances*.

Evaluating *Dewdrop*'s costs.

This section investigates two possible inefficiencies in *Dewdrop*: the cost of our timer-based adaptive sleep scheme, and the effect of our choice of step size for maintaining the starting energy level. We show that both are efficient, which is in keeping with our runtime performing almost as well as the *Oracle*.

To be effective, our runtime must not appreciably increase charging time. Figure 9 shows the median charging time from 1.5V to 2V for *Dewdrop*'s adaptive sleep mechanism, the hardware wake-up of *HwFixed*, and two strawman versions of our software controlled sleep mechanism that use fixed sleep periods.

We find that, at all distances, our adaptive scheme achieves a charge time within 5% of the charge time of the hardware mechanism. Moreover, as expected, its performance is good over a wider range of distances than schemes that do not adapt their sleep periods. For example, the fixed period of 100ms does well at 4m (1.3% longer than *HwFixed*), but performs poorly at close range (600% longer than *HwFixed* at 1m). Likewise, fixing the period at 10ms works well at close range, but incurs significant overhead farther away (32% at 4m).

The second potential source of inefficiency in our system comes from our choice of step size (β) when seeking the best starting energy level. In *Dewdrop*, upward pres-

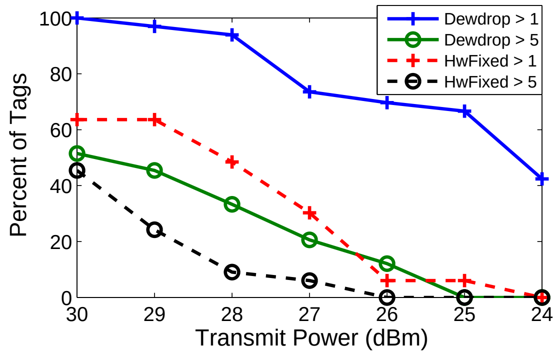

Figure 11: Percent of tags that have an average response rate above 1/s and 5/s using the two runtimes.

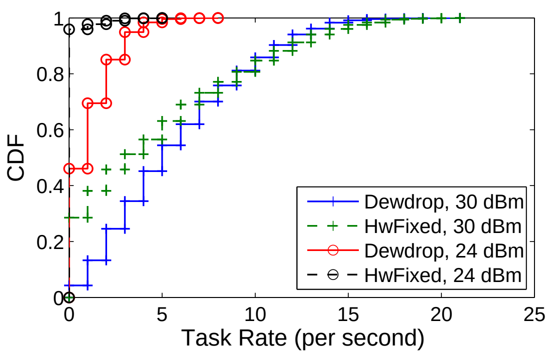

Figure 12: CDF of response rates for the two runtimes as power is reduced.

sure on the level is only exerted after it drops fairly low and tasks begin to fail; after failures, the starting energy level rises until the cost of overcharging outweighs the cost of failing. A small β increases the time it takes to adapt to environmental changes, while a larger β can result in large oscillations around the ideal wake-up threshold.

Figure 10 shows the effect of different step sizes on task rate for SENSETX at 3.5m. The average task rate per second is calculated over a 10 second sliding window. As step size increases, the task rates generally decrease and vary more widely. A larger step size means that *Dewdrop* increases/decreases its starting energy level too quickly, resulting in significant over/undercharging. The reverse then happens and the voltage is reduced by too much and more tasks fail. We found that a step size of 0.01V gave a good balance between damping oscillations in energy level and quickly adapting to environmental changes.

6.3 Multiple Tag Evaluation

Next, we evaluate *Dewdrop* in a realistic deployment consisting of multiple tags. To support CRFID applications such as activity recognition, our runtime should both increase the coverage region of the reader (e.g., so that distant devices respond) and also increase the response rates of the devices (e.g., so that object motion can more accurately be tracked). We consider both of these metrics for the 11 WISPs deployed in the model apartment.

Coverage. The coverage goal is to have as many devices as possible responding at a useful rate. Based on prior experience, we define two useful rates: a rate of 1/s, as is useful for low-rate object use detection; and a rate of 5/s, as is useful for higher-rate gestural recognition. To characterize the coverage of the deployment, the transmit power of the reader is reduced gradually to determine the "headroom" (in dBm) tags have for a given level of performance.[6]

We find that Dewdrop *has much better coverage than* HwFixed *because it enables tags to operate when much less incoming power is available.* Figure 11 shows the percentage of tags with average response rates above 1/s and 5/s when using the two runtimes. At 30dBm, all tags with *Dewdrop* respond at least once per second as compared to 64% with *HwFixed*. Coverage is better even when tags with *Dewdrop* receive one third the power of tags with *HwFixed* (viz., 67% for *Dewdrop* at 25dBm vs 64% for *HwFixed* at 30dBm). Moreover, at a four-fold reduction in power (24dBm), 42% respond with *Dewdrop* while none respond with *HwFixed*.

For a response rate of more than 5/s, the two runtimes perform equally well at 30dBm. This is because *HwFixed* works well when a tag receives good power from the reader. However, *HwFixed*'s coverage decays much more quickly with power than does *Dewdrop*'s coverage, e.g., at 27dBm *Dewdrop* has three times the coverage of *HwFixed*.

Response Rates. Figure 12 shows the distribution of the response rates of the tags when the reader is transmitting at 30 and 24dBm. The rates are computed over one second windows for both runtimes. *We find that* Dewdrop *consistently achieves higher rates, especially for the tags receiving less energy;* 30% of the data points are zero for *HwFixed* versus 5% for *Dewdrop*. *Dewdrop*'s ability to achieve useful rates is even more apparent when the reader transmits at 24dBm and tags are receiving one fourth as much power. *Dewdrop* obtains response rates greater than once per second 30% of the time, as compared to 2% with *HwFixed*. At 30dBm, *Dewdrop* and *HwFixed* achieve nearly the same rates for those tags that receive the most energy; 25% of the data points are above 9/s, and median rates are 5/s and 3/s respectively.

[6]This "attenuation thresholding" technique [10], has been shown to be more appropriate for characterizing RFID deployments than varying distance due to the high sensitivity of RFID to multipath.

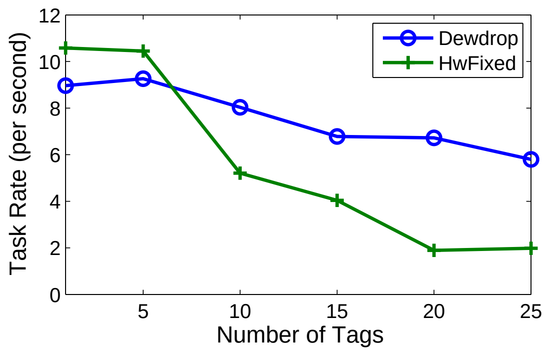

Figure 13: Response rate for the two runtimes as tag population size increases.

When more tags are present, the energy cost of communicating with the reader increases. This is because the reader increases the number of slots it uses to limit the likelihood of tag collisions, so CRFID tags must process more messages before transmitting to the reader.

Figure 13 gives the performance for a single tag when the reader transmits at 30dBm as additional tags are added to the deployment. The performance of *HwFixed* rapidly decreases with the number of tags. This is because the number of slots is increasing, and a tag cannot remain powered when it chooses a later slot. In contrast, *Dewdrop* simply increases its starting energy level to accommodate the additional communication overhead. With one tag, it wakes up around 2.5V whereas with 25 tags it wakes up closer to 3V. The result is that *Dewdrop* provides nearly three times the response rate as *HwFixed* when 25 tags are present.

7 Related Work

There has been significant work on building energy harvesting systems for sensor networks [27, 12, 1]. This work considers solar cells, but some conclusions apply equally to CRFIDs, e.g., [12] finds that capacitors should be used as the primary buffer to tolerate rapid charge/discharge cycles. In [26, 13, 15], the scheduling problem for energy harvesting devices is considered. The scheduling problem for these systems differs significantly from CRIFDs as they manage tasks and harvested power on the order of days, attempt to extend lifetime to months, and have no penalty for storing excess energy. In contrast, Dewdrop must store sufficient energy for a single task execution, and tolerate input power variations on the order of milliseconds in a context where every operation consumes precious energy.

Power management for CRFIDs has generally fallen into two categories; supplying additional energy and maintaining state information across power losses. Alternative methods of powering devices have been explored [16], with [5, 23] proposing solar cells and TV transmitters for CRFIDs. These approaches provide 10's of μW of supplemental power, an order of magnitude below the requirements of current CRIFDs, so energy still must be used efficiently.

In [20], the authors use offline profiling to estimate when state should be saved on the WISP, or transmitted to the reader [22], due to impending depletion of the energy store. We found that simply entering low power sleep mode is an effective way to maintain state, and it avoids the cost of writing to flash or transmitting to the reader in scenarios where the reader does not power off for long periods of time. In [8] the authors use offline modeling to help determine the appropriate capacitor size for a device designed to execute a particular task. While hardware modifications are necessary for tasks with dramatically different energy requirements, *Dewdrop* enables a wider range of tasks to be executed efficiently for any given energy store.

The WISP has been used to demonstrate power intensive applications that would benefit from our approach. RC5 cryptographic primitives were implemented in [4], and both cryptography and sensors have been used to increase the security of implantable medical devices [9], and credit cards [6]. For these applications, the energy requirements were far beyond what could be provided at range, and the studies were done using the WISP at close range. Dewdrop aims to enable such applications to operate more effectively at greater range.

8 Conclusion

We presented a runtime for CRFID tags that makes efficient use of the scarce available energy. Our runtime, *Dewdrop*, adapts a tag's duty cycle to match the harvested power to the sensing and computation cost of tasks. To do this, it estimates the time wasted by overcharging and by underestimating task needs, and uses the result to choose how much energy to buffer before starting a task. Using an implementation built on the WISP tag and a commodity RFID reader, we showed that *Dewdrop* runs tasks where prior techniques could not, and runs them at better than 90% of the best rate found by offline testing across a range of input powers, competing tags, and light and heavy tasks. *Dewdrop*'s adaptation effectively doubled the distance at which a tag executes tasks, which enables practical deployments. In an instrumented living space, all tags responded at useful rate to a single reader in the ceiling as compared to only 64% with fixed buffering. At over twice the distance (one quarter the transmission power), 42% of the tags still responded with *Dewdrop* while none responded with fixed buffering. We believe these performance levels bring us close to realizing a wide range of realistic CRFID applications.

9 Acknowledgments

We thank the anonymous reviewers and our shepherd, Jason Flinn, for their helpful feedback. We would also like to acknowledge the invaluable assistance of Josh Smith and Alanson Sample in helping us understand the design and operation of the WISP. This work was supported in part by NSF award #1016487.

References

[1] D. Brunelli, L. Benini, C. Moser, and L. Thiele. An efficient solar energy harvester for wireless sensor nodes. In *DATE*, 2008.

[2] M. Buettner et al. Revisiting smart dust with RFID sensor networks. In *HotNets*, 2008.

[3] M. Buettner et al. Recognizing daily activities with RFID-based sensors. In *Ubicomp*, 2009.

[4] H. J. Chae et al. Maximalist cryptography and computation on the WISP UHF RFID tag. In *RFID Security*, 2007.

[5] S. S. Clark et al. Towards autonomously-powered CRFIDs. In *HotPower*, 2009.

[6] A. Czeskis et al. RFIDs and secret handshakes: Defending against ghost-and-leech attacks and unauthorized reads with context-aware communications. In *CCS*, 2008.

[7] EPCglobal. EPC radio-frequency identity protocols class-1 generation-2 UHF RFID protocol for communications at 860 mhz-960 mhz version 1.0.9. 2005.

[8] J. Gummeson, S. S. Clark, K. Fu, and D. Ganesan. On the limits of effective micro-energy harvesting on mobile CRFID sensors. In *MobiSys*, 2010.

[9] D. Halperin et al. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *IEEE Symposium on Security and Privacy*, 2008.

[10] S. Hodges et al. Assessing and optimizing the range of UHF RFID to enable real-world pervasive computing applications. In *Pervasive Computing*. Springer-Verlag, 2007.

[11] D. E. Holcomb et al. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In *RFID Security*, 2007.

[12] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *IPSN*, 2005.

[13] A. Kansal et al. Power management in energy harvesting sensor networks. In *ACM Transactions on Embedded Computing Systems*, 2006.

[14] A. Mainwaring et al. Wireless sensor networks for habitat monitoring. In *WSNA*, 2002.

[15] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems.*, 2007.

[16] J. A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 2005.

[17] M. Philipose et al. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 2004.

[18] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *IPSN/SPOTS*, 2005.

[19] R. Prasad, M. Buettner, B. Greenstein, and D. Wetherall. Wisp monitoring and debugging. In *Wirelessly Powered Sensor Networks and Computational RFID (to appear)*. Springer, 2011.

[20] B. Ransford et al. Getting things done on computational RFIDs with energy-aware checkpointing and voltage-aware scheduling. In *HotPower*, 2008.

[21] M. Reynolds and S. Thomas. The blue devil wisp: Expanding the frontiers of the passive RFID physical layer. *WISP Summit Workshop*, 2009.

[22] M. Salajegheh et al. CCCP: Secure remote storage for computational RFIDs. In *USENIX Security*, 2009.

[23] A. Sample et al. Experimental results with two wireless power transfer systems. In *IEEE Radio and Wireless Symposium*, 2009.

[24] A. P. Sample et al. Design of an rfid-based battery-free programmable sensing platform. In *IEEE Transactions on Instrumentation and Measurement*, 2008.

[25] F. Schoute. Dynamic frame length aloha. *IEEE Transaction on Communications*, 1983.

[26] J. Sorber et al. Eon: A Language and Runtime System for Perpetual Systems. In *SENSYS*, 2007.

[27] J. Taneja, J. Jeong, and D. Culler. Design, modeling, and capacity planning for micro-solar power sensor networks. In *IPSN*, 2008.

[28] B. Warneke, M. Last, B. Liebowitz, and K. S. Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 2001.

[29] D. Yeager, P. Powledge, R. Prasad, D. Wetherall, and J. Smith. Wirelessly-charged UHF tags for sensor data collection. In *IEEE RFID*, 2008.

[30] D. Yeager, F. Zhang, A. Zarrasvand, and B. Otis. A 9.2a gen 2 compatible UHF RFID sensing tag with -12dbm sensitivity and 1.25vrms input-referred noise floor. *ISSCC*, 2010.