Continuous Sensing on Intermittent Power

Majid, Amjad

DOI
10.4233/uuid:0ad9f986-8a4a-4f31-a486-d6c44b30605b

Publication date
2020

Document Version
Final published version

Citation (APA)
Majid, A. (2020). *Continuous Sensing on Intermittent Power*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:0ad9f986-8a4a-4f31-a486-d6c44b30605b

CONTINUOUS SENSING ON INTERMITTENT POWER

CONTINUOUS SENSING ON INTERMITTENT POWER

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, Prof.dr.ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on Monday 10, February 2020 at 12:30 o'clock

by

Amjad Yousef MAJID

Master of Computer Science, Delft University of Technology
Born in Baghdad, Iraq

This Dissertation has been approved by the

promotor: Prof. dr. K. G. Langendoen
copromotor: Dr. P. Pawełczak

Composition of the doctoral committee:

Rector Magnificus	
Prof. dr. K. G. Langendoen	Delft University of Technology
Dr. P. Pawełczak	Delft University of Technology
Independent members:	
Prof. dr. G. Merrett	University of Southampton, UK
Prof. dr. ir. G. Dolmans	Eindhoven University of Technology
Prof. dr. ir. D.H.J. Epema	Delft University of Technology
Dr. B. Campbell	University of Virginia, USA
Dr. ir. A.B.J. Kokkeler	University of Twente
Prof. dr. ir. A.J. van der Veen	Delft University of Technology, reserve member

Keywords: Embedded Systems, Energy Harvesting, Battery-free, Intermittently-powered Sensors

ISBN 978-94-6384-105-4

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

Knowledge is the root of all good.

Ali Ibn Abi Talib

CONTENTS

ACKNOWLEDGEMENTS

My Ph.D. journey was a transforming experience. This transformation has been influenced by many extremely talented people. To all of you, thank you!

First and foremost, I have to express my deepest gratitude to my promotors dr. Przemysław Pawełczak (Przemek) and Prof. dr. Koen Langendoen for the selection, guidance, and support. Dear Przemek: your high standards and goals set me well on the path of doing valuable research. Your strategy of collaborating with the world's first-class universities widened my horizon and gained me a vital experience for any future academic life. Dear Koen: working under your supervision was an incredible experience. My meetings with you were never ended when I left the meeting room. My brain kept thinking about something you mentioned or highlighted, which at the end opened up a new direction to me to explore. Dear promotors: thank you very much for shaping me as a researcher.

I also would like to thank Dr. Kasım Sinan Yıldırım for all the discussions we had (and for the grapes that you were offering now and then). My thanks, also, go out to other members of the Embedded and Networked System group: Dr. Ranga Rao Venkatesha Prasad, Dr. Marco Zuniga, Dr. Vijay Rao, Dr. Fernando Kuipers, Dr. Mitra Nasri, Sujay Narayana, Nikolaos Kouvelas, Stef Janssen, Coen van Leeuwen, Jorik Oostenbrink, Weizheng Wang, Belma Turkovic, and the newcomers that I have yet to know. I thank all the master students that I worked with, in particular: Patrick Schilder, Michel Jansen, Carlo Delle Donne, Dimitris Patoukas, Guillermo Ortan Delgado, Henko Aantjes, and Koen Schaper. Special thanks go out to our department and group secretary Janneke Hermans and Minaksie Ramsoekh for their constant support.

Outside our group, I would like to thank Dr. Stephan Wong for hosting me during my last project and for the discussions, Prof. dr. Said Hamdioui for his wise advice, Dr. Zaid Al-Ars for the time and nice chat, Prof. dr. Koen Bertels for sharing his scientific vision with me, Marion de Vlieger for her advice and support, Nauman Ahmed, Tanveer Ahmad, and Toshiki Iwai. I thank my collaborators at other institutes: Prof. Josiah Hester (Northwestern University, IL, USA), Prof. Dr. Joshua R. Smith, Dr. Aaron Parks (University of Washington, WA, Seattle), Prof. Brandon Lucia, Dr. Alexei Colin, and Kiwan Maeng (Carnegie Mellon University, PA, Pittsburg).

Finally, my greatest gratitude goes to my father, the memories of my mother, wife, brothers, and relatives. My father showed me the value of learning and being open-minded; my mother fed me with love and protection. Noor (my wife and life companion): thank you for your unconditional love and endless support. Without you, I would not be able to accomplish what I have done. A big thank you to my father- and mother-in-law for the support and for taking care of our little angel, Ali, every time we needed a backup. Thanks to my brothers and brothers-in-law for their wishes and support. Finally, dear uncle Mohammad thank you very much for being all the time with me. Deep from my heart, thank you all.

Amjad Yousef Majid
Delft, January 21, 2020

SUMMARY

Battery-free energy-harvesting devices have the potential to operate for decades, since they draw power from virtually unlimited energy sources, such as sunlight. However, ambient energy sources are volatile, and tiny harvesters can extract only weak power from them. Thus, small energy-harvesting devices operate intermittently: first, they charge their buffers then start operating, which depletes the buffered energy and causes the devices to power down, letting the harvesters to refill the energy buffers for the next operational round.

Classical programming architectures assume continuous power. Therefore, frequent power failures render them useless; power failures reset the computational progress and delete volatile data. Thus, the intermittent programming and execution paradigm has emerged. Generally, there are two strategies being employed to support intermittent execution: *checkpoint-based* and *task-based*. Prior checkpoint- and task-based systems tackled mainly challenges related to enabling efficient computing on intermittent power. However, they have ignored the challenges associated with sensing, which is the primary application for intermittent systems. Therefore, from a sensing standpoint, these systems have several drawbacks.

Firstly, whilst sensing applications are inherently event-driven, these systems are static; they only allow polling-based sensing (i.e., the software can initiate a sensing action, but the environment cannot). Therefore, these systems are forced to oversample the monitored environment, wasting energy and losing sensing opportunities. Secondly, static intermittent systems face difficulties sizing their tasks (i.e., the size of the code between two checkpoints): on one hand, small tasks are guaranteed to be executed on a single buffer charge, but they drastically increase execution overhead; on the other hand, large tasks impose little execution overhead, but they risk non-termination: the system repeatedly tries executing a task but fails to finish it due to power interrupts. Hardware-dependent checkpointing strategies enable a system to overcome the non-termination problem. However, these strategies face difficulty respecting application-level atomicity constraints. For example, they may collect a checkpoint in-between sensor initialization instructions. If the system fails before placing a new checkpoint, on reboot, the sensor will not be fully re-initialized, and therefore, it becomes inaccessible which may cause a fatal error and crash the running application. Thirdly, there is no prior system that addresses the availability problem of intermittent sensors. A sensor that is frequently off, charging most of the time, has little value and limited potential applications. Finally, intermittent sensors eventually need to communicate the processed data to other layers in the system. Passive sensor-to-sensor communication is a promising candidate to enable ultra-low-power communication between intermittent sensors. However, it suffers from intermittent coverage due to a phenomenon called phase cancellation that happens due to interference between the carrier and backscatter signal.

We address the above-mentioned limitations in this thesis. Chapter 3 introduces InK, an *Intermittently-Powered Kernel* that supports event-driven intermittent execution. In addition to ensuring data consistency and preserving forward progress, InK allows intermittent sensors to sleep in low-power mode waiting for an event to wake them up and trigger the corresponding computational thread, greatly extending their availability. Further, InK supports timer, energy, and hardware interrupts, and it employs preemptive scheduling, which allows it to cancel the computation threads corresponding to stale data. Chapter 4 proposes Coala a task-based system that features a hardware-independent approach for on-the-fly task size adaptation. Its dynamic task-based intermittent execution model uses the execution history as a metric to coalesce static tasks and commits their progress only once at the end of the tasks sequence. This adaptive execution strategy optimizes the granularity of the protection mechanism, speeding up intermittent applications execution. Coala does not only take advantage of favorable energy conditions, but it also splits non-terminating tasks to ensure forward progress. This flexibility enables Coala to support heterogeneous devices without requiring applications to be reprogrammed and compiled. Chapter 5 tackles the availability problem of intermittent sensors. It introduces a new virtual sensor, the *Coalesced Intermittent Sensor* (CIS), which is a group of battery-less energy-harvesting sensors. A CIS exploits the embedded randomization in the powering subsystem to spread the intermittent sensors' on-times, increasing the overall availability of the system. The different power consumption, the volatility of ambient energy, and event arrivals may cause intermittent nodes to synchronize their power cycles, which hampers the overall availability of the CIS. Therefore, the nodes need to be powering-state aware to artificially randomize their response as to preserve the required availability of the system. In Chapter 6, we characterize the performance of a backscatter tag-to-tag (T2T) multi-hop network. For this, we developed a backscatter T2T transceiver and a communication protocol suite. This protocol is based on the new insight that backscatter reception is more energy costly than transmission. Further, we show that multi-hopping is as resilient as single-hop phase-shift technique to the dead spots in backscatter T2T networks, while it extends the coverage of backscatter networks by enabling longer backward T2T links (a tag far from the exciter sending to a tag close to the exciter).

In summary, this thesis tackled several important challenges to enable reliable and efficient sensing on energy-harvesting battery-less sensors. As such we have taken important steps to realizing the dream of exploiting self-powered sensors for futuristic applications like smart and green cities.

1

INTRODUCTION

The vision of smart cities [1], through the use of Internet of Things [2], requires billions of sensors (Figure 1.1). These sensors provide the necessary context to aid people in their daily lives. For example, cars will no longer need to wait in front of traffic lights for non-existing pedestrians to cross the road; doors, upon leaving, will provide people with the latest weather forecast; and jackets will adjust air circulation based on body temperature. However, realizing this vision is no mean feat, as a sustainable energy source for all these sensors is needed

Tethered power requires infrastructure that is expensive, cumbersome to maintain, and restrictive (if not infeasible) for many sensing applications. For example, smart clothes cannot be powered via cables. Batteries, unfortunately, do not provide a viable solution to power all sensors in smart cities. Batteries are usually bulky, for example, bat-

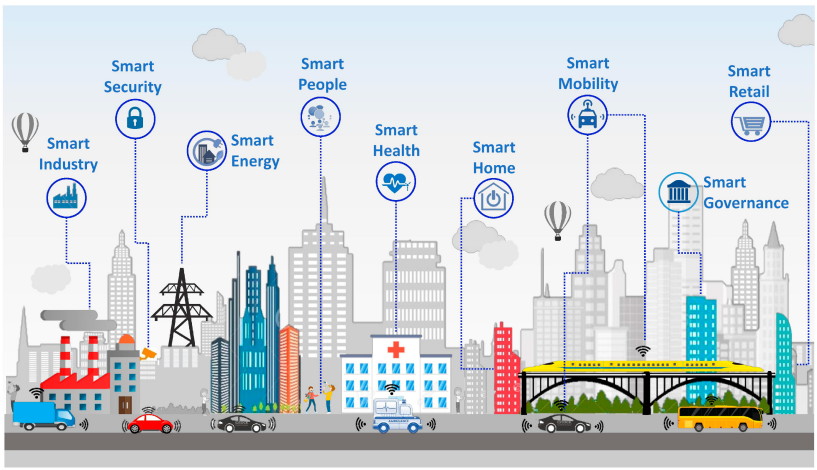


Figure 1.1: Generic smart city environment [3]. It shows that sensors are an integral part of a smart city.

teries are the largest and the heaviest internal part of current smartphones; they occupy more than the half of the volume of a Crossbow Telos mote [4]; and they constitute 32% of sensors fixed on a cyborg [5]. Furthermore, many types of batteries are hazardous, for instance, nickel-cadmium batteries contain, as the name suggests, cadmium, which is a highly toxic metal and exposure to it is known to cause cancer [6]. Moreover, battery replacement can be very expensive, for example, replacing a battery of an implanted sensor requires a surgery [7]. Finally, the raw materials for making batteries are limited. Therefore, many futuristic sensors must leave batteries behind and rely on green, perpetual energy sources.

Thankfully, energy-harvesting technology offers hope! It enables devices to take advantage of ambient energy, such as sunlight, electromagnetic waves, kinetic, and thermal energy [8–11] to power themselves and be energy autonomous. These sources, however, are not always available (an obvious example is the absence of sunlight during night), and their energy intensity is location- and time-dependent. For example, electromagnetic power has a quadratic inverse relationship with the distance between the source and the destination [12]. Despite the volatile nature of these sources, the promise of small form factors and perpetual power motivates researchers to actively work on the development of energy-harvesting sensors [13–16]

1.1. COMPUTING ON INTERMITTENT POWER

An energy harvester of a tiny embedded sensor can only scavenge very limited power from ambient sources [17]. For example, electromagnetic power ranges from nW-scale when harvested from generic sources such as TV and radio transmissions to μW-scale when collected from a dedicated radio wave emitter, and power varies from tens of μW to tens of mW when harvested by a solar panel of a few cm^2 illumination surface [18, 19]. To make efficient use of this weak and volatile power, it is usually buffered using a capacitor (or a super-capacitor). After sufficient energy is accumulated—when the voltage level in the buffer becomes higher than the operational voltage level of the load—the load is enabled to draw energy from the buffer. Often, however, the power consumption of a sensor node outpaces its energy harvesting rate leading to frequent power failures (Figure 1.2). These power interrupts hamper the execution of the software commanding the sensor.

A power interrupt clears the volatile state of a microcontroller; the initialization of peripherals, such as ADCs, DMAs, and UARTs; and the timers. On reboot (when a device powers up again after a power failure) the execution control flows back to the beginning of the program (i.e. `main()`). Therefore, applications running on intermittently-powered platforms have a forward progress problem, i.e., computation is always reset by a power failure.

Non-volatile memory can mitigate the effects of power failures. Microcontrollers used in energy-harvesting platforms feature a mixture of volatile and non-volatile memory [20, 21]. For example, MSP430FR5969 [22] has 2 KB of volatile memory (RAM) and 64 KB of non-volatile memory (FRAM) [23]. Non-volatile memory does not need power to retain its content; thus, it can be used to maintain the execution progress across power failures. Applications, running on intermittently-powered devices, save their progress state in non-volatile memory frequently. On reboot, they resume the execution from the

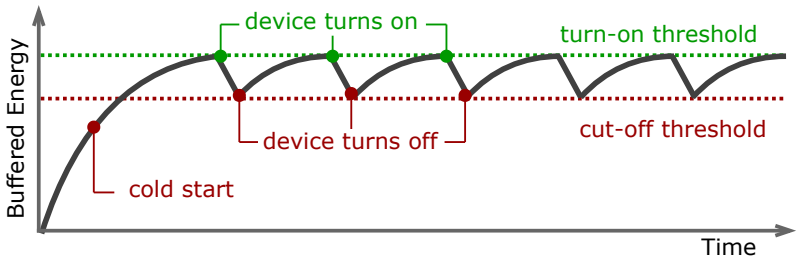


Figure 1.2: Buffered energy profile of an energy-harvesting battery-less sensor node. First, the node is off, and the energy being harvested accumulates up to the turn-on threshold. Upon reaching this threshold, the node is enabled to draw energy from the buffer. Since power consumption of a sensor node is usually much higher than the power harvesting rate, the buffered energy level quickly declines to the cut-off threshold causing the node to be powered down. This cycle of charge-discharge repeats.

last successfully saved state instead of re-executing their programs from the beginning. This type of execution is known in the literature as *intermittent execution* [18, 24].

In general, there are two software-based approaches to enable intermittent execution:

1.1.1. CHECKPOINTS

Systems adopting this approach save the volatile state of a microcontroller frequently in non-volatile memory. The volatile state may include the register file, the stack, the main memory, and the heap depending on the specifications of the chosen technique [24–27]. When the device powers up again, the microcontroller state is restored from the last checkpoint and execution continues.

The checkpoints are distributed throughout the program either statically or dynamically. Static distribution is done either manually [28] or with a help of a compiler [24]. Dynamic checkpointing uses hardware notification to trigger the checkpointing process. When the voltage level at the energy buffer drops below a certain threshold, a monitoring circuitry interrupts application execution and places a checkpoint. To ensure the atomicity of the checkpoint itself a certain amount of energy or memory must be reserved. Systems that monitor the voltage level must allocate a sufficient amount of energy to place the biggest checkpoint [25, 26]. Systems that do not monitor the energy level, on the other hand, must use a form of double buffering [29]. One buffer holds a consistent set of input data to the computation following the current checkpoint, and the second holds the output of this computation. After placing a new checkpoint, the rules of the buffers are switched (atomically).

Data Consistency Problem. Merely saving the volatile state is insufficient to ensure correct execution. The computation state can become inconsistent if an application checkpoints its volatile state and subsequently modifies variables in non-volatile memory. If the application between two checkpoints modifies a persistent variable with Write-After-Read (WAR) dependency and fails—due to a power interrupt—the computation state will become inconsistent on the next reboot. On the reboot, the computation state is

restored from the checkpoint, but variables in non-volatile memory keep their modified values across the power failure. Therefore, the system after the power failure will meet future modifications, which cause the intermittent execution to deviate away from executing the application on a battery-powered device. To overcome this problem, the values of these variables need to be included in the checkpoint. If the system modifies a persistent variable and fails, it undoes this modification using the variable's checkpointed value [28].

Optimized Checkpoints. Copying the volatile state of a microcontroller is an energy-expensive operation. Moreover, it includes copying non-modified data between volatile and non-volatile memory. Therefore, it is considered an inefficient approach. Two broad methods have been suggested to optimize the volatile state checkpoint technique. First, the "light-weight" checkpointing method allocates the stack, the heap, and the global variables are in non-volatile memory [29–31]. Consequently, a system needs only to checkpoint the register file to maintain the computation process. All other modifications are persistent as they happen directly in non-volatile memory. To ensure the correctness of this approach there must not be a power failure between modifying a variable with WAR dependency and the next checkpoint. This can be achieved either by checkpointing the progress at each WAR dependency [29] or ensuring that there is enough energy to place the next checkpoint. Although such an approach reduces the checkpoint size, accessing non-volatile memory is slower and more energy-expensive as compared to volatile memory [22]. Moreover, the systems that checkpoint at each WAR dependency is forced to checkpoint more frequently than what is essentially needed, which is one checkpoint per power cycle. Systems that ensure sufficient energy to place a checkpoint must sacrifice energy for a dedicated buffer monitoring circuit. As a result, this approach may exhibit significant overhead. The second method suggests tracking the differences between the last checkpoint and the changes in the volatile state and copying only the modified data [32]. The benefit of such an approach is application-dependent as it uses some energy tracking the changes in the volatile state to minimize data copying and its associated energy consumption.

1.1.2. TASKS

The second approach requires a programmer (or a compiler) to decompose a program into a chain of small static tasks: a region of code that produces output only if it is completely executed on a single energy buffer discharge [33–35]. The runtime keeps track of the currently running task across power failures. The control flow of the program is explicitly expressed, i.e., each task activates the next task. The output of a task is saved into non-volatile memory and made available as input for other tasks. The input and output of the tasks are separated to prevent data inconsistency issues caused by power losses. Task's local variables are assumed to be volatile. Therefore, if the execution of a task is interrupted, the system can safely roll back to the beginning of the interrupted task. Note that, it does not need to re-execute previous tasks as their output is saved in non-volatile memory, and is unaltered by the interrupted task. As a result, task-based systems do not need to place checkpoints to protect the forward progress of an application.

```
2  void dma_copy_page(uint16_t source, uint16_t destination)
   {
4
     // Initialization instructions
6    DMA1SZ = PAG_SIZE_W;
     DMA1CTL = DMADT_1 | DMASRCINCR_3 | DMADSTINCR_3;
8
     __data16_write_addr(source);
10   __data16_write_addr(destination);

12   // Enable and trigger the DMA.
     DMA1CTL |= DMAEN | DMAREQ;
14 }
```

Figure 1.3: `dma_copy_page()` instructs the DMA to move a data page from one location in memory to another. If the system places a checkpoint in-between the instructions of `dma_copy_page()` and fails, before placing a new checkpoint outside the `dma_copy_page()` function, the application will misbehave after rebooting. The system will instruct the DMA again, but the internal volatile state of the DMA will not be fully reinitialized, making it irresponsive.

1.2. CHALLENGES OF SENSING ON INTERMITTENT POWER

One can conclude from the above summary that the main question researchers have been trying to answer is "how to **compute** more efficiently on intermittently-powered devices?" However, when we ask the question "how to **sense** on intermittent sensors?" their proposed solutions fall short. First, static checkpointing [27, 29, 34] can suffer from data inconsistency if the sensory data introduces WAR dependencies. Second, automatic checkpoint placement [24, 31] faces difficulties with respect to application-level atomicity constraints such as not to checkpoint between correlated sensor readings or peripheral initialization instructions. For example, if the system checkpoints between line 6 and line 13 in Figure 1.3 and powers down before placing a new checkpoint outside the `dma_copy_page()` function, the system after rebooting will fail to access the DMA peripheral as the DMA's internal state will not be fully re-initialized, which makes it irresponsive. As a result, the application will not be able to make sure that the data page has been moved to the intended location. Third, current task-based runtimes [33–35] support sequential and polling-based execution model. Such an execution model prohibits sensors from initiating execution threads. Therefore, task-based runtimes do not consider data communication from a sensor to a task (a task may activate a sensor and get the data, but not the other way around). Consequently, they are doomed to waste energy by actively looking for changes in the surroundings (events) instead of passively waiting for events to trigger the associated execution threads.

Intermittent sensors face other important challenges, namely, the fact that they are frequently off, and the energy cost associated with communication.

Missed Sensing Opportunities. Energy-harvesting battery-less sensors are frequently off, spending most of the time harvesting energy. Therefore, they miss many sensing opportunities, which makes them incompetent alternatives to their battery-powered counterparts. For example, if a voice-controlled energy-autonomous switch used for controlling the light of a room has a duty cycle of 10%, it will usually require repeating a com-

mand (e.g., "light on") many times before changing the state of the light bulbs. Clearly, a switch with such characteristics does not have a commercial value.

A Tiny Energy Buffer and its Constraints on Communication. Communication is an essential part of any distributed system to function. Thus, it is also vital for intermittent sensors to communicate. Active communication requires emitting energy in the air to send messages. Supporting such an energy-expensive operation from a small energy reservoir (i.e., a tiny capacitor) is a poor design decision, as it will further tighten the already minimal energy budget for other functionalities (i.e., sensing and computing). Passive communication, on the other hand, offers a more energy-efficient alternative [36]. Communication, by means of backscattering, enables battery-less tags (tiny sensors) to piggyback their messages on top of existing-in-the-air signals to communicate. It achieves that by leveraging the reflection properties of the antenna to induce systematic changes to ambient signals, modulating information on top of them. A widely adopted system that uses backscattering to enable battery-less devices to communicate is the Radio Frequency and Identification (RFID) system. In this system, an RFID reader—a continuously powered device—emits a signal toward an RFID tag to power it and to enable it to scatter its information on top of the incident signal. Eliminating the need for an RFID reader-like device by enabling the battery-less tags to directly exchange messages (tag-to-tag communication) has clear advantages (e.g., less monetary and energy costs). However, tags primitive capabilities make realizing practical tag-to-tag backscatter networks challenging.

The limited energy available to battery-less energy-harvesting devices put many restrictions on their design. For example, passive RFID tags do not include active RF components such as mixers and oscillators. Instead, they use passive electronic components such as diodes, capacitors, and resistors to ensure minimal energy consumption. This energy-aware design, however, limits the effective communication range between the tags. Furthermore, the dis-locality between the carrier signal generator (e.g., an RF excitor) and the information modulator (the backscattering tag) imposes additional challenges when tag-to-tag backscatter networks are considered. This dis-locality makes the tag-to-tag links non-symmetric [37]: forward links (links going away from the carrier generator) are more powerful than the backward links (links coming toward the carrier generator). Second, it induces dead spots—locations where a receiver tag cannot receive the backscatter signal—in the network. These spots happen when the relative phase-shift between the backscattered signal and the original signal of the carrier generator meet a certain condition [37]. In conclusion, there are still several challenges that need to be tackled before intermittent sensors can gain widespread adoption.

1.3. PROBLEM STATEMENT

Energy-harvesting technology has the potential to enable tiny sensors to operate for decades with near-zero maintenance. However, harvested power is usually weak and volatile; therefore, it must be buffered until sufficient energy is accumulated. Energy buffering takes time, usually much longer than the time needed to consume the buffered energy. Therefore, energy-harvesting battery-less sensors operate intermittently.

Current intermittent programming and executing models have addressed the **computing** challenge on energy-harvesting intermittent platforms. However, they have ignored other challenges associated with reliable and efficient **sensing** on these platforms.

- *Event-driven Execution.* Despite the success of the dynamic and event-driven operating systems for the classical sensor network (e.g. Contiki, and TinyOS), state-of-the-art intermittent runtimes are static and polling-based. They waste energy by actively looking for changes in the environment instead of passively waiting for the events to trigger the corresponding computation threads. Consequently, sensors may not be able to capture events when they happen and change the threads of execution accordingly.

- *Energy-aware Execution.* Task-based intermittent execution models are ambient-energy oblivious. They cannot on-the-fly adjust task sizes according to the available energy. Therefore, if a task size is too big to finish on the maximum buffered energy, they fail to progress. Consequently, they do not facilitate code portability across different devices.

- *The Availability Problem.* Being energy-autonomous is a great feature of energy-harvesting battery-less sensors. However, being off most of the time charging and thereby missing sensing opportunities makes them unreliable sensors that have no value for a wide range of real-world applications.

- *Efficient Communication.* Backscattering is a promising energy-efficient communication means between energy-harvesting sensors. However, it is limited in range and suffers from dead spots in the network, which further restrict the communication range.

Given the aforementioned limitations, this thesis asks the following research question:

What does it take to reliably and efficiently sense on intermittently-powered sensors?

This dissertation addresses the limitations of prior work and fills the void of reliable sensing on intermittent power. It starts by introducing a system-level software architecture to enable *safe* and *efficient* data gathering and processing on intermittent sensors (Chapters 3 and 4). Then, it addresses the intermittent sensors *availability* problem, showing how to meeting applications duty cycles (or on/off cycles) requirements without changing harvesters dimensions (Chapter 5). Finally, it shows the effect of *multi-hopping* on the dead spots and the range of sensor-to-sensor (or tag-to-tag) backscatter networks (Chapter 6).

1.4. THESIS CONTRIBUTIONS

This thesis contains four chapters that detail the way toward reliable and efficient sensing on energy-harvesting intermittent sensors.

- **InK: Reactive Kernel for Intermittent Sensors - Chapter 3.** This chapter introduces the *Intermittently-Powered Kernel (InK)*, the first *reactive* task-based runtime system for battery-less, energy-harvesting sensors. InK eschews the static task execution model, and instead enables, event-driven, and time-sensitive applications for battery-less sensing devices. It features a *preemptive* scheduling policy that enables several (in)dependent *task threads* with different priorities to run in an interleaved manner, responding to energy, time, and sensing events, while preserving correct execution.

  Compared to existing kernels for embedded systems, InK exhibits new properties dedicated to battery-less systems. In particular, InK (i) ensures forward progress of computation by executing restartable atomic tasks encapsulated by task threads each with unique priority; (ii) ensures time constraints of task threads by employing preemptive and power failure-immune scheduling and building a timer subsystem composed of *persistent timers*; (iii) ensures memory consistency during event handling, as interrupt handlers are not inherently atomic and power failures during their execution might lead to memory inconsistencies.

  InK evaluation against state-of-the-art systems shows that it is up to 14 times more responsive to events in realistic intermittent power conditions. InK's event-driven approach has enabled the development of a new category of battery-less applications that are characterized by their reactive feature (e.g., a small intermittent battery-less robot).

- **Dynamic Task-Based Intermittent Execution - Chapter 4**. In this chapter, we introduce Coala: an energy-aware task-based runtime. Coala addresses several challenges to optimize application execution under frequent power failure. Coala's first challenge is how to optimize on-the-fly task size given volatile ambient energy and heterogeneous energy buffer sizes. Coala uses recent execution history as a metric to estimate available energy shots. Then, it coalesces (groups) static tasks accordingly and commits their progress only once at the end of the tasks sequence.

  Merging static tasks on the fly raises the need for dynamic memory-consistency handling. This leads to the second challenge: how to dynamically detect inter-coalesced-task data dependencies and ensure efficient protection against power interrupts? Coala features a novel *Virtual Memory Manager* (VMM). The VMM performs real-time dependency tracking on a coalesced task scope. Individual variable tracking, however, slows down the system dramatically. Therefore, the VMM keeps memory consistent through *privatizing memory pages* and optimizes bulk data transfer through Direct Memory Access (DMA).

  A static task decomposition model assumes that each task can execute to completion. However, if the energy requirement of a task exceeds the energy buffer size, the program will not terminate [38]. This leads to the third challenge: how to enable the dynamic execution model to progress on a sub-task level? To avoid non-termination under adverse energy conditions, Coala uses a timer-based *partial task commit* mechanism. Partial execution avoids non-termination by commit-

ting the intermediate state of a long-running task that has repeatedly failed and restarted.

Comparing Coala's performance to state-of-the-art task-based systems shows that it is up to *two* times faster, and it is able to progress where static systems get stuck in non-terminating tasks.

- **Coalesced Intermittent Sensor - Chapter 5.** In this chapter, we present the Coalesced Intermittent Sensor (CIS), an intermittently-powered "sensor" that senses continuously! CIS is the abstraction of a group of energy-harvesting intermittent sensor nodes. The key observation is that if the power cycles of intermittent nodes are different, then the distribution of their on-times resemble uniformly distributed. As such, the emerging collective behavior of CIS can be modeled and the required number of intermittent nodes to meet a certain collective on/off cycle can be determined.

  An important finding is that a CIS designed for certain (minimal) energy conditions requires no explicit spreading of awake times due to randomness in the power source and node hardware. However, when the available energy exceeds the design point, nodes employing a sleep mode (to extend their availability) do wake up collectively by an external event. This synchronization leads to problems as multiple responses will be generated, and (what is more worse) subsequent events will be missed as nodes will now recharge at the same time. To counter this unwanted behavior we designed an algorithm to estimate the number of active neighbors and respond proportionally to an event.

  We prototype, evaluate, and demonstrate the feasibility of the CIS concept in the form of voice-control application recognizing individual words on solar-powered nodes equipped with microphones. We show that when intermittent nodes randomize their responses to events, in favorable energy conditions, the CIS reduces the duplicated captured events by 50% and increases the percentage of capturing entire bursts above 85%.

- **Multi-hop Backscatter Tag-to-Tag Networks - Chapter 6.** This chapter presents our work on backscatter sensor-to-sensor (or tag-to-tag) networks: a communication technology that has the potential to enable *battery-less wireless sensor networks*. From a tag's energy perspective, backscattering is very efficient communication means. However, tag-to-tag backscatter communication is limited in range, and tag-to-tag networks suffer from a phase cancellation phenomenon that induces dead spots in the network coverage.

  To study how *multi-hop* communication affects the phase cancellation and the range of a backscatter link, we built a fully operational multi-hop backscatter tag-to-tag network. To this end, we built a novel backscatter tag, performed in-depth characterization of tag-to-tag links (i.e., hop count, and per link packet error rate), and developed a novel network stack tailored toward backscattering. Our Medium Access Control (MAC) considers both selection of the phase with which frames are transmitted, and the use of *low-power listening* to conserve the energy of the microcontroller commanding the tag.

Our results show that multi-hopping enables tags to exploit the full range provided by the carrier signal generator, instead of being limited to the ranges of backward links. Moreover, multi-hopping provides a "self-defense" mechanism against the phase cancellation phenomena. As a result, multi-hop backscatter architecture provides a much wider coverage than single-hop backscatter networks, and it mitigates the effect of phase cancellation as good as the phase-shifting technique.

2

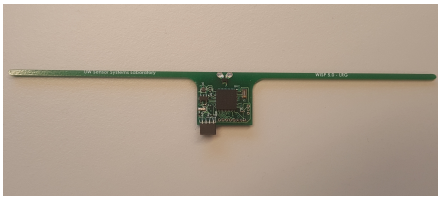
RELATED WORK

This chapter discusses the related work relevant to the problem of intermittent sensing. It reviews battery-less energy-harvesting devices; discusses intermittent computing approaches and highlights their limitations; and investigates a potential means for communication between intermittent sensors.

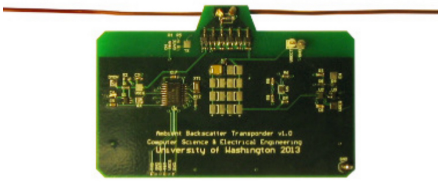
2.1. BATTERY-LESS ENERGY-HARVESTING SYSTEMS

Sensing technology underpins a vast number of applications, such as, monitoring people and structural health, predict disasters, and manage traffic flows. [7, 39–41]. Non-tethered sensing technology was first powered by batteries [4, 42–45]. Battery-powered sensors require regular service for battery replacement to operate longer. Replacing batteries, however, is labor-intensive and impractical for large wireless sensors networks [46]. The battery-replacement problem was mitigated by the use of energy-harvesters and rechargeable batteries as a power source [47–55]. This combination gives birth to the Energy-neutral systems. These systems have the objective to support a desirable performance forever (subject to hardware failure) [56–58]. Unfortunately, even rechargeable batteries wear out after a number of charging cycles and require replacement. For example, Eneloop batteries have a nominal service life of ≈ 2000 charging-discharge cycles [59]. Moreover, batteries are chemical devices that produce toxic waste when disposed [60–62]. Therefore, they require dedicated efforts and specialized tools for safe disposal [63]. In conclusion, batteries and battery management impede the realization of the pervasive sensing that futuristic smart cities require.

Recent advances in ultra-low-power microcontrollers and energy-harvesting circuitry have enabled us to leave batteries behind and build energy-autonomous sensors. These sensors extract power from perpetual energy sources such as light, radio frequency waves, and vibrations [26, 64, 65]. Consequently, they have the potential to provide decades-long sensing with near-zero operational costs. These features have motivated researchers to propose the use of energy-harvesting battery-less devices in many domains such as healthcare [66–68] , environment monitoring [13], energy-efficient buildings [69], and



(a) WISP extracts energy from a dedicated RF energy source [77].



(b) Ambient tag harvests energy from ambient RF signals [17].

Figure 2.1: Battery-less energy-harvesting platforms. WISP is a wireless sensing and identification platform [77]. It can only talk back to a dedicated RFID reader [78]. Ambient tags exchange messages by backscattering them on top of ambient RF signals (e.g., TV transmissions) [17].

human-nature interaction [70]. Moreover, tiny energy-harvesting platforms have begun to be commercially adopted. Some of the companies that are driving the development of the commercial energy-harvesting products are EnOcean [71], Powercast [72], and NOWI [73].

A typical energy-autonomous sensor consists of a general-purpose computing unit (e.g., a microcontroller), one or more sensors, a communication module, an energy harvester, and an energy buffer (i.e., a capacitor) [18]. Such a sensor operates intermittently, as its power consumption is usually higher than its energy harvesting rate. A typical uptime of such sensors is 100 ms [24, 29]. However, depending on the chosen size of the energy buffer and the amount of available ambient energy, the uptime can vary significantly [74]. Similarly, energy consumption varies dramatically depending on the functionality of the sensors. For example, the power consumption of a WISPcam [75] while taking a photo is 74 mW, whereas the FM backscatter tag [76] consumes only 24μW to communicate its sensed information.

Platforms. Many energy-harvesting battery-less platforms with different capabilities have been proposed. WISP (Figure 2.1a) is a commonly used energy-harvesting battery-less platform [79]. It is an RFID tag with a general-purpose microcontroller and an accelerometer [79]. Its microcontroller is the ultra-low-power MSP430FR5969 [80], which features non-volatile memory based on Ferroelectric RAM (FRAM) technology [23]. As compared to Flash memory, widely used non-volatile memory [81, 82], FRAM is significantly more energy efficient [23, 29]; it is, however, less dense, which is less of a concern for energy-harvesting devices. WISP extracts energy from RF signals generated by an RFID reader [78] with a central frequency of 915 MHz. The harvested energy is then stored in a 47 μF capacitor. WISP's harvester allows the microcontroller to draw power when the voltage in the buffer reaches 2.4 V. When the voltage in the buffer becomes 1.8 V the microcontroller powers off, allowing the harvester to recharge the buffer. For communication, WISP relies on backscatter technology. It scatters back the signal emitted by the RFID reader to communicate its information back to the reader. SolarWISP triples the effective communication range of the original WISP by harvesting power from light (instead of harvesting energy from RF transmissions) [64]. Parks *et al.* [83] showed the feasibility of utilizing the RF power emitted by public transmitters (e.g., TV and cellular

towers) to power a battery-free sensor node (or a tag). Then, Liu *et al.* [17] proposed tags (Figure 2.1b) that use public RF transmissions for powering and exchanging messages, using backscattering. Dementyev *et al.* [84] presented a wirelessly powered display tag: an NFC powered tag with an e-paper display. This tag needs power only during a content update. Once the update process is finished, the power source can be removed as the e-paper does not need power to maintain the displayed image on the screen. The tag's content can be updated with an NFC-enabled smatphone. Battery-free cameras have also been developed. For example, WISPcam is an adapted WISP platform with a camera [75]—it accumulates 20 mJ in a 6.08 mF supercapacitor to activate the Omnivision OV7670 camera and takes a photo—and a battery-free video camera [85] that harvests energy from both RF signals and light to capture and backscatter 13 frames per second. Talla *et al.* [86] presented the world's first battery-free cell phone. The phone can be powered by a nearby RF basestation (≈9 m) or ambient light. When the power is extracted from light the cell phone can communicate with a basestation that is 15 m away. KickSat is a battery-free solar-powered nano-satellite [87]. It senses temperature and magnetic field, processes the collected data, and transmits the information back to Earth. Human-powered devices have been also proposed, such as a self-powered push-button [88]; and paper-like interactive materials that harvest energy from touching, tapping, and sliding [89].

Tools. The unique characteristics of battery-less energy-harvesting devices necessitate the development of dedicated debugging and prototyping platforms. CleanCut [38] consists of an intermittent program checker and checkpoints placer. The checker analyses an intermittent program for a non-terminating path: a set of instructions that demand more energy than what the corresponding intermittent device can buffer. CleanCut's placer inserts checkpoints in such paths to eliminate non-termination. Baghsorkhi and Margiolas [90] proposed a closely related system that analyzes a C program and generates multiple intermittent versions of it with different task sizes and empirically selects the version that best suits the corresponding hardware.

Flicker [21] is a modular prototyping platform for developing energy-autonomous devices. It supports federated energy storage, which means that each element has its own energy buffer. Its modular design and smart energy management unit fasten the development process of energy-harvesting battery-less platforms by allowing developers to easily experiment with different components. Capybara [91] dynamically tunes the size of the capacitor (by switching to a larger or smaller one) to a task's energy demand. Capybara's API allows the programmer to associate energy labels (modes in Capybara's terminology) with tasks. Dynamic energy allocation is desirable because a fixed energy storage architecture faces the following dilemma: on one hand, a large capacitor allows executing energy-intensive tasks without a pause for a recharge, but it takes a long time to charge, making the system unavailable for a long interval of time; on the other hand, a small capacitor charges fast, but it does not allow energy-intensive tasks to be timely or atomic. Gomez *et al.* [92] also explores the concept of dynamic energy scaling to minimize the cold start of intermittent systems. Ekho [93] enables the reproducibility of ambient energy conditions. It records ambient energy levels and reproduces them. Using Ekho, energy-harvesting devices can be re-tested on the same power trace for inspection

or comparison between different applications. EDB [94] is an energy-interference-free debugger for intermittent devices. It operates in one of two modes, active or passive. In passive mode, it monitors input-output (I/O) operations between the microcontroller and attached peripherals such as sensors and radios without powering the device-under-test (DUT). In the active mode, EDB gives the developer the ability to check the state of the program and manipulate data. It also supports inserting breakpoints, assertions, and energy guards. Before enabling an interactive debugging session between the DUT and the host, EDB records the voltage level of the energy buffer. During a debugging session, the DUT is continuously powered. Upon exiting the interactive session the target's buffer voltage is restored to the recorded level. Stork [95] is an over-the-air programming protocol for computational RFIDs (CRFIDs) tags. It includes a bootloader the ensures a power-interrupt immune software update (e.g., a firmware update). Aantjes *et al.* [16] presented a testbed for experimenting with CRFIDs. It makes use of an online server, Stork [95], and dedicated RFID readers to enable over the Internet CRFIDs reprogramming.

2.2. INTERMITTENT EXECUTION

Despite the success of the operating systems commanding battery-powered wireless sensor networks [96–99], they cannot operate energy-harvesting battery-less sensors. Frequent power failures reset their compuation progress and render their scheduling and interrupt-handling mechanisms useless. Therefore, new energy-aware runtimes have been proposed and built. Dewdrop [100] was the first runtime to use unstable harvested energy to run tasks. A task is considered a short program that should complete without a pause: putting the microcontroller into sleep mode for energy recharge. To maximize the chance of successful task execution, Dewdrop goes into low-power mode until sufficient energy is accumulated. To not overcharge the energy buffer, wasting time and energy, Dewdrop adjusts the charging time based on the rate of successful task executions and the execution time. QuarkOS [101] divides a given task (i.e., sending a message) into small segments and sleeps after finishing a segment for energy recharge. Both systems, however, are not power-failure immune since they do not maintain the computation state across power failures. Consequently, if a system fails and reboots—because the rate of harvesting is less than the energy consumption rate—all volatile state is lost and the execution is reset to the beginning of the program.

Ambient energy is volatile and scarce. For example, light intensity can differ by orders of magnitude depending on the characteristics of the source and the environment [102, 103]; RF signals are affected by many factors such as the noise, interference, multipath, and movements, and therefore, it is constantly changing across a wide interval. As a consequence, tiny energy-harvesting sensors operate intermittently. Intermittent operation necessitated the development of software that allows the execution to span across power failures [24]. We can classify the proposed intermittent software architectures under two broad approaches: the checkpoint-based approach and task-based approach.

2.2.1. CHECKPOINT-BASED INTERMITTENT COMPUTING

The first approach to support intermittent execution is based on the concept of *checkpointing*: the volatile state of the microcontroller unit is frequently backed up in persistent memory. MementOS [24] was the first runtime that enabled applications to span their execution over power failures. To this end, it periodically saves (or checkpoints) the volatile state (i.e., register file, and main memory) of a program into non-volatile memory—a form of memory that does not need power to retain its content, such as FRAM [23]. MementOS compiler injects trigger points (or function calls) at functions returns and in loops to check the voltage level of the energy buffer. If the voltage level is below a certain threshold, MementOS checkpoints the application's progress. It also includes a timer-aided mode, optimizing the rate at which trigger points check the voltage level. On reboot, MementOS resumes the application execution from the last successfully saved checkpoint, instead of restarting from the beginning of the program.

HarvOS [27] operates at compile time to enable application execution on intermittent power. It divides the Control Flow Graph (CFG) of a program into subgraphs. Each subgraph is analyzed and instrumented with a trigger point at the location that corresponds to the smallest checkpoint needed to maintain the computation state across power failures. At each trigger point, the voltage level is checked. HarvOS places a checkpoint only if it concludes that the remaining energy is insufficient to reach and place the next checkpoint.

Hibernus [25] proposed an event-driven approach—as compared to MementOS and HarvOS polling-based approaches—to checkpoint the volatile state of a program. It takes advantage of the internal on-chip comparator to notify the runtime about the voltage level of the energy buffer. When the voltage drops below a predefined threshold (the hibernation threshold), Hibernus snapshots the progress of the running application and puts the microcontroller into sleep mode. Once the energy level has risen again and surpassed a certain threshold (the restore threshold), Hibernus resumes the execution of the application from the checkpoint. If it fails to collect a complete checkpoint, it restarts the application from the beginning. Hibernus++ automatically adjusts the hibernation threshold when a non-valid checkpoint is found [26].

DICE (DIfferential ChEckpointing) [32] reduces the amount of data being copied during a checkpoint. Once an initial checkpoint is placed, DICE records the modified data cells in the main memory and updates only the corresponding slices of the last checkpoint. DICE optimizes its modification tracking technique based on the variables' context. Global variables are tracked on an individual level, reducing the checkpoint size. Local variables of a function are tracked on a stack frame level as they are likely to be updated frequently during the function execution, reducing tracking overhead.

DINO [28] showed that checkpointing the volatile state alone does not ensure data consistency when the intermittent program accesses non-volatile memory. In particular, non-volatile data structures can become inconsistent when they have Write-After-Read (WAR) dependencies. Figure 2.2 illustrates how the state of a data structure becomes inconsistent in intermittent execution using a simple example of an average operation over an array of integers. The non-volatile variable `sum` introduces the WAR, being read and written sequentially by the increment operation. If a power failure occurs right before updating the non-volatile index `i` (Figure 2.2b, Line 6), `sum` gets erroneously incremented

```
1  NV int i, sum, x[];
2  i = sum = 0;
3  while (i < N) {
4      checkpoint();
5      sum += x[i];
6      i++;
7  }
8  sum = sum / N;
```

(a) WAR-affected code

```
1  NV int i, sum, x[];
2  sum += x[i]; // i = 0
3  i++;
4  checkpoint();
5  sum += x[i]; // i = 1
6  ↯   // power failure
7  ↻  checkpoint();
8  sum += x[i]; // i = 1
```

(b) WAR-affected execution

Figure 2.2: WAR dependency example. `NV` marks non-volatile variables, `checkpoint()` is a checkpoint of volatile state.

twice consecutively by the same array element (Figure 2.2b, Lines 5 and 8). DINO overcomes this problem by offering an API to programmers to include problematic persistent variables within the checkpoints preceding them. Consequently, on reboot, both the volatile and non-volatile state of a program are rolled back to the state of the last checkpoint and made consistent.

Ratchet [29] eliminates the need for hardware support—to monitor the energy level before checkpointing—and programmer intervention—to instrument the code. Instead, it leverages the information available to the compiler to preserve correct execution under frequent power interrupts. Ratchet analyzes a program code to extract idempotent code sections: code segments that do not have WAR dependencies. Then, it places checkpoints at the beginning of these sections. Checkpointing the entire volatile state of a program is an energy-expensive solution [27]. Therefore, Ratchet [29] proposed the use of non-volatile memory as the main memory. As a result, Ratchet needs only to checkpoint the processor register file (i.e., general-purpose registers, the stack pointer (SP), and the program counter (PC)) to preserve the forward progress of the running application. However, the protection mechanism of such a compiler-based approach does not reason about energy availability. Consequently, it generates inadequate idempotent regions, leading to significant performance degradation. While, too small idempotent regions force the system to checkpoint more than necessary (i.e., more than once per power cycle), too large idempotent regions may cause the system to get stuck in a non-termination problem: repeatedly failing to reach to the next checkpoint because the energy needed, to do so exceeds the maximum capacity of the energy buffer. Ratchet overcomes the non-termination problem with help of timer-based technique. However, it is an inefficient approach as the system must first fail several times before the timer mechanism kicks in. Even worse, Ratchet proposed a fixed interval timer that does not completely eliminate the non-termination problem. Using non-volatile memory as the main memory is also a disadvantage. Writing to non-volatile memory is slower and more energy-expensive as compared to volatile memory access (Figure 2.3).

QuickRecall [30] adopted a similar approach to that of Hibernus—-monitoring the energy buffer and placing a checkpoint only when the voltage drops below a certain threshold. It, however, requires a unified main memory (similar to Ratchet). Therefore,

it features a constant voltage threshold as the size of the checkpoint is fixed (only the register file). QuickRecall utilizes modified linker script to map all the memory sections of the binary file (i.e., .bss, .text, .data, and .stack) into non-volatile memory. Therefore, to checkpoint the forward progress, QuickRecall needs only to save the register file into non-volatile memory. However, since all the data modifications are persistent QuickRecall's correctness hinges on the following condition: there must be no execution between a checkpoint and a power failure. Therefore, QuickRecall spin-waits after a checkpoint until sufficient energy is accumulated again.

Recent work have addressed the limitations of prior work from a sensing perspective [104–107]. Briefly, compiler-based (static) systems may suffer from data consistency problems if accessing a peripheral leads to a WAR dependency with a persistent variable. Hardware-dependent (dynamic) checkpointing may break the correctness of applications by placing a checkpoint in sensitive locations such as between peripheral initialization instructions and the instructions to access it. If the system checkpoints in such locations and fails, on reboot, the system will attempt to access an uninitialized peripheral (i.e., an unavailable peripheral), which may lead to a fatal error that crash the running application. To overcome these problems Maeng and Lucia [104] proposed the use of dynamic checkpointing with atomic regions. Before these regions, the system collects a checkpoint and then disables checkpointing to enable safe peripheral access (first initialization, and then utilization). However, in sensing scenarios hardware interrupts (e.g., timer interrupts) are extremely common. The system after disabling checkpointing may be requested to serve an ISR (Interrupt Service Routine) before executing the atomic region (or after it, but before placing a new checkpoint). Now, the system is unable to checkpoint its progress and is requested to execute, which violates the correctness condition of the dynamic checkpointing systems. From a science fiction perspective, [104] is a broken time machine, as detailed in [108] and highlighted in Figure 2.2. Furthermore, in sensing scenarios, nodes spend most of the time in low-power mode waiting for an external event to wake them up. Therefore, monitoring the energy buffer results in significant energy waste [29]. As opposed to the approach presented in [104], Branco *et al.* [106] adopted a static checkpointing based system to enable safe asynchronous operations on intermittent devices. The presented intermittent system features a middle-ware layer interfacing between the drivers of peripherals and the running application to ensure safe I/O operations. Furthermore, it is able to roll the peripheral state forward and the computation state backward to ensure the consistency of an intermittent application after a power failure. A potential point of concern of this system is the size of the checkpoint, as each peripheral is represented by a state-machine and a queue that must be added to the program context during checkpointing. Such a checkpoint technique may scale poorly and/or consume a significant amount of energy to maintain the application state. The checkpoint size limitation has been addressed in [109, 110] through the use of a paged memory management system.

To summarize, current checkpointing systems (except for [106]) make it difficult for the programmer to respect application-level atomicity constraints. For example, if the application-level logic is to sample a signal four times and compute their average, then checkpointing between these four samples breaks this logic as the time intervals between these samples are not the same. Moreover, a checkpoint-based system may check-

Approach	Model	Data Copied to/from NVRAM
Checkpointing	Mementos [24], HarvOS's [27]	Registers + Stack + global variables
	Hibernus [25, 26]	Registers + all volatile state (i.e., SRAM)
	DINO [28]	Registers + Stack + WAR NV variables
	Ratchet [29], Clank [31], QuickRecall [30]	Registers (requires NV main memory)
Task-based	Chain [33], InK [111]	PC + NV variables used in task
	Alpaca [34]	PC + WAR NV variables used in task
	Region Formation [90]	Registers + Updated variables in task
	Coala [74]	PC + privatized pages

Table 2.1: Non-volatile memory access for data consistency; PC: program counter, WAR variables: variables involved in WAR dependencies, NV: non-volatile.

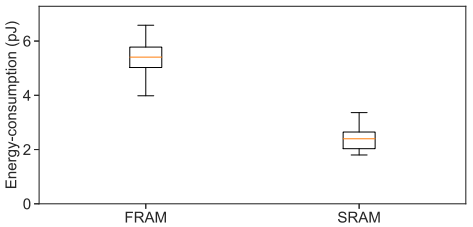


Figure 2.3: Energy cost of writing to the FRAM and SRAM memory of the MSP430FR5969-based platform, WISP [20]. The reported values are the mean energy consumption of 1000 `write` operations. These values were obtained using EDB [94], and each measurement was repeated 100 times.

point between the initialization of a peripheral and its access instructions. Such a checkpoint may cause the system to fail if the power is interrupted right after the checkpoint as the peripherals do not maintain their internal state across power failures.

2.2.2. TASK-BASED INTERMITTENT COMPUTING

The second approach to support intermittent execution is based on the concept of *static tasks*. Task-based programming and execution models require a programmer [33, 35] or a compiler [34, 90] to statically decompose a program into a collection of tasks. A task is a section of code with well-defined input and output. It does not expose output to other tasks until it is completely executed. Once a task execution is finished, its output is saved in non-volatile memory using a form of two-phase commit to ensure that non-volatile data is updated in an atomic manner. The tasks' execution flow is explicitly defined by the user. Therefore, the runtime needs only to maintain a persistent pointer to the currently running task to span the computation progress over power failures.

Chain [33] proposed a programming model that requires a programmer to construct their software into a chain of idempotent tasks. Tasks should use Chain's persistent input-output channel abstractions to ensure data consistency. Mayfly [35] tackled the data freshness problem under intermittent execution. It extends Chain with timing functionality. It does that by using an external (RC-based) timer to maintain the notion of time across power failures. Further, it associates a timestamp with each data channel. These timestamps are used to determine the freshness of the data. As compared to Chain's

non-volatile memory interface, Alpaca's [34] interface is more efficient. It is based on automatic privatization and redo-logging. Alpaca's compiler identifies the data structures with WAR dependencies and provides private task-local backups for them to preserve task idempotency. Table 2.1 summarizes the data that get copied to and from the non-volatile memory in service of memory consistency and progress preservation. The differences in the amount of data being backed up is justified by applications' requirements (e.g., accessing non-volatile memory or not) and hardware capabilities (e.g., monitoring voltage level).

The protection mechanisms of the task-based systems are ambient-energy agnostic. Therefore, they must be conservative and use small tasks to avoid being stuck trying to execute a non-terminating task: a task that requires more energy than the capacity of the energy buffer. Moreover, they support only sequential execution, which is ill-suited for sensing applications that often include event-style (interrupts) processing. However, the concept of a task well-suits the development of intermittent sensing applications, as it enables a programmer to respect applications atomicity constrains despite power interrupts. Therefore, our proposed runtimes are also based on the concept of tasks.

2.2.3. HARDWARE ARCHITECTURE FOR INTERMITTENT EXECUTION

Integration of non-volatile memory to the processor architecture ensures immunity to power loss [112], which removes the burden of explicit checkpointing and recovery with software. Researchers have proposed several non-volatile processors concepts that integrate different non-volatile memory technologies. For example, FRAM-based processors were presented in [113–115]; the benefits of MRAM-based non-volatile processors was discussed in [116]; and the concept of ReRAM-based processors was introduced in [117]. Such processors are emerging especially for energy-harvesting scenarios in which the available power supply is unstable [118]. However, these processors are still in the experimental stage [26, 31]. Therefore, the work in this thesis targets conventional off-the-shelf processors that have both volatile and non-volatile memory.

2.3. COMMUNICATION FOR ENERGY-AUTONOMOUS SYSTEMS

In contrast to active communication where a device emits its own energy to communicate, passive communication enables devices to communicate by means of reflected power [119] (currently, known as backscattering). Radio frequency and identification (RFID) systems take advantage of this energy-efficient communication means to enable battery-less tags to communicate with the energy exciter, the RFID reader [120]. However, an RFID tag has a dedicated purpose; therefore, it is equipped with an application-specific integrated circuit (ASIC) that has very limited capabilities.

Sample *et al.* [77] presented a Computational RFID (CRFID) tag: An RFID tag with a general-purpose microcontroller. They also added sensing capabilities to their proposed CRFID, WISP, which sparked the idea of a battery-less wireless sensor network (WSN). However, realizing a battery-less WSN based on CRFIDs faces two major obstacles: (i) CRFIDs do not directly exchange messages; and (ii) CRFID tags are rendered unusable without RFID readers, providing power and instructions. Nikitin *et al.* [121] tackled the first problem. They showed the feasibility of direct tag-to-tag (T2T) communication in

the context of papers tacking, using RFID paperclip tags, in a smart office environment. In [122] the hardware design of a reader tag—a tag capable of sending commands that are normally sent by an RFID reader—was presented. However, this tag-to-tag communication system still requires a dedicated energy source. This problem was addressed in [17]. The authors presented the design of a tag that can backscatter on top of signals generated miles away from a TV tower to communicate with other tags. Ryoo *et al.* [123] discussed the reported energy level presented [17] and proposed a new tag architecture and showed that their tags are able to communicate at double the distance reported in [17]. The tag-to-tag range of the ambient backscatter system was dramatically improved by using multi-antenna cancellation and CDMA coding techniques tailored towards backscattering [124]. Researchers have also proposed tag architectures that are capable of backscattering on top of signals emitted by widely adopted active RF devices such as Bluetooth, WiFi and LoRa [36, 125–128].

Shen *et al.* [37], Ryoo *et al.* [123] showed that tag-to-tag networks suffer from a unique problem that they call the phase cancellation problem. Basically, if the relative phase shift difference between the signal from the exciter and the backscattered signal meets a certain value the backscatter signals representing the logic “1” and the logic “0” will be indistinguishable. Consequently, the receiver tag cannot detect the backscatter signal at this location (a dead spot). The authors suggested to backscatter on two (or more) phases to elevate this problem. This solution, however, sacrifices half of the bandwidth.

To summarize, most of the presented intermittent systems focus on enabling safe computing on intermittent power supply. However, the main expected application of intermittently-powered devices is sensing. Sensing requires reliable data gathering, on the individual level, and it is at odds with intermittent operations, on the application level. Furthermore, intermittent sensors have access to a small energy budget per power cycle. Therefore, communication by means of backscattering is preferred. However, backscatter tag-to-tag networks suffer from dead spots and weak backward links. This dissertation takes the first steps toward reliable sensing on intermittent power, and it promotes the backscatter communication paradigm to enable backscatter-based intermittent sensor networks.

3

INK: A REACTIVE KERNEL FOR INTERMITTENT SENSORS

3.1. INTRODUCTION

Tiny energy-harvesters power battery-less sensor nodes intermittently. Intermittent execution introduces several new challenges, such as preserving the computation progress and maintaining a consistent set of data. As has been discussed in Chapter 1 and 2, prior intermittent systems [24, 26, 27, 30, 33–35], dealt with these challenges from a computation standpoint: they do not consider the scenario where a change in the environment triggers a computation thread. Consequently, these systems progress in a sequential and surrounding-oblivious manner. Such an execution model is ill-suited for sensing applications: it forces the system to frequently poll sensors' state (and thereby wasting energy), instead of sleeping in low-power mode waiting for the sensors to trigger the execution when an event arrives (saving energy and thereby increasing the probability of successful sensing). Making these systems event-driven, i.e., reactive to environmental changes, is not straightforward because they completely miss a control layer that alters the execution thread when necessary, and they do not provide means (e.g., language abstractions) for safe sensory data reception.

On the other hand, instrumenting the current event-driven operating systems of wireless sensor networks (e.g. TinyOS [96] and Contiki [97]) with checkpoints is insufficient to ensure safe intermittent execution. Power failures, in particular during interrupt handling, can leave non-volatile memory partially updated, leading to data inconsistency problems. For example, if a signal is sampled multiple times, and the power is interrupted while updating the corresponding data structures in non-volatile memory, then the saved data does not reflect the current environmental (sensor-environment inconsistency).

In this chapter we introduce the *Intermittently-Powered Kernel (InK)*, the first reactive task-based run-time system for battery-less, energy-harvesting sensors. InK extends the

Parts of this chapter have been published in ACM SenSys'19, Shenzhen, China [111].

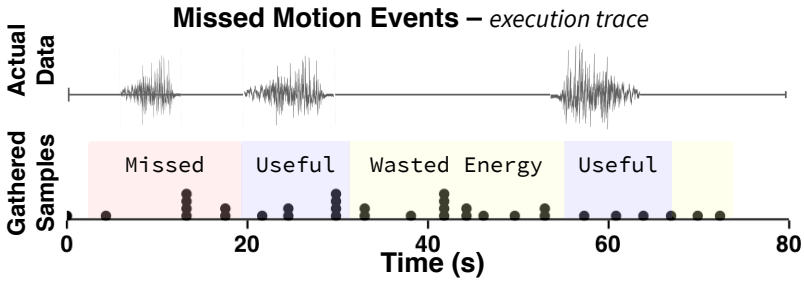


Figure 3.1: State of the art Intermittent programs, based on e.g. Chain [33], Alpaca [34], Mayfly [35], opportunistically gather data, missing important events and wasting scarce energy.

Model	Control flow	Mem. type	Journaled Data	No Dedicated HW	ISR interaction	Concurrent Apps.	C1	C2	C3	C4
TinyOS [96], Contiki [97]	Task-based	DRAM + Flash	None	✓	✓	✓	✓	✓	✓	✓
Dewdrop [100]	Task-based + Scheduler	SRAM + FRAM	None	✓	✗	✗	✓	✗	✓	✗
Mementos [24]	Instruction-based	SRAM + FRAM	Reg. + SRAM	✓	✗	✗	✓	✗	✓	✗
DINO [28]	Instruction-based	SRAM + FRAM	Reg. + SRAM + NV vars.	✓	✗	✗	✓	✗	✓	✗
Hibernius++ [26]	Instruction-based	SRAM + FRAM	Reg. + SRAM	✗	✗	✗	✓	✗	✓	✗
QuickRecall [30]	Instruction-based	FRAM	Reg.	✗	✗	✗	✓	✗	✓	✗
Ratchet [29]	Instruction-based	FRAM	Reg.	✓	✓	✗	✓	✗	✓	✗
Clank [31]	Instruction-based	FRAM	Reg.	✗	✗	✗	✓	✗	✓	✗
HarvOS [27]	Instruction-based	SRAM + FRAM	Reg. + SRAM	✓	✓	✗	✓	✗	✓	✗
Chain [33]	Task-based	SRAM + FRAM	PC + Channel data	✓	✗	✗	✗	✗	✗	✗
Alpaca [34]	Task-based	SRAM + FRAM	PC + NV vars.	✓	✗	✗	✗	✗	✗	✗
Mayfly [35]	Task-based + Scheduler	SRAM + FRAM	PC + Edge data	✓	✗	✗	✗	✓	✗	✗
InK (this work)	Task-based + Scheduler	SRAM + FRAM	PC + NV vars.	✓	✓	✓	✓	✓	✓	✓

Table 3.1: A comparison of relevant models to program embedded devices. Among them, InK is the only one that overcomes challenges C1–C4 (Section 3.1.1); NV vars.: *variables in non-volatile memory* , ISR: *interrupt service routine*, Mem.: *memory*, PC: *program counter*, Reg.: *registers*.

C language with several new abstractions (e.g., tasks, task threads, and persistent pipes) to enable event-driven and time-sensitive intermittent applications.

Compared to existing kernels for embedded systems, InK exhibits new properties dedicated to battery-less intermittent systems (see Table 3.1). In particular, InK (i) ensures forward progress of computation by executing restartable atomic tasks encapsulated by *task threads* each with unique priority; (ii) ensures time constraints of task threads by employing preemptive and power failure-immune scheduling and building a timer subsystem composed of *persistent timers*; and (iii) ensures memory consistency during event handling, as interrupt handlers are inherently not atomic and power failures during their execution might lead to memory inconsistencies.

3.1.1. EVENT-DRIVEN SENSING CHALLENGES

First, we start by discussing in details the challenges associated with using state-of-the-art intermittent programming models: Chain [33], Alpaca [34] and Mayfly [35], to implement an event-driven application with three threads of execution listed in Figure 3.2: `TH1`–`TH3`.

C1–Responding to Events: With Chain, Alpaca and Mayfly these three task threads cannot operate concurrently. To enable event response, another task that constantly polls the energy level (`TH1`), the motion (`TH2`), and the elapsed time (`TH3`) has to be inserted that can trigger the execution of these threads. However, as shown in Figure 3.1 where

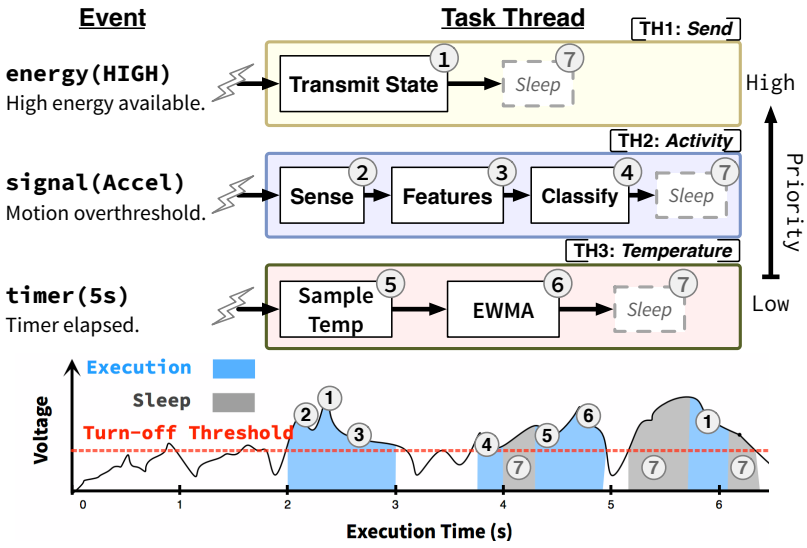


Figure 3.2: An InK sensing application that measures and sends data depending on available energy, motion triggers, and the output of a power failure-resistant timekeeper. The simulated task execution trace is shown. InK is the first event-driven runtime for battery-less, energy harvesting sensor networks. InK fairly schedules concurrent task threads that span power failures and respond to events—like high energy availability, hardware interrupts, and elapsed time. InK sleeps when there is nothing to do, storing harvested energy for the most impactful tasks. EWMA: Exponentially Weighted Moving Average.

an application samples an accelerometer, events can be missed and energy wasted. Task-based and checkpoint-based programming models for intermittent computing are rigid in their specification and inherently non-reactive. To approximate event-driven sensing, tasks frequently *check* shared global variables in order to change the control flow when events have been captured. Such *polling*-based decision making puts extra effort on the programmer, wastes considerable amounts of energy, might interrupt timely responses and also breaks the memory model—see C3. In order to respond to events in a timely manner, battery-less systems require a dynamic scheduling mechanism that can switch between different threads at runtime. However, this is not an easy task since the scheduler itself should work correctly despite power failures, ensure forward progress of computation, and maintain memory consistency of the running threads—a crucial difference as compared to the existing schedulers.

C2–Scheduling Tasks: Aforementioned programming models *cannot schedule events in the future, or perform periodic sensing tasks* as in TH3. Scheduling tasks using one-shot or periodic timers is a common action in battery-powered sensors with a reliable notion of time and persistent power. Again, shared global variables need to be polled continuously by tasks in order to detect periodic events. Scheduling events, like sampling an accelerometer, are a way for a developer to gather information or perform a task at the exact planned moment in time. Without this ability, intermittent programs are doomed to oversample at the wrong time, miss important events or actions in the environment,

and waste precious energy and compute resources. Keeping track of time is challenging as compared to general-purpose embedded systems. To schedule tasks, battery-less systems need a power failure resilient timer subsystem that will not lose track of time despite intermittent power. Scheduling mechanisms can leverage this subsystem to make scheduling decisions.

C3–Handling Interrupts: The memory model of Chain, Alpaca and Mayfly allow tasks to access internal non-volatile memory via input/output abstractions (e.g. *channels* in Chain). Global memory is not accessible to tasks: each task can only read the outputs of predecessor tasks and write to the inputs of the successor tasks. By this means, memory consistency issues are avoided. As a consequence, sharing global variables among tasks and interrupt service routines *breaks existing memory models*—making event-driven applications infeasible and continuous sensing to be the only approach. To support event-driven applications, interrupt service routines (ISRs) should be able to activate tasks. A reactive scheduler is required that eliminates polling and abstractions are needed to let tasks receive data from ISRs without data races and breaking memory consistency. All these issues, in particular memory inconsistencies arising from the fact that ISRs are not atomic and in turn re-executable, are unique challenges belonging to intermittent systems.

C4–Adaptation: With Alpaca, Chain, and Mayfly *tasks run a high risk of starvation* as control flow cannot be interrupted or changed based on the changing external environment, or programmer insights. Tasks in intermittent computing applications always run the risk of starvation; sometimes there is not enough energy in the environment to power any computation, but the rigid task models of the state-of-the-art make this more likely. If a single link in the chain of tasks is never able to complete, either because of low energy or a programmer error, then all subsequent tasks starve. Consider that sometimes applications have multiple actions that can be done at any given time: with current programming models, these actions must be put in a sequence with rigid control flow. Developers have few avenues to respond to this; they cannot bake in runtime logic to handle changing and unpredictable energy situations or new application requirements.

Contributions: In this chapter, we make the following contributions to the intermittent domain:

1. *Event-Driven Programming:* we introduce a new programming model and several new abstractions for intermittent computing; comprised of *task threads* with different priorities, inter- and intra-thread control flow declarations, inter-thread communication interfaces, event notification and handling mechanisms, and time management.
2. *Reactive InK Runtime:* we design and implement a reactive runtime featuring a *preemptive* scheduling policy; enabling several (in)dependent task threads and applications with different priorities to run in an interleaved manner, responding to energy, time, and sensing events, and ensuring power failure resilience, memory consistency, and correct control flow.
3. *Performance Comparison and Reactive Applications:* we evaluate InK against state-of-the art systems and find our approach is up to 14 times more responsive to events in realistic intermittent power conditions. We develop, for the first time,

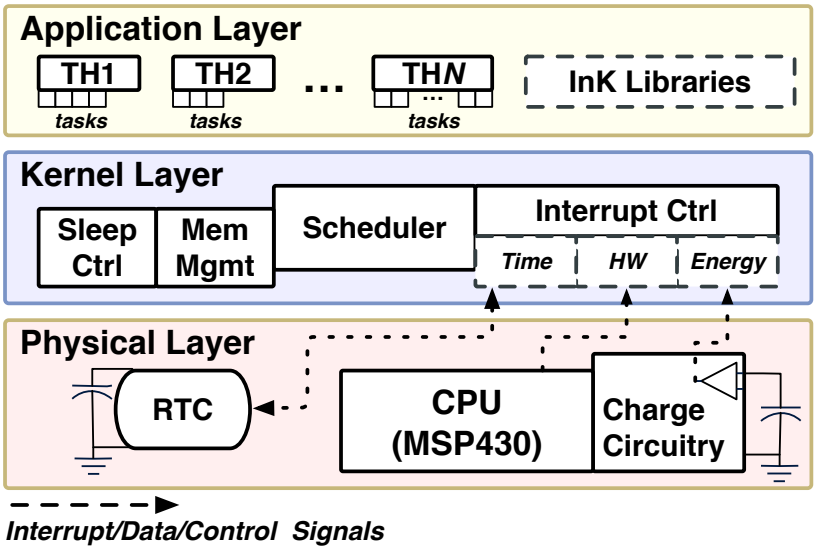


Figure 3.3: InK system overview; developers define task threads comprised of multiple tasks that are compiled with user and InK libraries in the application layer. The kernel layer interfaces with hardware to gather events, schedule task threads, manage time, and connect events to application level event handlers.

event-driven and reactive applications for battery-less sensors and tested them on different platforms like an intermittently-powered small battery-less robot. We also conduct a user study that demonstrates usefulness of InK.

4. *Open Source Release:* we release InK [129][1] as a open-source resource to the community; with example applications to increase the impact of this work and battery-less sensor networks.

3.2. INK: THE REACTIVE KERNEL

InK's kernel and programming model, discribed in this section, enable the development of timely event-driven applications for intermittently-powered battery-less sensors. InK simplifies or eliminates the event management and intermittent programming challenges described in Section 3.1 in three key ways:

1. **Event classification:** We classify three categories of events that are encountered in energy harvesting sensors that could lead to longer periods of failure, task starvation, and inability to precisely schedule or time execution of a task.
2. **Task threads Abstraction:** To respond to events, we introduce a novel concept, task threads, that have unique priority and encapsulate multiple restartable atomic tasks dedicated to a particular job.
3. **InK Kernel:** We design a scheduler and kernel that efficiently manages multiple task threads, handles forward progress of computation, ensures memory consis-

[1] We invite the reader to explore the source, documentation, and resources for InK: `https://github.com/TUDSSL/InK`.

tency, and keeps track of time. InK kernel provides a new programming model and language structures to develop event-driven battery-less programs including services for inter-task thread communication, event notification and handling, and timekeeping.

The **Intermittent Kernel (InK)** (shown in Figure 3.3) enables developers to write adaptive programs, reduces task starvation, and allows periodic sensing and timely response to externally generated.

3.2.1. INTERMITTENT COMPUTING EVENTS

Development of InK is motivated by the lack of proper event handling in current intermittent runtimes. Certain types of event can cause problems if not handled differently in the context of intermittent computing versus a traditional continuously powered (battery laden) device. Handling each of these events requires a new programming model and a more dynamic runtime operation beyond the static task-based models and the rigid and inflexible time focused models in the state-of-the-art. We integrate the handling of each of the following event types in InK.

Energy Thresholds: Energy harvesting battery-less devices only store small amounts of energy and expend it quickly. The amount of energy available for any period is not constant; it changes based on the time of day (for example in outdoor solar environments), the weather, and the location (if mobile). Static-task models are not robust to this energy irregularity; if a high energy radio broadcast is set to execute in a low energy situation, that task will never complete. Moreover, if a low energy reading on an accelerometer executes in a high energy situation, excess energy is wasted. Recent hardware designs like UFoP [130], Flicker [21], and Capybara [91] can capture this energy thresholding phenomenon, however current programming models do not associate tasks with their energy requirement, potentially exposing them to starvation.

Timers: Scheduling events in the future is difficult with intermittently powered devices because maintaining time through power failures is not trivial. When the microcontroller loses power, an external device powered by a small capacitor can support timekeeping until the microcontroller turns back on. A notion of time beyond timestamps and data expiration like in Mayfly [35] can provide useful scheduling mechanisms for periodic or single-shot tasks.

Hardware Interrupts: Nearly all sensing devices generate interrupts of some kind. Sensors like accelerometers, gyros, and magnetometers can gather data without any involvement from the microcontroller, storing this in a buffer and then alerting the microcontroller via an interrupt pin when the buffer is full. Analog sensors have thresholding circuitry that will wake a microcontroller when a point is reached. These hardware interrupts are not captured by current programming models, but are incredibly valuable to battery-powered sensors for extending battery life and will be valuable to battery-less sensors by increasing microcontroller responsiveness (by allowing the microcontroller to sleep).

3.2.2. INK DESIGN SPACE

Before proceeding with the design and implementation details of InK, we outline trade-offs in the design space for intermittent programming models and clarify some of our high-level design decisions.

Task-based versus Checkpointing. Two programming models dominate intermittent computing; task-based and checkpoint based (see Table 3.1). In our view, a task-based system provides language scaffolding that enables reactivity without high runtime cost or extensive static analysis. Tasks have traditionally been seen as a useful abstraction to implement scheduling and to enable concurrent applications. Moreover, task-based systems handle forward progress and memory consistency with less overhead. However, the task abstraction puts a burden on the programmer to decompose the program code into tasks (or task threads composed of several tasks, in the case of InK) and define the control flow. This also requires explicit data handing to ensure memory consistency which is seen in other task-based systems [33, 34]. An alternative method using automatic checkpoints can be imagined to implement a reactive system. This requires the programmer, compiler, or runtime to place checkpoints correctly while respecting memory consistency during event handling. Novel checkpoint placement policies to ensure a correct and consistent system execution would be required. Tasks, instead of checkpointing, lend themselves more naturally to scheduling unique threads of execution and provide scaffolding for dynamic execution to overcome starvation and ensure timely execution.

Dynamic versus Static Scheduling. In battery-less systems tasks should only start execution when sufficient energy is available. On the other hand, tasks that do not require much energy that are executed frequently will starve high energy tasks. For programming models with static tasks, this starvation possibility depends on how the programmer defines the task graph and the size of tasks. Once deployed this static schedule cannot react to changing energy conditions. A dynamic scheduling method can solve this at the expense of higher computational overhead. We take a dynamic approach with energy level-driven scheduling enabled by priorities that identify energy requirements as well as the criticality of the task. This introduces a higher programmer burden as priorities are decided by the programmer. The tension then becomes managing programmer burden, complexity of dynamic scheduling, and starvation. Our choice of scheduling algorithm matches the requirements of most applications with an implementation which introduces reasonable overhead. We provide details in following sections.

Preemption. Dynamic scheduling requires some concept of preemption to provide flexibility in the face of changing energy availability. The design trade-off comes from the coarseness of the preemption strategy. InK scheduler preempts task-threads on individual tasks' boundaries. This task level coarseness of preemption ensures reactivity with less switching overhead (if for example the level of preemption was at the instruction level) while maintaining task atomicity and avoiding concurrency errors. This comes with the price of less reactivity since the control flow is changed only after the execution of the active task is finished. Alternatively, the scheduler could preempt tasks at any point during their execution. However, this requires checkpointing that introduces extra memory and compute overhead as well as the possibility of memory inconsistencies.

InK Language Construct	Explanation
`__shared(...)`	Declares task-shared protected variables
`TASK(`**name**`)`	Declares an atomic task with given **name**
`ENTRY_TASK(`**name**`)`	Declares a task that will be the entry point of a task thread
`NEXT(`**name**`)`	Delivers control flow to the task with a given **name**
`__EVENT_DATA`	Holds the pointer to the event data in the event queue of a task thread that should be accessed by the entry task
`__EVENT_TIME`	Holds the timestamp of the current event in the event queue of a task thread
`__GET(`**x**`)`	Returns the value of the task-shared variable **X**
`__SET(`**x**`,`**val**`)`	Sets the value of the task-shared variable **X** to **val**
`_CREATE(`**priority**`,`**entry**`)`	Declares a task thread with a given entry task **entry** and **priority**
`_SIGNAL(`**priority**`)`	Activates the task thread with given **priority** within the context of a task thread
`_STOP(`**priority**`)`	Stops the task thread with given **priority** within the context of a task thread
`_interrupt(`**signame**`)`	Defines an interrupt handler with given service point **signame**
`__SIGNAL_EVENT(`**priority**`,`**event**`)`	Pushes event data **event** to the event queue of the task thread with given **priority** and activates it from ISR
`__CREATE_PIPE(`**src**`,`**dst**`,`**size**`)`	Creates a pipe structure of given **size** between task threads **src** and **dst** in order to share data between them
`__GET_PIPE_DATAPTR(`**src**`,`**dst**`)`	Returns the pointer to the data stored in the pipe between task threads **src** and **dst**
`__SET_PIPE_TIMESTAMP(`**src**`,`**dst**`,`**x**`)`	Sets the timestamp of the pipe between task threads **src** and **dst** to the given value **x**
`__GET_PIPE_TIMESTAMP(`**src**`,`**dst**`)`	Returns the timestamp of the pipe between task threads **src** and **dst**

Table 3.2: Summary of InK Language Constructs. The system API includes necessary calls for task and task thread declaration, memory consistency and control flow handling, event and interrupt management, as well as inter-task thread communication. ,

3.2.3. INK EXECUTION MODEL AND TASK THREADS

Taking into account these event types and design trade-offs we discuss the implementation of InK. InK handles the previously mentioned events by introducing *power failure proof task threads*. These task threads are the main building blocks of an InK program. A *task thread* responds to events and ISRs that triggers corresponding event-handling. An example implementation of `[TH2: Activity]` as described in Figure 3.2 is presented in Listing 3.1 and the corresponding execution and memory model is presented in Figure 3.4. A summary of InK language constructs are given in Table 3.2.

Task Threads: A *task thread* is a lightweight and stack-less thread-like structure with a single *entry* point that encapsulates zero or more successive *tasks*. These tasks can do computation, sensing, or other actions, are idempotent, atomic, and have access to shared memory. Each *task thread* has a unique *priority* and accomplishes a single objective, e.g. periodic sensing of accelerometer. In order to preserve the progress and timeliness of computation despite power failures, InK kernel keeps track of each task thread

Listing 3.1: Task thread code for TH2 and Energy ISR.

```
        // task-shared persistent variables.
        __shared(int data[10]; int i;);

        // the entry task of the thread
        ENTRY(Sense){
              // sample sensor
         int read = __sample_acc();
         // data[i] = read
         __SET(data[__GET(i)],read);
         ...
         NEXT(Features); // next task is sample
        }
    ...
        TASK(Classify){
              ...
         //write pipe
         __WRITE_PIPE(TH2,TH1,value);
         ...
         NEXT(null); //thread finishes
        }
    ...
        _interrupt(HIGH_energy)
        {
              ...
         event.data = dataptr; // data pointer
         event.size = datasize; // data size
         event.timestamp = __getTime();
         // post to TH1's event queue
         __SIGNAL_EVENT(TH1,&_event);
         ...
         /* turn on CPU */
         __bic_SR_register_on_exit(LPM3_bits);
        }
```

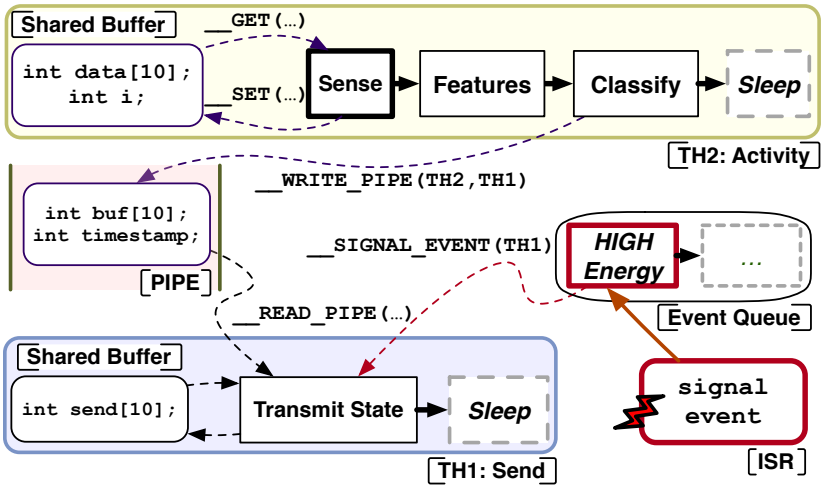


Figure 3.4: Overview of InK execution/memory model and task threads/ISRs interaction. Arrows indicate API calls of InK services.

by maintaining a *task thread control block* (TTCB) in non-volatile memory[2]. TTCB holds the *state* and the *priority* of the task thread, pointers to its entry task, to the next task in the control flow and to *buffers* in non-volatile memory that holds *task-shared variables*.

Task Thread Scheduling: The InK kernel implements *preemptive* and *static priority-based* scheduling of task threads: the InK scheduler always executes the next task in the control flow of the highest-priority task thread. Upon successful completion of this task, the pointer in the corresponding TTCB is updated so that it points to the next task in the control flow. In InK, tasks *run to completion* and can be preempted only by interrupts. Therefore, task thread preemption may only happen at tasks boundaries. When an ISR preempts the current task, it might activate other task threads of high-priority that are waiting for the corresponding event. Then, InK does not switch control to the higher priority task thread immediately; it waits for the atomic completion of the current task.

3.2.4. InK Memory Model

Tasks inside a task thread communicate with each other by manipulating *task-shared variables*. InK adheres to the data *encapsulation* principle by limiting the scope of these variables to the tasks of the corresponding task thread. Therefore task-shared variables are bound to the tasks that manipulate them and they are kept safe from misuse and interference by other task threads. InK allocates these variables in the non-volatile memory and they are double-buffered [28] to preserve data consistency across power losses—namely an *original* buffer holding the original copies and a *privatization* buffer holding the task-local copies [34].

Data Privatization: The TTCB of each task thread holds *pointers* to these buffers. Before running any task, InK initializes the privatization buffer by copying the contents from the original buffer. Tasks can read/modify only the content in the privatization buffer (via `__GET` and `__SET` interfaces). On successful task completion, the buffer pointers are swapped so that the outputs of the current task are committed *atomically*.

Inter-Thread Communication: InK facilitates inter-thread communication through *persistent pipes*. A pipe is a unidirectional buffer in non-volatile memory with a timestamp. Any task inside the producer task thread can write to a dedicated pipe so that any task in the consumer task thread can read and perform computation by considering the timeliness of the data. Since tasks cannot preempt each other and also pipes are unidirectional, pipe access does not lead to data races even upon power failures.

For the sake of efficiency and simplicity, we did not provide extra protection over the persistent pipes that enable data sharing among task thread—the consistency of these shared memory regions should be explicitly handled by the programmer. Alternatively, this protection could be handled by InK, however, this increases the implementation complexity and overhead of our system.

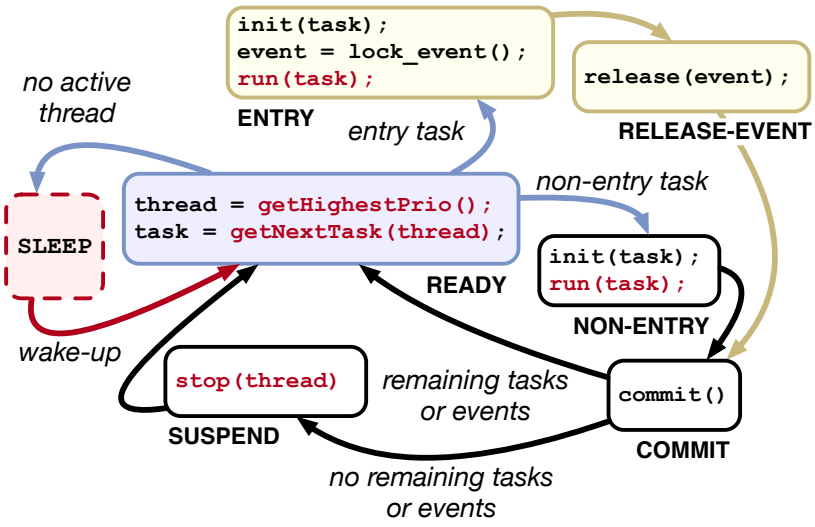


Figure 3.5: The InK scheduler state machine that selects the next task in the control flow of the thread of highest priority, ensures forward progress and puts the CPU in sleep mode when possible.

3.2.5. Reactive Execution

In order to ensure reactivity and adaptability, InK implements the state machine depicted in Figure 3.5 and maintains a *scheduler-state* variable in non-volatile memory in order to ensure forward progress despite power failures.

The Scheduler Loop: At each loop iteration, the scheduler selects the task thread of highest priority and executes the next task in the control flow of the selected thread. During task execution, the scheduler (i) initializes the task privatization buffer via `init`; (ii) for the entry tasks, it locks the event data that triggered thread execution via `lock_event` (to eliminate data races between ISRs and tasks—see following sections), (iii) it executes the task via `run`, (iv) for the entry tasks it releases the event via `release`, (v) it commits the tasks modifications by swapping buffer pointers, (vi) it suspends the thread if there are no dedicated events or remaining tasks. If there is no thread in a ready state, the scheduler puts the micro-controller into low-power mode, saving energy and waiting for an interrupt for activation. The state machine enables the progress of computation since it continues from the state it is interrupted.

Reducing Starvation: Tasks inside task threads and ISRs can activate and deactivate other task threads and change control flow *dynamically*. In existing run-times, e.g. Mayfly and Alpaca, control flow is static, in the sense that all subsequent tasks in the chain should wait for the completion of predecessor tasks. This leads to the problem of *priority inversion* since high-priority tasks can be blocked due to the lower-priority tasks

[2] Commercial off-the-shelf microcontrollers like the MSP430FRxxxx (a common microcontroller for intermittent computing) have volatile SRAM and non-volatile FRAM memory segments. The MSP430FR5969 has only 2KB of SRAM and 64KB of FRAM.

holding the CPU. On the contrary, since the InK scheduler alternates between the aforementioned states, it can switch execution to the high-priority thread: first, the kernel awaits the completion of the interrupted current task inside the lower priority thread; then it starts executing the entry task of the high-priority thread.

Responding to Events: Each task thread in InK has a dedicated non-volatile *event queue* that holds the events generated by ISRs. When any event is generated, the corresponding task thread is activated so that the thread execution will start from its *entry task*. In InK execution model, the event data is only accessible by the *entry task* of the task thread: the *entry task* locks the event data (see `lock_event` in Figure 3.5) to eliminate data races between ISRs and tasks. The entry task reads the event data and modifies necessary task-shared variables and then the event lock is released so that the event data will be removed from the event queue.

Event Handling: Circular buffers hold the data to be shared between an ISR and a task thread to prevent data races. The buffer handling introduces additional implementation and execution overhead but eliminates the need for the programmer to be involved in this process. As an alternative, unbounded buffers could be implemented, however management of a dynamically growing buffer introduces extra overhead at run-time for already memory constrained devices[2].

Interrupt Management: The *pre-processing* of an interrupt is performed by the corresponding ISR. Then, the rest of the computation is done by a task thread. When an interrupt is generated, the corresponding ISR delivers the received or generated data to the upper layers of the system and notifies task thread. Event queues are *ring buffers* dedicated for each task thread. They form an intermediate layer that prevents race conditions and preserves the event data consistency by eliminating ISRs from modifying task-shared data directly. When the event queue is full InK removes the event that has the oldest timestamp from the event-queue to increase the probability of having fresh data. Once an interrupt is generated, the task threads is notified by creating an *event* holding a pointer to the *ISR data* and its *size*, and a timestamp indicating the time at which interrupt is fired. The corresponding task thread is notified (via `__SIGNAL_EVENT`) by passing the pointer event structure so that the event will be placed in the event queue of the given task thread *atomically*.

3.2.6. SCHEDULING EVENTS AND TIMERS

InK builds a timer sub-system using an external *persistent timekeeper* [131] that keeps track of time across power failures: (i) when the microcontroller is running, its internal timers are used to measure elapsed time; (ii) upon a power failure, the external timekeeper keeps running and provides elapsed time until recovery. The timer system implements a *timer wheel algorithm* to provide two types of timers for the task threads: *expiration timers* and *one-shot/periodic timers*.

Expiration Timers: Task threads set expiration timers in order to enable timely execution of task threads and stop unnecessary and outdated computation if necessary; analogous to Mayfly [35] concepts of `expiration`. As an example, data read from a sensor should be processed within a time constraint and if computation exceeds the required

deadline the outputs of the computation are not useful anymore. When an expiration timer fires, the corresponding task thread is evicted so that it does not consume systems resources, e.g. CPU, anymore.

One-Shot/Periodic Timers: One-shot and periodic timers are used in order to schedule events in the future and generate periodic events, e.g. activating task thread at a given frequency. Since most of the sensing applications are periodic, these timers are the foundations of task threads that perform periodic sensing, these timers build on the *persistent timekeeper* to keep time across power failures.

3.3. EVALUATION OF INK

We proceed with the experimental evaluation of InK. We compare InK against its counterparts by implementing sensing applications that require timely response to various events. In our evaluation we measure several metrics to observe the reactiveness as well as overhead in terms of time, energy and system resources. Considering these metrics, we show that InK improves the reactivity of battery-less sensing applications **up to 14 times** as compared to its counterparts, by introducing a reasonable system overhead. Finally, our case studies show that InK enables new, never before seen sensing applications. We aim to help developers in learning and contributing to InK by providing resources to the community with a dedicated website [129].

3.3.1. EXPERIMENTAL SETUP

We describe the experimental setup used in assessing the performance of InK against existing state-of-the-art runtimes and as a stand-alone system. Our setup considers replicability of results and varying types of energy supply.

Target Embedded Platform: The experiments were conducted using TI MSP-EXPFR5969 evaluation boards [132]. This platform uses 16 MHz MSP430FR5969 MCU with 64 kB and 2 kB of non-volatile (FRAM) and volatile memory (SRAM), respectively. We set the microcontroller frequency to 1 MHz during our experiments. Whenever necessary, InK sensing system interacted with low-power accelerometer [133], microphone [134] and infrared transmitter (Vishay Semiconductor TSOP38238)/receiver (generic 950 nm infrared LED) pair.

Runtimes for Intermittently-Powered Devices: InK was compared against two state-of-the-art runtimes: MayFly [35] and Alpaca [34]. For each runtime, we have prepared the same application introduced in the subsequent sections and composed of the same set of tasks and control flow.

Measurement Equipment: We used the Saleae logic analyzer [135] to measure the performance metrics of all applications that were implemented during experiments. Data was parsed with dedicated, on-line accessible [129], Python scripts.

Intermittent Power Supply: We used two setups to provide repeatable experimentation: a real wireless power supply (used in InK case studies) and emulated power (for repeatability and replicability of comparative measurements). *Real wireless power supply*: To power MSP430 evaluation boards, we used Powercast [72] TX91501-3W trans-

mitter emitting RF signal at 915 MHz center frequency to P2110-EVB receiver [72] (one per each MSP430 board) co-supplied 6.1 dBi patch antenna. *Controlled-power supply:* we considered two approaches per different experiments. Approach (1) used the Ekho platform [136] that replays a realistic and repeatable (recorded from real harvesters) I-V surface to the MSP430 evaluation board. Approach (2) is based on a dedicated MSP430 evaluation board that interrupts the power supply of the other board by controlling the RST pin [22, Sec. 5.12.2], with a uniformly distributed interrupt period in the interval of [0, 0.5] seconds.

3.3.2. REACTIVE APPLICATION PERFORMANCE

We start by demonstrating the main strength of InK: by fastest reactivity of programs for transiently-powered devices.

Implementation: We implement a *battery-less condition monitoring* application in InK. Two threads are considered: one (first priority) that detects whether a new event happens (like the arrival of a new item to a CNC router), and a (second priority) event which detects the condition of a device (like specific vibration of a machine). The first thread is triggered by a sound overthreshold and implements an FFT (analyzing data from a microphone), while the second samples and records data from the accelerometer periodically. This application was also implemented using MayFly [35] and Alpaca [34] runtimes for comparison. Since Mayfly and Alpaca are not event-driven and they do not allow interaction with interrupts, the only way to implement this application was to use a control flow that implements a *polling* loop: the microphone is checked to catch the sound overthreshold and perform FFT if required; and then reading the accelerometer and performing sensor data related computations continuously. Since Alpaca has no notion of time (contrary to InK and MayFly) we could not consider timeouts. Also, Alpaca makes it impossible to use external libraries, e.g. accelerated FFT library for TI MSP430 MCU. Therefore, in the comparison, we mimic the FFT operation by a constant delay loop.

For the replicability of power failures from energy harvesting, an Ekho [136] emulator powered the target embedded platform. Ekho repeats, in an infinite loop, a pre-recorded one minute (i.e. a maximum length Ekho can support) I-V curve recorded from one of two energy harvesting sources: (i) 22×7 mm IXYS Solarbit solar panel [137] which was relocated from indoor to cloudy outdoor sunlight—representing the trace with long periods of energy availability; and (ii) a WISP harvester powered by Impinj Speedway 420 RFID reader [78] with readers' antenna at initial 25 cm distance from WISP was relocated to 10 cm and again to 25 cm—representing trace with very high power intermittency rate. Data from the condition monitoring application was collected for five minutes of continuous operation, for five runs with each runtime.

Metrics: We have measured the following reactivity metrics for each runtime: *Success Rate*—the rate of successfully executed highest priority events; *Missed event rate*—how many high priority events are missed due to either power failures or on-going low-priority computation; *Maximum 'Power On' Time*—the longest duration that the device was alive (at intermittent power); and *Death Rate*—the number of power failures (at intermittent power) during the experiment.

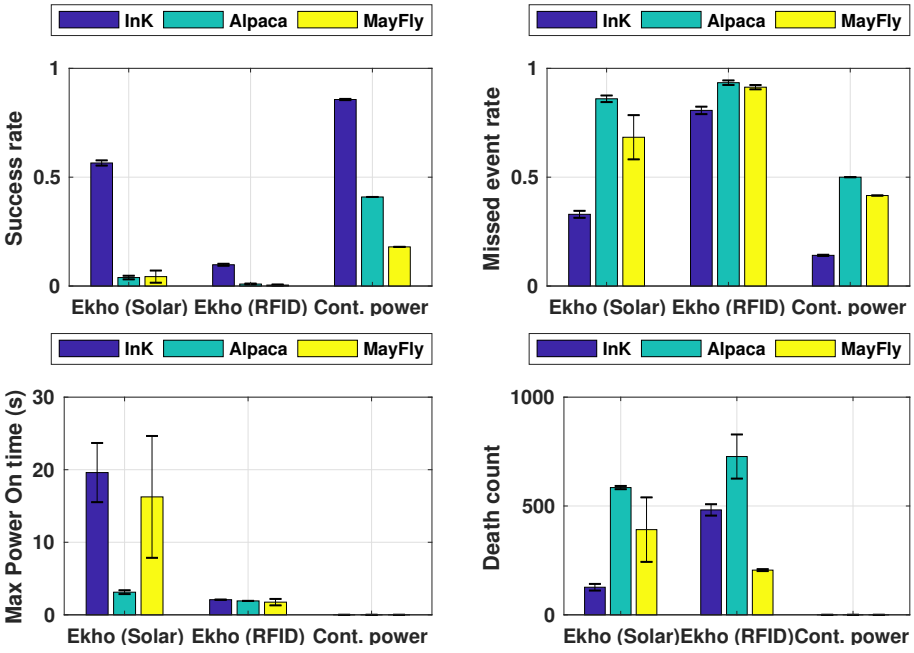


Figure 3.6: Performance metrics of reactive sensing application, implemented using InK, MayFly and Alpaca. Performance of all runtimes is compared to the continuous power case. InK improves the reactive application performance by orders of magnitude for *all* metrics.

Results: The result is presented in Figure 3.6. We observe that InK is the most reactive runtime of all, **improving over the success rate of Alpaca and Mayfly by 14 and 13 times**, respectively (for the solar power case). Naturally, InK cannot obtain a perfect success rate, simply because of death during the interrupt arrival. Also, InK misses the least number of priority threads (Figure 3.6 (top, right)) compared to other runtimes. Mayfly is less reactive compared to Alpaca, as it must check the timing constraints of every single task in between actual task execution; for sophisticated programs with more than ten tasks, this reduces the percentage of time Mayfly can poll for events. We note that Mayfly died fewer times in our experiments, due to implementation differences in the systems startup code where Mayfly stores energy to enable timekeeping, however, we note that this reduced death count did not increase reactivity as Mayfly was unavailable for computing during this startup period. InK had the highest effective 'power on' time and the lowest death count among all runtimes: Figure 3.6 (bottom left and right).

3.3.3. REAL-WORLD EVENT-DRIVEN APPLICATIONS

We demonstrate several never before seen battery-less applications enabled with the event-driven programming supported by the InK kernel. The development of these applications is not feasible with existing runtimes like Alpaca or Mayfly due to the challenges C1–C4 listed in Section 3.1.

Power	Power Failures	Motion over Threshold	Catch Rate	IR Trans.
Continous	0	8	1	6
RF	36	6	0.66	1

Table 3.3: The response to the activity event of the battery-less voice-controlled activity recognition application.

BATTERY-LESS EVENT-DRIVEN SENSING

We start with the first case study: voice-controlled battery-less activity recognition.

Application Challenges: We used a low-power accelerometer [133] for the classification of activity and a microphone [134] for voice recognition connected to TI MSP430 board. Initially, the system is sleeping and a voice recognition task thread is waiting to respond to the accelerometer interrupt to detect motion over a threshold (*requirement for C1 and C3*). When this event is detected, the voice recognition task thread starts responding to the interrupts generated by the microphone and recognizes the 'start' command (*requirement for C1 and C3*). After the start command is detected, the activity recognition task thread is activated to periodically sense and classify ongoing activities (*requirement for C2*). When the available energy is over 2.5 V the comparator COMP_E of MSP430fr5969 is programmed to generate an interrupt. When this energy threshold interrupt is generated (*requirement for C4*), the control is switched to task thread that sends the classification results using an infrared transmitter via simple OOK modulation.

Results: We collected the success rate of activity recognition for five minutes. The system is powered continuously (to use as a reference for comparison) and by using the Powercast transmitter. Table 3.3 shows our measurement results. With continuous power, 8 motion threshold interrupts were detected and all of them were processed on time. Among them, energy levels allowed to perform 6 IR transmissions after activity classification. With RF power, power failures were observed 36 times, 6 motions above thresholds were detected and almost 4 of them processed on time. Energy levels allowed to perform only 1 IR transmission in this case.

BATTERY-LESS INTERMITTENT ACTUATION

We continue with the second case study: a tiny battery-less robot designed to perform autonomous reconnaissance sensing tasks. Using this robot, we demonstrate that InK enables reactive control of battery-less energy-harvesting actuators.

Battery-less Robot Motivation: Referring to Table 3.4, state-of-the-art robotic platforms are battery-dependent and require physical/proximity contact to recharge. Our idea is to remove these obstacles by providing power *directly* from the harvesting source (solar panel) to the environmentally-friendly storage (super-capacitor). The consequence is the intermittent movement of a robot. Movement stops after short move duration (in the order of seconds).

Robot Design: Figure 3.7 provides our robot design overview. The robot is designed around a WISP 5 [20] which allows the observer to send control messages from the RFID reader. WISP's TI MSP430FR5969 MCU serves as the main robot control system. DC

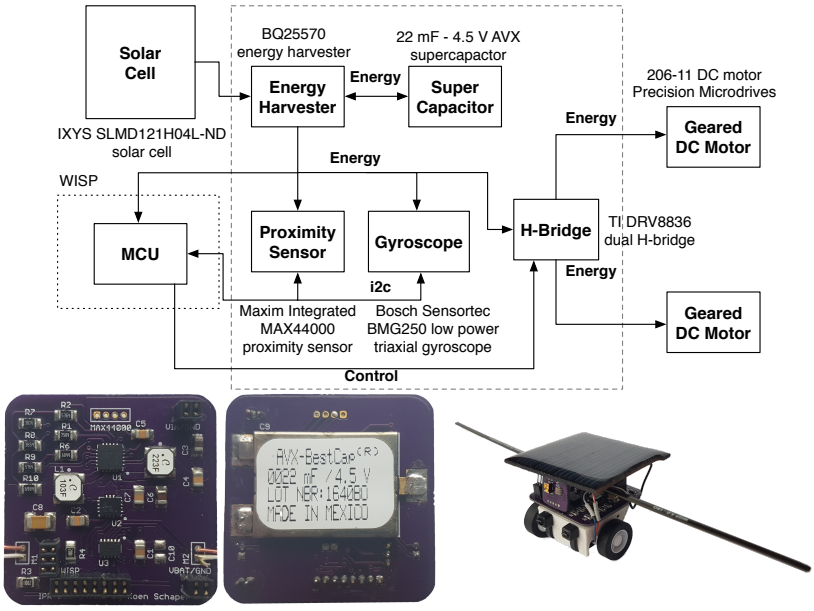


Figure 3.7: Battery-less small autonomous robot: top figure—block diagram; bottom left—robot PCB design (front side): harvester (U1), gyroscope (U2) and H-bridge (U3); bottom center–robot PCB design (back side): super-capacitor), bottom right—complete robot.

Design	Motion	Speed (cm/s)	Size (mm)	Weight (g)	Storage (mAh)	Time to recharge	Recharge method
Roverables [138]	wheel	N/A	40×26	36	100	45 min	inductive
Zooids [139]	wheel	50	26×26	12	100	1 h	manual
mROBerTO [140]	shaft	15	16×16	10	120	1.5 h	manual
GRITSBot [141]	wheel	25	31×30	60	150	1 h	contact
Kilobot [142]	vibration	1	33×33	17.6	160	3 h	manual
HAMR-VP [143]	legged	44	44×44	2.3	8	3 min	manual
This robot	wheel	25	35×40	22	0.006	<5 s	solar

Table 3.4: Comparison of our battery-less harvesting robot against state-of-the-art small robotic platforms.

motors were used as actuators. Two motors are mounted diagonally opposite from each other in a 3D-printed frame. Small plastic wheels with rubber tires are mounted directly on each of the motor shafts. Behind the motors, a free-running caster wheel is mounted to the frame. PWM controls the robot's speed and is used to reduce the average current consumed by the motor. A large bulk capacitor supplies short high current demand from the motors.[3]

[3] In-depth information about the robot hardware and software and InK implementation of robot control algorithm is provided in [144, 145] and InK repository [129], respectively.

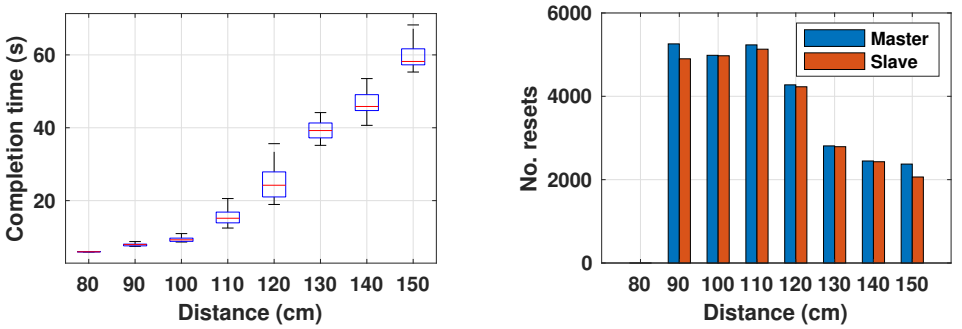


Figure 3.8: Intermittent communication experiment result: two MSP430 boards are connected together using UART RX/TX ports exchanging messages in an infinite loop, also connected to Powercast receiver boards and powered by the RF. Left figure: completion time, right figure: the number of resets at different distances to the transmitter.

Robot Control: We implemented a simple PID controller feedback loop to drive the robot. Our transiently-powered robot can make one movement, which requires multiple power cycles to complete. To finish the required move upon power failures and to capture the progress towards the movement target, several InK services are used. The PID control loop is implemented as a task thread that is scheduled to execute every time a robot is powered. At each periodic activation, the thread samples the yaw-rate using the gyroscope and executes the tasks of the control algorithm to update the motor parameters.

BATTERY-LESS INTERMITTENT COMMUNICATION

We conclude with the final case study. We demonstrate that InK enables a basic distributed processing system via communication over the serial port.

Battery-less Communication Challenges: Two master and slave TI MSP430 boards are powered using independent energy harvesting sources; each board was connected to a Powercast receiver and placed at several distances away from the corresponding RF Powercast transmitter. The master board implemented two task threads (*requirement for C1*): the main thread generates a random signal composed of 16 floating-point values, transmits the signal to the slave board and computes the DFT of this signal; meanwhile another thread waits UART RX events (*requirement for C1 and C3*), receives the DFT result of the slave node composed of 16 floating-point values and pushes the result to the pipe of the main thread and activates it (*requirement for C1*); then the main thread compares its results with the one in the pipe and toggles an output port if they are equal. The slave board implemented one task thread that waits for UART RX interrupt to receive the random signal from the master node (*requirement for C1 and C3*), computes the DFT of the received signal, sends the result to the master node. In order to keep track of the delivered packets, the sender side sets a one-shot timer (*requirement for C2 and C3*) and awaits acknowledgment (ACK) from the receiver side. If ACK is not received, the packet is re-transmitted.

	InK			Alpaca			MayFly		
	Time (ms)	Memory (B)		Time (ms)	Memory (B)		Time (ms)	Memory (B)	
		.text	*.data*		*.text*	*.data*		*.text*	*.data*
AR	4151	3442	4459	8361	7970	724	4464	8700	2496
BC	546	2922	4433	912	6290	818	1019	8066	846
CF	495	2648	4693	199	8494	2352	—	—	—

Table 3.5: Execution time and memory consumption for three benchmark applications written in InK, Alpaca and MayFly. Since it was not feasible with Mayfly to develop *CF* application, its corresponding values are shown with —. Overall results show that InK's overhead is comparable with its counterparts.

Results: We monitored the output ports of the boards for 15 minutes per each Powercast transmitter/receiver distance. Figure 3.8 present our measurement results. During the execution of the application, not only the computation but also the communication is interrupted frequently, especially at distances 80–120 cm—since time to charge is longer at further distances, this led to the longer duty-cycles, less power failures and number of completions. We observed that the applications on both master and slave nodes are always completed successfully despite frequent power losses.

3.3.4. INK SYSTEM OVERHEAD

We continue with assessing the overhead of InK by implementing common computation-based benchmarking applications and comparing their execution time, code size and memory requirements.

Implementation: The complete suite of benchmarking is composed of: (a) *Activity Recognition (AR)*: machine-learning enabled physical activity classification using locally generated accelerometer data, (b) *Bitcount (BC)*: bit counting in a random string based on sever different methods, cross-verifying correctness, and (c) *Cuckoo Filtering (CF)*: runs a cuckoo filter over a set of pseudo-random numbers and performs the sequence recovery using the same filter. We implemented these applications in InK, Alpaca and Mayfly using the same task partitioning and control flow. Unfortunately, loops are not allowed in a Mayfly task graph as the data and control flow are the same (leading to infinite data growth). Therefore, non-sensing applications like *CF* cannot be implemented in Mayfly because of the multi-level loops and control flow disassociation from data flow.

Results: For the fairness of the comparison, we run the aforementioned benchmarking applications on continuous power. To measure the execution time, we sampled the output port of the MCU using the logic analyzer that is toggled after the program completed its execution successfully and the results are correct.

Table 3.5 presents the summary of evaluation. We conclude that InK is always faster than MayFly and only slower than Alpaca for *CF*. Additionally, we measured the memory overhead and code size for all runtimes. The increased memory and execution overhead in selected applications are due to the fact that InK maintains queues and data structures in order to manage/schedule task threads and makes scheduling decisions at runtime for

Operations	≈Overhead (in μsec)
Initial Boot	7900
Reboot	70
Scheduling and Selecting Next Thread	89
Task Init (10 B/1 KB shared data, resp.)	121/315
Task Commit	42
Activating Thread	75
Event Register	778

Table 3.6: Approximate overhead of the initialization and scheduler overhead.

the sake of reactivity. Our results show that InK enables an event-driven paradigm shift for battery-less devices while introducing a reasonable overhead as compared to Alpaca and Mayfly.

Point to Point Overheads: Table 3.6 presents detailed overhead of the InK runtime operations. When InK runs for the first time, all of the internal non-volatile state variables are initialized (denoted as the *Initial Boot* overhead). After the first boot, each *Reboot* requires only the initialization procedures of the MCU, peripherals and the recovery operations of the InK scheduler. The *Scheduling* overhead is introduced to select the thread of highest priority and execute the next task. *Task Init* and *Task Commit* overheads are introduced to prepare the privatization buffer and commit the modifications on this buffer to the original buffer atomically, respectively. The time spent for *Activating Thread* is required to change the state of the corresponding task thread to ready so that the scheduler will consider to run it later. *Event Register* is the overhead of committing an event in the event queue.

3.3.5. USER STUDY

We have performed an on-line user study to assess the usability of InK in programming intermittently-powered devices. The study is approved by the Human Research Ethics Committee of Delft University of Technology. The study suggests that (i) InK is *intuitive* and applicable to a varying set of sensing applications, and (ii) InK provides the *necessary constructs to write periodic sensing applications*.

Methodology: Participants were provided a link to the on-line survey (questionnaire and detailed answers in [129]) via a personal invitation. The survey was accompanied by a short document introducing the concept of intermittently-powered devices and the issues associated with programming such devices. Then, participants assessed three programs implementing non-ISR `[sense]` → `[compute]` → `[transmit]` loop written in InK, Aplaca and MayFly languages, after which a set of questions were asked. Additionally, participants had to assess the same program written only in InK, but implemented using interrupt service routines. Finally, the participants were asked to write a simple InK program themselves (submitted to us for inspection) and again assess InK usability. An answer to each question was one from a five-level Likert-type scale (From *Strongly disagree* to *Strongly agree*). There was no time limit on the assignment and the survey

could have been performed at the most convenient location and time for each participant.

User Pool: We have collected 22 responses in total from a pool of graduate (MSc/PhD level) students studying embedded systems, computer science and experienced scientific developers. 63% of the participants had more than five years of formal computing education. 82% of the participants had more than five years of programming experience. C was considered the language of choice for the majority of participants, while 68% of them considered their knowledge of C as average and above average. Also, 42% self-assessed themselves with above-average knowledge of embedded systems (compared to others with a similar background, age and education). 73% of them considered their knowledge of intermittently powered embedded devices as low and very low. Participants were located in at least four different countries in Europe and North America.

Results and Discussion: Participants assessed the ease and intuitiveness of using all three languages in writing generic applications for intermittent devices: Alpaca being the easiest (18 strongly agreed or agreed), followed by InK (13 respective answers) and MayFly (only 8 respective answers). The same order applied to questions on the programming model flexibility. All thee codes were assessed equally in terms of programming mode completeness. All of them were assessed as the language that could be used for a variety of sensing applications(InK being the most selected one for this task).

Considering the task of assessing the reactive programming difficulty with InK, all responders provided their example InK code. 59% of the participants agreed that it was easy to understand how InK handled interrupts for intermittently-powered systems, while 57% agreed that it would be harder to understand if the code was written in plain C. 76% of respondents strongly agreed or agreed that InK provides the necessary constructs to write periodic sensing applications, with only 38% and 30% respondents for Alpaca and Mayfly, respectively. Only 23% agreed that it was hard to write InK program.

We acknowledge that our study is limited due to the small sample size and difficulty in preparing the comparable program in three programming languages. Nonetheless, survey results indicate that InK is the right tool for reactive programming of intermittently-powered devices.

3.4. DISCUSSION AND FUTURE WORK

InK application developer effort: An InK programmer does not need to reason about power failures or memory inconsistency—which confuse and frustrate even experienced developers—but needs to follow a new programming model that is different from existing ones targeted for continuously-powered systems. In particular, the programmer needs to (i) identify task-shared variables; (ii) provide a task-division and annotate task boundaries/inter-task dependencies; and (iii) define an explicit control flow. Future work can address removing this burden from the programmer: for example with a guided compilation tool that translates programs into InK, or with additions of features like module reuse.

Limits of reactivity: Although InK's goal is to enable reactivity for systems powered intermittently, it will never be able to provide the same reactivity as battery-powered or

tethered devices. Simply, with no control over available power there is no feasible way to make a sensing system reacting immediately to stimuli. The question remains how big is the set of applications that can accept reduction of system responsiveness, while not compromising the quality of service. Using our transiently-powered robot as an example: what is the acceptable number of stops (and their duration) in-between consecutive moves that would make robot still considered *reactive*? Future work could investigate networked approaches to this problem, with higher density of cheap battery-less sensing devices, overall coverage increases. Multiple challenges in networking and synchronization must be resolved to make this a reality.

Dealing with peripheral I/O: Sensing systems must manage peripherals (sensors, memories, radio) that have a volatile state. On power failure, this state is lost and peripherals must be reinitialized. InK leaves this as a programmer burden. A common solution is to split input operations into two tasks: one task reads the sensor value and another task consumes it accordingly. This guarantees consuming the value once since tasks run in order and cannot preempt each other: the consumer task can be re-executed safely since its output is produced by the former task. However, to re-execute output operations safely at intermittent power, e.g. blink an LED exactly once, hardware assistance is required. Future work could leverage emerging non-volatile sensors, or build software models for handling of failure in peripherals.

Starvation, fairness, multi-tenancy: task threads shows the potential for multi-tenancy on battery-less, energy harvesting devices. However, multiple issues surround the practical use case where third party applications coexist peacefully on a single intermittently powered device.

3.5. CONCLUSIONS

We have shown that state-of-the-art programming and execution models for intermittent systems are inadequate for developing real-world sensing applications: they do not respond to events in a timely manner, they do not schedule events to perform periodic sensing, and they do not handle interrupts while preserving memory consistency. To address these limitations, we introduced InK: the first reactive task thread scheduling kernel that facilitates event-driven applications for intermittently-powered systems. We evaluated InK based on software benchmarks and compared its performance against task-based runtimes (i.e., Mayfly and Alpaca) using real hardware and real energy harvesting traces. Our results showed that InK significantly improves the reactivity of battery-less sensing applications by up to 14 times, introducing a reasonable overhead. A significant portion of this overhead is attributed to InK's inability to optimize the task size according to the available energy. The ambient energy level can change significantly. When it rises, it creates opportunities for intermittent systems to execute more instructions before suspending application execution for computation-progress protection. Moreover, tasks energy demand is not fixed, when it is low the system has the chance to execute more tasks per power cycle. Prior task-based systems including InK are oblivious to these opportunities. Next, we investigate what does it take for an intermittent system to optimize its task size without hardware support.

4

DYNAMIC TASK-BASED INTERMITTENT EXECUTION

4.1. INTRODUCTION

A task-based programming and execution model requires a programmer [33, 34] or a compiler [90] to statically decompose a program into a collection of tasks. *Tasks*, top-level functions, can include arbitrary computation that should be executed despite power failures. At each task transition, a task-based system incurs an overhead to track and atomically commit modifications to non-volatile memory, to maintain consistency of program state [33, 34]. An obvious objective of such systems is thus reducing task transitions overhead. For that, a programmer may create very large tasks. A large task, however, may require more energy to complete than what a device's fixed hardware energy buffer can hold, which may lead to a task *non-termination* problem (Figure 4.1, right): the system repeatedly tries to execute a task but fails to finish it due to power failures. Therefore, statically optimizing task size faces the following *dilemma*: should large tasks be used, that are efficient but risk non-termination, or small tasks that are guaranteed to complete but incur a high task transition and commit overhead? Additionally, energy-harvesting battery-less devices have access to a varying power source (ambient energy). And, the energy demand of a task is not fixed. These variations in the required/available energy budget make opportunities for intermittent systems to execute more tasks per power cycle. However, static task-based systems are oblivious to energy conditions and consequently to such opportunities.

Challenges and Contributions. We introduce Coala: a new task-based system that employs *adaptive task size execution by task coalescing and splitting*. By means of this novel technique, small tasks can be executed efficiently by trimming unnecessary overhead *dynamically*, meanwhile avoiding the risk of non-termination. Coala accepts any static decomposition and it coalesces (groups) tasks or splits them (see again Figure 4.1)

Parts of this chapter have been published in ACM TOSN [74].

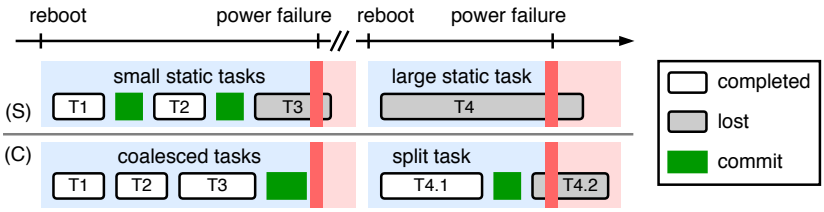


Figure 4.1: Task coalescing reduces commit overhead (left), while task splitting enables termination of large tasks (right). S: static legacy task-based runtime, C: Coala (this work).

based on the estimated energy availability without demanding any hardware support. To the best of our knowledge, Coala is the only system that eliminates restructuring and re-compilation of applications considering a device's energy buffer—enabling energy-storage independency for the applications, while keeping execution efficient.

The unique contributions of Coala in relation to challenges of task-based systems revealed by this work are listed below.

C1 *Overcoming Task Transition Overhead:* given unpredictable incoming energy, how to save computation state at task transitions as rare as possible? Coala tries to minimize task transition overhead by estimating energy conditions at run-time using *recent execution history* as a metric.

C2 *Dynamic Memory Consistency Handling:* merging static tasks on the fly rises the need for dynamic memory consistency handling. This leads to the second challenge: how to dynamically detect inter-coalesced-task data dependencies and ensure efficient protection against power interrupts? Coala addresses this challenge by relying on its novel *Virtual Memory Manager* (VMM). The VMM performs real-time dependency tracking to enable protection on a task transition. Individual variables tracking, however, slows down a system dramatically. Therefore, the VMM keeps memory consistent through *privatizing pages* and optimizes bulk data transfer through Direct Memory Access (DMA).

C3 *Ensuring Task Termination:* a static task decomposition model assumes that each task can execute to completion. If the hardware energy buffer provides inadequate energy to execute each task to completion, a program will not terminate [38]. This leads to a third challenge: how to enable the dynamic execution model to progress on a sub-task level? To avoid non-termination under adverse energy conditions, Coala uses a timer-based *partial task commit* mechanism. Partial execution avoids non-termination by committing the intermediate state of a long-running task that has repeatedly failed and restarted.

To asses the benefits of Coala over existing task-based systems, we implemented and tested six benchmarks on a real energy-harvesting platform. Our evaluation shows that Coala reduces run time overhead by up to 54% and solves task non-termination problem where existing static task-based systems fail.

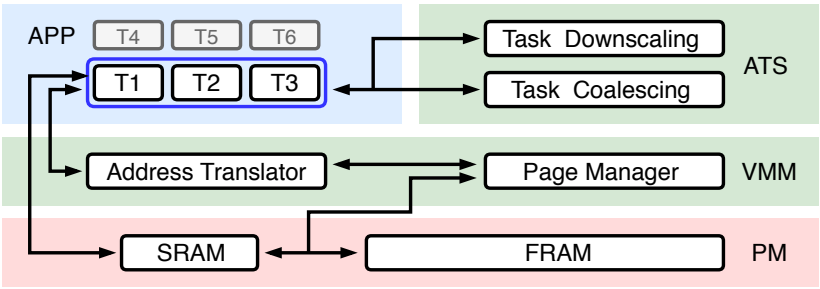


Figure 4.2: Coala top-level view. APP: application, ATS: adaptive task scheduler, VMM: Virtual Memory Manager, PM: physical memory.

4.2. COALA SYSTEM OVERVIEW

Coala is a new programming and execution model that supports adaptive task-based execution and eliminates restructuring and re-compilation of applications considering the device's energy buffer. Coala addresses the challenges given in Section 4.1 by making task-based intermittent applications portable in the sense of different energy storage sizes while keeping their execution efficient. Fig. 4.2 shows an overview of Coala.

Programming and Execution Model. To use Coala, a programmer must (i) convert a plain C code into tasks by encapsulating the code in a top-level set of functions, (ii) sequence the control-flow between these tasks, and (iii) annotate memory accesses that manipulate task-shared data. Then, they compile and link the code against Coala's runtime, producing a Coala-enabled binary. The runtime relies on Coala's novel *adaptive task scheduler* to adapt its execution to the energy conditions. Facilitating efficient task adaptation requires dynamic memory protection which Coala's *virtual memory manager* handles through *page privatization*.

Adaptive Task Scheduler. Coala's adaptive task scheduler (ATS) makes *energy-aware* scheduling decisions to group tasks together or split a task. By coalescing tasks Coala *amortizes commit and transition costs*, and by splitting a task, after it repeatedly failed to complete, it *avoids the task non-termination problem*. The scheduler uses its recent execution history—i.e. it is hardware independent—as a metric to estimate energy availability and eventually to decide on the coalesced task size. Section 4.3 describes ATS.

Virtual Memory Manager. Coala virtual memory manager (VMM) is the key enabler to ensure memory consistency while coalescing tasks. VMM allows applications to interface with only fast volatile memory pages and privatizes the pages demanded by a coalesced task in order to solve a novel data consistency problem; namely *task coalescing-induced WAR dependencies*. VMM achieves page privatization by keeping all page modifications in non-volatile memory on a coalesced task transition. Section 4.4 explains VMM.

Algorithm 1 Coalescing

```
1:  C ← f_REBOOT(z)                          ▷ Update coalescing target by reboot update function
2:  H ← 0                                    ▷ Initialize history
3:  while true do
4:      j ← 0
5:      while j ≤ C do
6:          EXECUTE_TASK(T_i)                ▷ T_i is the ith task executed since the last power failure
7:          W ← f_WEIGHT(T_i)                ▷ Assign new task-dependent weight
8:          j ← j + W
9:          H ← H + W
10:     COMMIT_TO_FRAM()
11:     C ← f_COMMIT(C)                      ▷ Update coalescing target by commit update function
```


4.3. COALA TASK ADAPTATION

Coala's design includes a novel task scheduler. It uses efficient, energy-aware task coalescing to amortize static task overheads, and a timer-based task-splitting mechanism to avoid non-termination of tasks too long for a device's energy buffer.

4.3.1. TASK COALESCING

When a device's buffered energy is sufficient to run multiple tasks without a power failure, committing state after each task is unnecessary overhead. Coala reduces this overhead by coalescing a sequence of tasks and deferring commit operations for all tasks to the end of the sequence. In general, committing involves moving state manipulated by a task into its permanent location in non-volatile memory. In particular, Coala's commit procedure copies dirty pages of memory that a task updated from fast, volatile working memory to slower, non-volatile main memory. We defer the details of paging privatization and commit to Section 4.4. An effective coalescing strategy must be *aggressive* enough, attempting to coalesce a large number of tasks to amortize commit overhead. However, it must also be *conservative* enough, coalescing only as many tasks as will execute to completion given certain energy conditions, reducing the risk of re-execution penalty for long coalesced tasks.

GENERIC DESIGN OF TASK COALESCING STRATEGIES

Algorithm 1 shows the general structure of a coalescing strategy. In the algorithm, C (coalescing target) is total number of static tasks that Coala will next attempt to coalesce. T_i is the ith task executed since the last power failure. H (history) is the total number of tasks executed since the last power failure. W_i is the *weight* of a task. A task's weight is an arbitrary quantity associated with the task that represents its cost in time or energy to execute. Different coalescing strategies may apply different weights to a task, e.g., $W_i = 1\ \forall i$, or $W_i = \alpha E(T_i)$ where $E(T_i)$ is the average execution time of T_i and α is a constant.

TASK COALESCING STRATEGIES

Different coalescing strategies adhere mainly to the template in Algorithm 1, varying in only a few *characteristic operations* that the algorithm leaves deliberately abstract. The reboot update function, f_{reboot}, updates C, the coalescing target, after a reboot. The

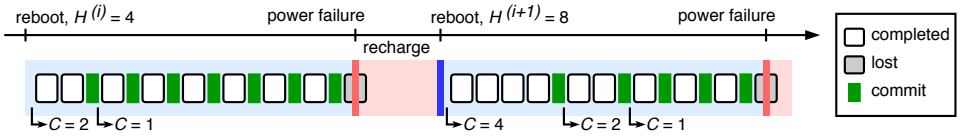


Figure 4.3: EG coalescing sample execution across two power cycles. C: coalescing target, H: history.

weight lookup function f_{weight} returns a task's weight. The commit update function f_{commit} updates C after a successful commit. The following paragraphs detail three coalescing strategies that we developed for Coala, albeit not the only possible. In fact, Coala *allows programmers to design their own strategy* to implement and link against Coala's task scheduler.

4

Energy-Oblivious Coalescing. *Energy-Oblivious* (EO) coalescing strategy treats all tasks as having unity weight and varies the size of the coalescing target linearly. In such a scheme, the number of tasks to coalesce, C, increases by a constant x when a coalesced sequence of tasks commits and decreases by the same constant x when a coalesced sequence of tasks fails to complete, i.e., after a power failure. The characteristic operations of EO are

$$\begin{aligned} f_{\text{reboot}}(C) &= C - x, \\ f_{\text{weight}}(T_{\text{i}}) &= 1, \\ f_{\text{commit}}(C) &= C + x. \end{aligned} \tag{4.1}$$

EO reacts slowly to the variation in the energy required to execute different tasks and to the variation in the effective quantum of energy available to the device. With the successful commit of each coalesced task sequence, C increases. Eventually, the target may be too high and only a coalesced task composed of a few units will commit without interruption by a power failure. The strategy then linearly decreases C, eventually reaching a value that allows completion. A key limitation of this algorithm lies in the equality of the target decrease in f_{reboot} and the increase in f_{commit}. Let us assume $x = 1$. If, after k successful commits, the target must decrease to its original value $C - k$ due to an energy drop, EO requires k successive reboots to progress.

Energy-Guided Coalescing. The *Energy-Guided* (EG) coalescing strategy adapts its coalescing target more quickly, to adhere to changes in energy conditions, addressing a key limitation of the EO. It uses its recent execution history, H, to estimate energy availability and alters its target accordingly. The EG algorithm is characterized by the following functions

$$\begin{aligned} f_{\text{reboot}}(H) &= \lceil \rho H \rceil, \\ f_{\text{weight}}(T_{\text{i}}) &= 1, \\ f_{\text{commit}}(C) &= \lceil \gamma C \rceil, \end{aligned} \tag{4.2}$$

where $\rho, \gamma \in [0, 1]$. By relying on the history of execution EG eliminates the problem of frequent power failures on a single coalesced task. In fact, at each reboot, EG will *con-*

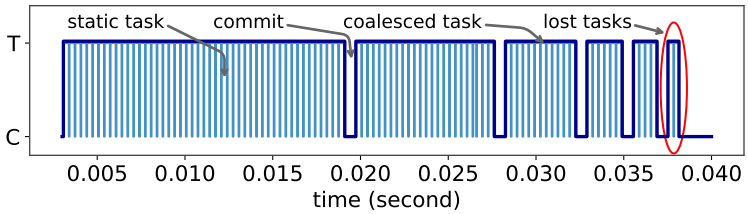


Figure 4.4: Anatomy of Coala coalescing. A snapshot of measured data showing how Coala coalesces static tasks. Coala takes advantage of the guarantee that an energy buffer offers after a reboot: code execution resumes after energy accumulation reaches the operational threshold (a maximum threshold). Thus, Coala starts with a big coalesced task and shrinks it gradually as the risk of a power failure increases. T stands for task code execution and C for commit or power cut.

servatively try to coalesce only a fraction (ρ) of tasks that had successfully completed during the last power cycle. Fig. 4.3 illustrates a snapshot of the operation of EG, with $\rho = \gamma = 0.5$. The figure shows two power cycles following the ith, the latter having an execution history $H^{(i)} = 4$. Based on that, C is initially set to two and then decreases to one. Once the value of C reaches one, EG is expecting a power failure, justifying the conservative approach. After the reboot, EG uses the most recent history $H^{(i+1)} = 8$ to set $C = 4$ and continue execution.

Weighted Energy-Guided Coalescing. The *Weighted Energy-Guided* (WEG) coalescing strategy accounts for non-uniform energy and time costs of a program's tasks when setting C. Each different task in the program consumes a different amount of energy to run to completion. The EO and EG coalescing strategies assume that each task has the same cost: for these strategies, C simply corresponds to a target *count* of tasks to coalesce, regardless of the individual cost of each task. However, if one task executes for ten seconds and another executes for one second, counting tasks misjudges the amount of *work* in the coalesced tasks (and the history). Instead, WEG associates a non-unity weight with each task in the program, and tracks the sum of the weights of tasks in a coalesced sequence. When the sum of weights reaches C, the target, WEG commits the coalesced task. WEG is characterized the same way as EG (equation (4.2)), except that $f_{\text{weight}}(T_\text{i}) = W_\text{i}$.

Fig. 4.4 shows a real measurement of Coala coalesced tasks. It shows how Coala coalesces aggressively when the execution is just re-started, taking advantage of the full energy buffer, and how it reduces the coalesced task gradually to minimize the risk of significant progress loss when a device fails. The average measured static task size is ≈ 0.28 ms while the commit time (time needed to save the data into non-volatile memory) averages at ≈ 0.64 ms. This highlights the importance of coalescing.

4.3.2. TASK DOWNSCALING

Coala uses *task downscaling* to make progress through a task that is too large to complete using the buffered energy. Task downscaling executes part of the long task and

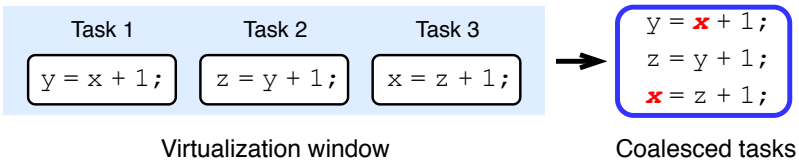


Figure 4.5: Task coalescing-induced WAR dependency (in the case of this example, at variable x). Such dynamic dependencies cannot be resolved at compile-time and they break memory consistency.

interrupts its execution with a *partial commit*, after which the long task continues. This timer-based solution is similar to the one proposed in [29]. If power fails, the partial commit preserves the progress through the task, and execution resumes from the point of the partial commit, rather than from the start of the task. Eventually, after some number of partial commits and power failures, the task will complete and execution will proceed.

Detection of Non-terminating Tasks. The key design issue for task downscaling is deciding when to partially commit. A task is likely to be non-terminating if it fails to run to completion twice consecutively. The second incomplete run executes after a power failure, when the device will have fully recharged its energy buffer. If a task cannot complete when executing with a full energy buffer, Coala marks the task as non-terminating. Task downscaling violates task atomicity in a non-terminating execution, favoring continued progress over atomicity. If a programmer requires a task's atomicity to be preserved, they can disable task downscaling for that task or a portion of it.

4.4. COALA MEMORY MANAGEMENT

Resolving WAR (Write-After-Read) dependencies by considering static task boundaries is not sufficient to keep non-volatile memory consistent when tasks are coalesced. Not handling WAR dependencies on a coalesced task scope introduces a new problem, which we denote as *task coalescing-induced WAR dependency*. This problem is illustrated in Fig. 4.5. Therefore, Coala implements a novel *Virtual Memory Manager* (VMM) that enables safe task coalescing and ensures efficient data manipulation.

VMM overview is given in Fig. 4.6. VMM abstracts the physical address space of the non-volatile memory (FRAM) and divides it into `private` and `shadow` (underlined locations are non-volatile) buffers, and each buffer is divided into pages. `private` holds the consistent version of pages after each commit (and on a reboot), while `shadow` enables atomic two-phase commit: it allows Coala to ensure a persistent and consistent set of non-volatile pages before copying them to `private` (Section 4.4.3).

The VMM prohibits applications from directly accessing these buffers. Instead, it redirects any request to the `working` buffer: a relatively small buffer located in the volatile memory (SRAM), which has a lower latency and energy cost to access than FRAM. It populates the `working` buffer with the privatized pages from non-volatile memory requested by tasks. With the help of `shadow`, the `working` buffer can serve more pages than its own capacity.

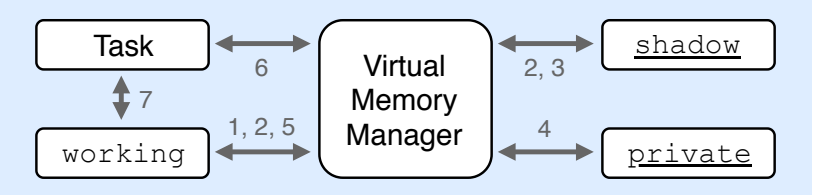


Figure 4.6: Interaction between task, memory manager and memory buffers upon RP and WP, and order of occurrence. Steps 2, 3, 4 and 5 are conditional.

Algorithm 2 RP(variable v)

```
1: t ← GETTAG(v)
2: i ← {j | GETTAG(working[j]) = t}                ▷ Search page
3: if i = ∅ then                                   ▷ Variable in a resident page?
4:     i ← PAGEFAULT(t)                            ▷ Swap page in
5: o ← GETOFFSET(v)
6: return working[i][o]                            ▷ Return from page
```

When a coalesced task completes, the VMM ensures that *dirty*, i.e., modified, pages are committed to FRAM and visible to subsequent tasks. If a power failure interrupts a task, all temporary modifications to pages in `working` (and `shadow`) occurred after the last commit are discarded in order to keep data consistency. On a commit, the VMM atomically copies, through a two-phase commit, all dirty pages back into their location in the non-volatile memory.

4.4.1. ADDRESS TRANSLATION AND VARIABLE ACCESS

A task must access protected non-volatile variables through Coala's restricted memory interface. The interface includes RP(v), to read the value of variable v, and WP(v) to assign a value to variable v. The implementation of RP is shown in Algorithm 2, and WP's implementation is similar except that accessed page are marked as dirty. RP and WP operations translate a variable's physical address in non-volatile memory into a *virtual address* in `working` in volatile memory. In Coala, a virtual address is composed of a *page tag* that identifies the page and a *page offset* that identifies a byte.

After address translation (Fig. 4.6, Step 6), a task accesses the protected variable's location in the volatile `working` buffer (Step 7). The VMM keeps track of the page tags for the pages currently resident in the `working` buffer. When a task accesses a variable, it compares the variable's page tag to tags of the pages in `working` (Algorithm 2, Line 2). If the accessed variable's page tag is not found in the page buffer, the operation incurs a *page fault* (Line 4). The byte is accessed in the page buffer at the index of the resident page and the variable's page offset (Lines 5–6).

4.4.2. PAGE FAULTS AND PAGE SWAPPING

When accessing a protected variable with RP or WP, the memory manager first searches the variable's page in the `working` buffer (Fig. 4.6, Step 1). If the page is not found there, a page fault is incurred and a new page needs to be swapped in. If the `working` buffer is full, a page fault on memory access requires the VMM to swap out one of the pages in the

working buffer (a *victim page*) preserving updates made to that page. If a task modified any byte in the victim page (i.e., using WP), then the page is dirty and its changes have to be persisted to non-volatile memory (Fig. 4.6, Step 2). If the accessed page was previously modified and swapped out since the last power failure, the most recent version of the page is in shadow (Step 3), otherwise it has to be retrieved from private (Step 4). Finally, the page is copied to the volatile buffer (Step 5).

4.4.3. ATOMIC TWO-PHASE COMMIT OF DIRTY PAGES

When the last task in a sequence of coalesced tasks completes, Coala must commit *all* dirty pages in working and shadow, copying them back to their original locations in private.

To make the commit atomic, Coala commits dirty pages in two phases, as shown in Algorithm 3. The first phase copies dirty pages from working to the non-volatile shadow (Line 4). The second phase commits pages from shadow to private (Line 12). If power fails during the first phase, the whole commit is aborted, and the execution restarts from the most recently committed point (i.e., from the beginning of a coalesced task). If power fails in the second phase, the commit process safely resumes on reboot. The second phase depends on some run-time metadata. The committing bit indicates that a commit is in progress and is set before the first page is committed (Line 9) and cleared after the last page is committed (Line 16). The shadowCount records the number of dirty shadow pages to be committed and the VMM clears the counter when commit completes (Line 14). The commitIndex indexes the next page to be committed (Line 11) and the VMM clears the index at the end of the phase (Line 15).

Coala's commit is efficient because it maintains an index of dirty pages instead of iterating through all potentially dirty pages to check their state. Another source of efficiency lies in the second phase of the commit: the VMM does not copy page content from shadow to private, instead it swaps pointers in an indirection table that maintains the pages as a double buffer.

Memory Consistency. Coala's paging mechanism ensures that a task only ever executes using consistent protected data. During task execution, modifications to protected data do not affect the private buffer, as a task reads and writes the volatile working buffer only, and modified pages are kept in shadow until commit. A power failure erases the contents of the working buffer, preventing a re-executing task from observing updates from a previous execution attempt. Clearing shadowCount as part of the second phase commit (Line 14) ensures that all accesses to protected variables in subsequent tasks correctly access their consistent memory locations in private. This solves the coalescing-induced WAR dependency problem (see Fig. 4.5 again).

4.4.4. DYNAMIC PAGING OF COALA

Coala asks the programmer to use its RP and WP API methods on every access to a protected variable (Section 4.5.1). These API invocations present a risk of high overhead because there is a dynamic check on every read and write. Despite the risk of per-access overhead, Coala's dynamic memory protection scheme brings several benefits over a static approach (i.e., [34]). First, the limitations of static analysis preclude some uses

Algorithm 3 Two-phase commit

```
1: procedure COMMITPHASE1                                   ▷ On completion of a coalesced task
2:     for i ∈ 0..|working| − 1 do
3:         f_dirty, t ← TAGFLAG(working[i])
4:         if f_dirty then                                  ▷ Is the page content modified?
5:             shadow[TAG(working[i])] ←DMA working[i]
6:             shadowList[shadowCount] ← t
7:             shadowCount ← shadowCount + 1
8:     COMMITPHASE2
9: procedure COMMITPHASE2                                   ▷ commitIndex 0 on first boot
10:    committing ← true
11:    while commitIndex < shadowCount do
12:        t ← shadowList[commitIndex]
13:        COMMITTOPRIVATE(shadow[t])   ▷ Copy shadow to private by swapping their pointers
14:        commitIndex ← commitIndex + 1
15:    shadowCount ← 0
16:    commitIndex ← 0
17:    committing ← false
18: procedure ONBOOT                                        ▷ Invoked on every boot
19:    if committing then COMMITPHASE2
20:    shadowCount ← 0
```

of pointers due to potential pointer aliasing. For example, in the presence of arbitrary pointer operations, a function call using a function pointer, or an interrupt within a task, the system cannot statically analyze the memory behavior. Second, a static approach *cannot handle task coalescing*, because a protected variable's lifetime, i.e., from first use to commit, is unknown at compile-time. Coala's dynamic, per-access instrumentation supports arbitrary use of pointers and enables task coalescing.

4.5. COALA IMPLEMENTATION

We implemented Coala's programming and execution model as a runtime library and API that a programmer can use to make a plain C program intermittency-safe.

4.5.1. APPLICATION PROGRAMMING INTERFACE

Coala's API adds only a few syntactic constructs to a C-based language, summarized in Table 4.1.

New Tasks. The `TASK` annotation on a function declaration statically allocates a non-volatile constant variable holding a task's weight and declares that the function is a task.

Task Transitions. `NEXT_TASK` marks the task to be executed after the current one and it can be invoked along any control path to dynamically determine the next task.

Protected Variables. The `PV` annotation on a variable statically allocates a protected non-volatile variable. The variable must then be accessed with the `RP` and `WP` API methods at run-time to ensure correct operation. The `SM` annotation is used to align C structure data type within a page.

Table 4.1: API summary; **T**: set of all tasks, **V**: set of all protected variables, [, *s*]: optional argument.

Method	Arguments
`INIT`(t)	$t \in \mathbf{T}$: scheduled task on first boot
`RUN`()	—
`TASK`(t, w_t)	$t \in \mathbf{T}$: task name, w_t: weight of task t
`NEXT_TASK`(t)	$t \in \mathbf{T}$: task to be run next
`PV`(p, v [, s])	p: type, $v \in \mathbf{V}$: name, s: array size
u := `RP`(v)	$v \in \mathbf{V}$: protected variable to read, u: dest. operand
`WP`(v) := u	$v \in \mathbf{V}$: protected variable to write, u: source operand
`SM`(p, m [, s])	p: type, m: name, s: array size
`DISABLE_PC`()	—
`ENABLE_PC`()	—

Initialization. The behavior of the API method `INIT` is very similar to `NEXT_TASK`, with the addition of performing preliminary kernel initializations, including hardware setup.

Execution. The programmer passes control to Coala's task scheduler by calling `RUN` after device initialization.

Partial Commit. The `DISABLE_PC`() and `ENABLE_PC`() allow a programmer to disable partial commit around certain code.

4.5.2. INITIALIZATION PROCEDURE

On a reboot, Coala's scheduler does a number of system level operations before executing a task. First, it updates the coalescing target according to the applied coalescing strategy. Then, it finishes any interrupted commit and clears the list of dirty shadow pages. After that, the scheduler sets the program counter to the next task to run, which Coala tracks in non-volatile memory. Before executing the task, Coala checks whether there is a partially committed task to resume, which requires Coala to restore the volatile state, including the program counter. If there is no in-progress, partially committed task, Coala starts executing and coalescing tasks.

4.5.3. TASK COALESCING

Parameters. Equations (4.1) and (4.2) parametrize the behavior of the coalescing strategies in terms of x, ρ and γ. We experimented with a range of values, and then used the ones that yielded the best performance: $x = 1$ (for EO) and $\rho = \gamma = 1$ (for EG and WEG).

Weights for WEG. The effectiveness of the WEG hinges on correctly identifying the weight of each task, which WEG assumes is *statically* available. Profiling the time and energy cost of tasks in a program is a difficult, orthogonal problem [38, 90]. WEG could use the result of an arbitrarily sophisticated profiling procedure. To produce a concrete result, we give WEG access to a simple profile of task run time (collected offline) using a

single fixed input. WEG stores the profile in a lookup table that maps a task's identifier to its weight, making the information available to Coala's scheduler at run time.

4.5.4. TASK DOWNSCALING

Timer Strategy. After identifying a task as likely non-terminating, Coala must decide when during the task's execution to partially commit the task's state. Coala sets a timer at the start of the likely non-terminating task, initializing it to a very large value (e.g., an estimate of the maximum execution time using the device's energy buffer). If the timer expires before power fails, Coala partially commits, retaining the timer value to use for future partial commits. If the timer does not expire before power fails, Coala halves the timer and tries again. The exponential decrease of the timer value converges rapidly to a usable one.

Partial Commit and Task Atomicity. Some applications may prevent downscaling a task because of a need for task atomicity. For example, sampling an analog signal requires consecutive samples at a known interval, or the digitally sampled signal is meaningless. In such a case, the programmer can disable partial commit for a task or a span of code, marking the code with a pair of `DISABLE_PC` and `ENABLE_PC` annotations. These annotations respectively halt and resume the partial commit timer.

4.5.5. PAGING

Efficiency. Coala uses address-based page tagging to make finding a variable efficient. The upper bits of a variable's memory address identify its page, and the lower bits denote the variable's offset in its page. The total number of pages in memory, P, determines the number of tag bits, which is $\log_2 P$. Furthermore, the VMM moves pages of data efficiently using hardware-accelerated DMA support.

Alignment. Page tagging imposes a data alignment requirement. The page size S must be a power of two. Pages must be aligned to an S-byte boundary for efficient memory access. To preserve alignment, when typedef'ing a C structure for protected variables, the programmer has to use the API method `SM` on all members of the structure (only when defining the structure).

Page Eviction. When a page in `working` has to be swapped out to make room for a newly requested one, an eviction policy has to be chosen. While we opted for the simplicity of FIFO, any other replacement policy, such as Least Recently Used, could work in its place.

We experimented with 32-, 64-, 128- and 256 B pages, an 8 kB non-volatile `shadow` buffer and an 8 kB non-volatile `private` buffer, and a `working` buffer of 1 kB.

4.6. METHODOLOGY

We prototype Coala and use its API to implement a set of benchmark applications representative of the embedded domain. We build the applications and deploy the binaries onto a real energy-harvesting device.

Table 4.2: Characteristics of benchmarks used for evaluation.

App.	Tasks	SLOC	Description
ar	10	428	activity recognition using a KNN
bc	10	371	several bitcount algorithms
cuckoo	14	426	Cuckoo Filter with pseudo-random values
dijkstra	5	198	Dijkstra shortest path algorithm
fft	8	449	Fast Fourier Transform
sort	4	167	selection sort algorithm

4.6.1. EXPERIMENTAL SETUP

We used three different setups to evaluate Coala: (i) an RF-powered energy-harvesting device, WISP 5.1 [20, 77], with a fixed capacitor size of 47 μF; (ii) an MSP-EXP430FR5969 launchpad [146] powered by a BQ25570 [147] solar power harvester that is connected to an IXYS SLMD121H04L solar cell [148] for experimenting with different energy buffers; and (iii) an MSP-EXP430FR5969 launchpad [146] that is continuously powered.

Every benchmark used in Section 5.4 was run repeatedly on each platform for a few minutes to ensure capturing a diverse power trace. The exact number of iterations depends on the ambient power intensity and the application itself. In our experiments, the number of complete runs ranges from 4 to 125.

WISP contains a low-power MUC (MSP430FR5969 [149]) with 64 kB of non-volatile (FRAM) memory and 2 kB of volatile (SRAM) memory and was configured to 1 MHz clock speed. WISP was powered using an RF signal generator emitting a 20 dBm sinusoidal wave at 915 MHz. The signal generator was connected to the Laird RFMAX S9028PCRJ 8 dBic antenna [150]. The antenna was oriented towards and in parallel with WISP's antenna, and no objects obstructed the path. We affixed WISP with a paper harness at the edge of a table at a height, from the table surface, of 10 cm. For distance-controlled experiments we positioned WISP at $d = \{15, 30, 50\}$ cm from the exciter antenna. To obtain execution time, the software toggled GPIO pins at sections of the code under profile, and the Saleae [135] logic analyzer measured intervals between edges in the signal. For continuous power experiments, the execution time was measured using the clock features in TI Code Composer Studio IDE version 7.1.

4.6.2. SOFTWARE BENCHMARKS

We evaluated Coala using six benchmarks that are often used in embedded systems (summarized in Table 4.2). All applications were compiled using MSP430 GCC [151] version 6.4.0 with `-O1` as optimization flag. The source code for all benchmarks is released via [152].

Comparison with Alpaca using the GCC compiler. We compare Coala against Alpaca, a state-of-the-art task-based system for intermittent computing. For a fair comparison, the task decomposition of the benchmarks is ensured to be the same for both systems. Since Alpaca's compiler pass is implemented only for LLVM, we could not use that implementation to instrument the benchmarks and compile them with GCC. Instead, we

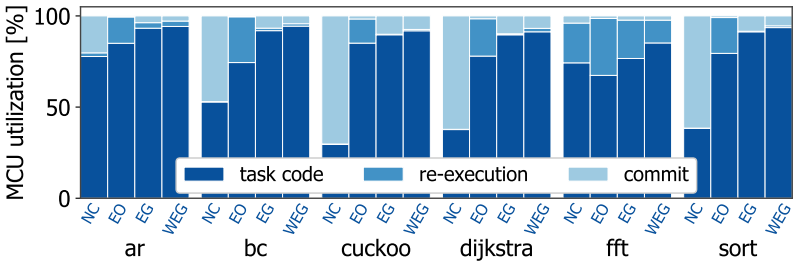


Figure 4.7: Breakdown of overhead for the proposed coalescing strategies: NC (no coalescing), EO (energy-oblivious), EG (energy-guided), WEG (weighted energy-guided). All coalescing strategies reduce total overhead and maximize useful work. EG and WEG perform better than EO.

manually performed the instrumentation done by the compiler pass. The instrumentation consists of identifying WAR dependencies and adding code and memory allocations to make a private copy of the affected variables. By compiling both systems with the same compiler (GCC), we ensure that our comparison in Section 5.4 is fair.

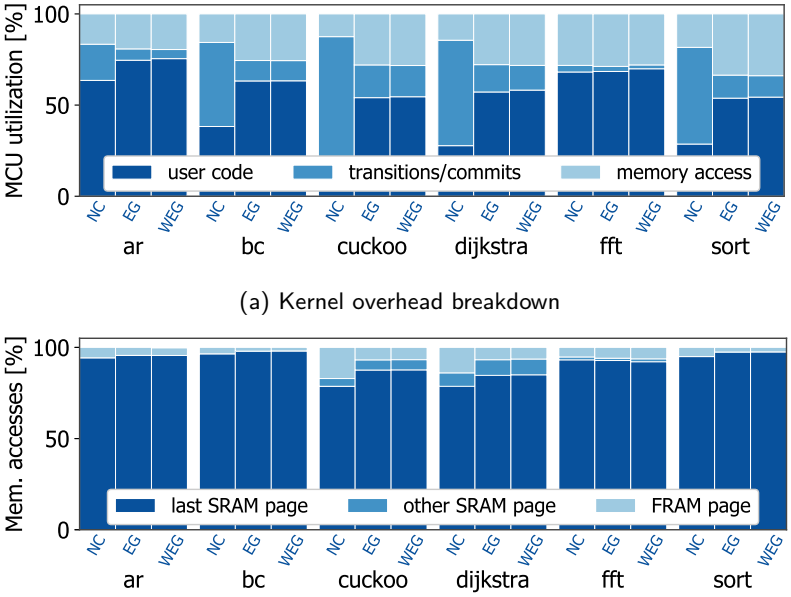
4.7. COALA EVALUATION

Our evaluation quantitatively demonstrates that Coala (i) *reduces memory protection overhead*, (ii) *improves execution speed* in most cases, and (iii) *is able to progress* where static systems suffer from a task non-termination loop.

4.7.1. CHARACTERIZATION OF OVERHEAD

To characterize Coala's overhead we experimented with WISP positioned at 15 cm away from the signal generator antenna. We have broken down Coala's overhead to explain the source of its improved performance.

Overhead Reduced by Coalescing. For each coalescing strategy from Section 4.3 (EO, EG, and WEG) and for a baseline without coalescing (NC), we have measured the time spent on executing (i) useful task code, (ii) task code wasted due to a power failure, and (iii) commits to non-volatile memory at the end of each (coalesced) task. The overhead incurred by each coalescing strategy is broken down in Fig. 4.7. Without coalescing enabled (NC), the re-execution penalty is the smallest, because the amount of work that can happen between commits and may have to be re-executed if interrupted is reduced when work from multiple static tasks is not combined. However, any gain from a reduced re-execution penalty is canceled out by the increased commit overhead that is incurred at the end of each static task. Across all benchmarks, all Coala's coalescing strategies reduce more commit overhead than the re-execution overhead they add. This net overhead reduction is greatest in EG and WEG strategies compared to the EO strategy. We attribute this discrepancy to EO's energy-unaware adjustment to the coalescing target. In the subsequent experiments, we focus on the better-performing EG and WEG.



(a) Kernel overhead breakdown

(b) Protected memory accesses breakdown

Figure 4.8: Coala's internal overhead. NC: no coalescing, EG: energy-guided, WEG: weighted energy-guided.

Coala's Kernel Overhead. Fig. 4.8a breaks down the time spent on executing user task code versus the time spent on kernel operations. When coalescing is not enabled the commit overhead is highest. The overhead for memory accesses increases in percentage when enabling coalescing, but not in absolute terms. For both coalescing strategies (EG and WEG) accesses to protected variables constitute about 30% of the runtime overhead. Dynamic address translation necessary on each protected access is the most critical bottleneck for Coala.

Protected Memory Accesses Breakdown. Fig. 4.8b breaks down protected memory accesses into three categories. Each type of protected access incurs a different overhead. Accessing the most recently used SRAM page is of the cheapest kind. Accessing a different page in SRAM has a slightly higher cost. Finally, accessing a page that needs to be swapped in from FRAM into SRAM is the most expensive. The results in the figure show that the overwhelming majority of accesses are of the cheapest kind, which motivated us to optimize this accesses in our implementation. Only *cuckoo*, *dijkstra*, and *fft* have non-negligible number of accesses to a different SRAM page, which is due to the larger working set and a less regular access pattern in these applications. In general, memory access patterns are shaped by the application, and the more program state is protected, the higher the rate of page swaps.

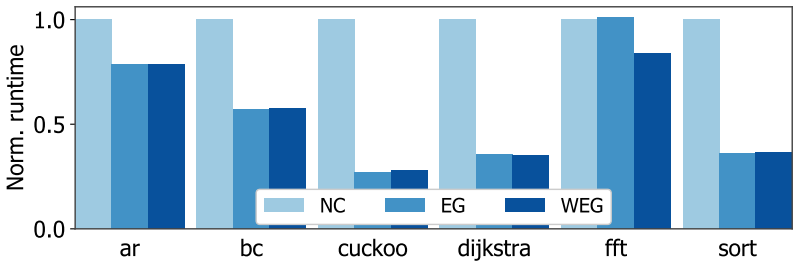


Figure 4.9: Coala's coalescing performance. Application execution time with coalescing (EG and WEG) normalized to the execution time without coalescing (NC).

Cap. size (μF)	**Exp. time** (s)	**On/Off cycle**	**Coal. task** (ms)	**Runs**	**Run time** (ms)
47	1205	12.93%	33	85	183
470	1205	12.79%	88	87	177

Table 4.3: Comparison of Coala performance running the *sort* application on two different capacitor sizes. The results show that Coala optimizes its coalesced task size based on the energy buffer size. **Coal. task** refers to the length of the first coalesced task after a reboot, **Exp. time** shows the experiment duration, the **Runs** column lists the number of complete runs of the application during the experiments. **Run time** is the device collective uptime needed to finish a single iteration of the *sort* application.

4.7.2. EXECUTION TIME

Having shown in Section 4.7.1 that coalescing reduces overhead, we now investigate the outcome of this reduction on the total execution time. We first investigate different variants of Coala and then compare the best variant to Alpaca [34]. Additionally, we compare Coala performance running with different energy buffer sizes.

Speedup with Coalescing. Fig. 4.9 shows Coala's run time of two coalescing strategies (EG, WEG) normalized to the run time without coalescing (NC). The results show that all benchmarks complete faster with coalescing than without coalescing: from 25% (*ar*) up to 70% (*sort*). This speedup is a consequence of the reduced overhead demonstrated in Section 4.7.1. However, the magnitude of the speedup is (1) highly application-dependent and (2) largely similar across the two coalescing strategies, with the exception of *fft*. In some cases (*bc*, *cuckoo*, *sort*) WEG's task weighting system is counter-productive. This occurs in task decompositions with energy-uniform tasks, where counting tasks disregarding their energy consumption provides an equal amount of information with a smaller effort. In *fft*, tasks are not uniform, and accounting for their different weights is beneficial. In fact, the lack of task energy awareness is detrimental: with EG *fft* runs slower than without any coalescing (NC). The speedup is highest for *bc*, *cuckoo*, *dijkstra* and *sort*, because their tasks are relatively small and are easily coalesced.

Benefits of Adaptive Tasks. We now compare Coala's performance to Alpaca [34]—a *non-adaptive* task-based system with tasks fixed at compile-time. Fig. 4.10 shows the

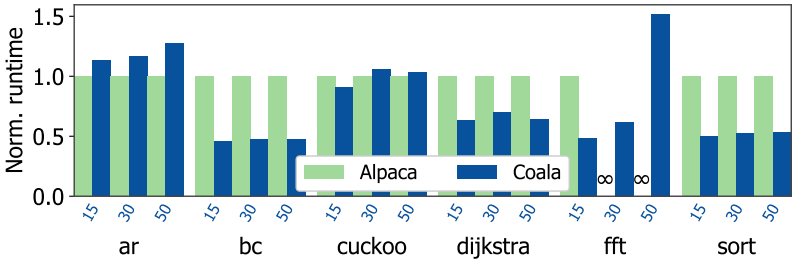


Figure 4.10: Coala's execution time normalized to Alpaca's, for three distances from the energy source (15, 30 and 50 cm).

average execution time of each application for Coala and Alpaca, normalized to the latter or, when not possible, to one second. Coala provides a performance benefit compared to Alpaca for most applications. For example, it is 54% faster than Alpaca when executing the *bc* application. In general, the speedup is greatest for applications with repeated WAR dependencies throughout their code, particularly involving arrays (*dijkstra, fft* and *sort*). Coala's VMM successfully amortizes the overhead of protecting memory that is accessed in such patterns. In applications without locality among accesses to protected variables Coala incurs overhead from memory virtualization that causes its performance to be comparable to (or worse than) Alpaca (*ar, cuckoo*).

Due to its static progressing behavior, Alpaca was unable to complete the *fft* benchmark on distances larger than 15 cm[1]. This is marked with ∞ signs in Fig. 4.10. Coala, however, managed to complete *fft* by enabling its task downscaling at 30 cm and 50 cm from the energy source.

Different Capacitor Sizes Table 4.3 shows how Coala optimizes its Coalescing task size based on the amount of buffered energy at runtime. This means that applications implemented in Coala are portable across devices with different capacitor sizes without recompilation; they are also more resilient to degradation in capacitor size due to temperature and device lifetime.

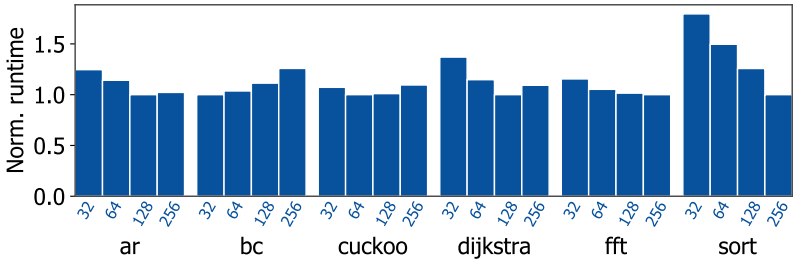
We see that Coala scales up its coalesced task size with a bigger energy buffer and vice versa. This allows it to reduce the time-to-completion of the applications. For example, the *sort* run-time is reduced from 183 ms to 177 ms when the capacitor is changed from 47 μF to 470 μF. It should be emphasized that a device with a bigger energy buffer suffers less from power failures (but requires longer charging time).

Overall, Coala shows better performance than its counterpart, and it is able to overcome the *big task* (i.e. *fft* tasks) problem that the static task-based systems suffer from.

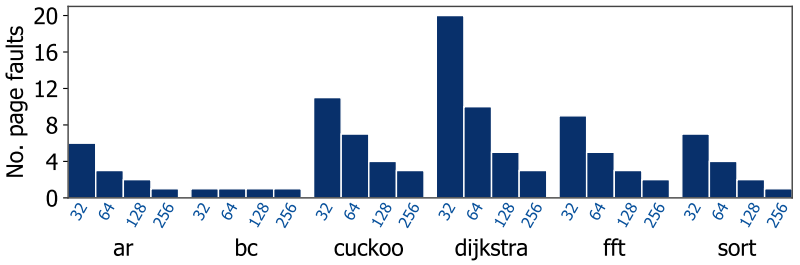
4.7.3. VIRTUAL MEMORY PERFORMANCE

We characterize the performance of Coala's virtual memory sub-system in an experiment on a continuously-powered evaluation board, as described in Section 4.6.1.

[1] At distances less than 15 cm the total amount of energy available for task execution includes significant amount of energy being harvested while the device is executing in addition to the stored energy.



(a) Execution time normalized to lowest per-application



(b) Number of page faults per application run

Figure 4.11: Effect of page size (in bytes).

Effect of Page Size on Runtime. Fig. 4.11a shows the execution time as a function of page size (in bytes), normalized to the lowest per-application performance among the set of page sizes. The data suggest that there is a page size that minimizes execution time. The best page size is not the same for each application. Nevertheless, if a choice must be made for all applications, 128 B pages are the best option.

Effect of Page Size on Page Faults. Fig. 4.11b reports the number of page faults, per application run, as a function of page size (in bytes). The smaller the page the more likely that a memory access will land outside that page and that a new page will have to be swapped in. This trend is visible for all applications, except for *bc*. The total amount of data accessed by *bc*, as well as its working set, is small. Even with the smallest page, all accesses are contained within that page, and no page fault occurs. Without any page faults to begin with, increasing the page size only yields overhead.

4.8. CONCLUSIONS

Software for intermittently-powered energy-harvesting devices requires a dedicated run-time system. Coala is a new task-based system whose distinguishing feature is its adaptability to changing energy conditions at run-time. When more energy is available, Coala makes faster progress through the computation by coalescing statically-defined tasks. When less energy is available, progress is latched at sub-task granularity. Coala's page privatization system ensures that program state in non-volatile memory remains con-

sistent and amortizes the cost of state transfer between volatile and non-volatile memory. Our evaluation across applications on a real embedded energy-harvesting device demonstrates the utility of Coala, as well as its practicality when heterogeneous devices are considered.

The primary application of tiny energy-harvesting battery-less devices is sensing. InK and Coala provide reactive and dynamic execution, which enable efficient intermittently-powered sensors to directly react to external events. Energy-harvesting battery-less sensors, however, are most of the time off charging. Therefore, they miss many sensing opportunities, which violates the *available* requirement of many real-world applications. This limitation is addressed in the next chapter.

5

COALESCED INTERMITTENT SENSOR

5.1. INTRODUCTION

An intermittent power supply introduces a set of new challenges. In Chapters 1 and 2, we have argued that prior work has addressed challenges associated with intermittent computing. Then, Chapter 3 and 4 detailed what it takes to enable reactive (InK) and dynamic execution (Coala) on intermittent power. Such an execution model is well suited for driving sensing applications because these applications are reactive in nature (e.g., a sensor reacting to a change in temperature). Despite that, intermittently-powered sensors still suffer from a fundamental shortcoming: *the intermittent availability of the system*. Being frequently off compromises the value of these devices. For example, a sensor that has a low probability (e.g., 10% [74]) to be available when an event of interest occurs has no value. Overcoming the intermittent availability challenge without changing the

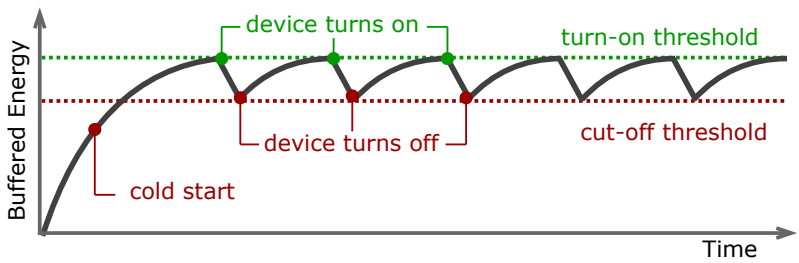


Figure 5.1: Harvested energy profile of an intermittently-powered device. Ambient power is weak; therefore, it is usually buffered. The buffered energy is then consumed to operate the device. The operation period is often short as power consumption is much higher than the energy harvesting rate.

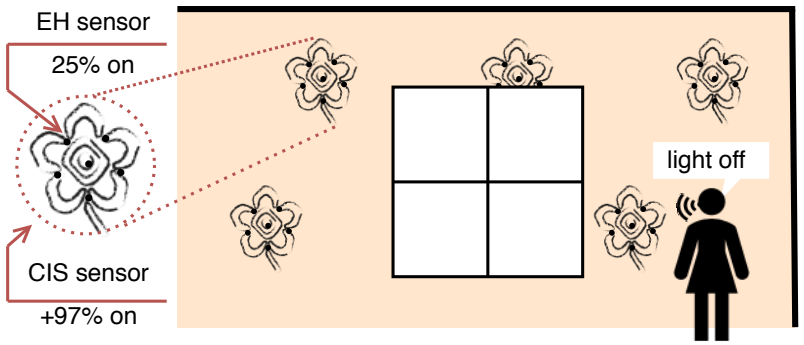


Figure 5.2: A Coalesced Intermittent Sensor (CIS) is a group of energy-harvesting (EH) battery-less nodes that sense continuously despite the intermittent power supply. Being a collection of tiny sensors facilitates embedding a CIS in patterns without significant distortion (the dots in the flowers). A CIS exploits the inherent randomization of energy-harvesting systems to approach continuous sensing availability. On the left, we see that collective availability of a CIS is +97% while its individual sensors are only 25% of the time on.

size of the device or re-including batteries requires a novel approach that explores new design dimensions.

5.1.1. VISION AND APPLICATION

Miniaturized sensors are non-intrusive devices. Therefore, they can be embedded in new locations and enable new applications. Miniaturizing sensors, however, signifies their powering challenge. On one hand, batteries make these sensors *continuously* available -for sensing opportunities- but for a short period of time (even rechargeable batteries wear out after a certain number of charging-discharging cycles [59, 153]). On the other hand, removing batteries and relying on ambient energy make them available for *extended period of time*, but intermittently. Our vision is that by combining multiple battery-less energy-harvesting sensors we can create a new virtual sensor that operates permanently (no batteries) and reliably (continuously available): we call this sensor the *Coalesced Intermittent Sensor* (CIS).

Sensors with such characteristics would allow us to add a cheap and maintenance-free sensing layer to many objects, making them smart and interactive. For example, one can imagine developing smart wallpaper that users can interact with. Smart wallpaper with embedded microphones can enable direct in-building human-to-object communication (Figure 5.2). Such a permanently operating sensor can be deployed, for example, in kids' playgrounds to monitor their occupancy. These battery-less sensors can enable interactive and safe-to-dispose sports rugs (that count how many times a person has jumped on them) or play rugs for kids. In short, we would like to develop small sensors with permanent and continuous sensing capabilities.

5.1.2. RESEARCH CHALLENGES

Many sensing applications require the sensor to be available when there is a change in the monitored environment. Energy-harvesting battery-less sensors can provide cheap

and maintenance-free sensing, but they do not meet the availability requirements of many real-world applications.

C1-Approach continuous availability on intermittent power: A battery-less sensor collects energy from ambient sources to power its operations. Such a sensor, however, is frequently off, spending most of the time charging. One way to increase the system availability is by using multiple nodes. However, coordinating the nodes' awake times using communication may introduce prohibitive overhead as a scattering algorithm must be regularly executed, and messages for synchronizing nodes' clocks and reserving time slots need to be repeatedly exchanged. Thus, the question is *can we exploit some of the inherent characteristics of energy-harvesting battery-less sensors to distribute nodes' awake times without the need for communication?*

C2-Continuous sensing on intermittently-powered sensors: Even when the collective availability of intermittent sensors approaches 100%, the emerging overall sensing behavior may still be intermittent. Event-triggered sensors sleep in low-power mode waiting for an event to wake them up. When ambient energy rises, the energy-harvesting rates of these sensors may equal (or approximate) their sleeping mode power consumption. Under such energy conditions, these sensors become available for an extended period of time. Therefore, when an external event arrives, nodes respond collectively, which exhausts their energy buffers, making them unavailable for the next set of events. This is a significant problem when events arrive in bursts, like a command of a few words (e.g., "light on"). Thus, the question is, *how to prevent energy-harvesting battery-less sensors from synchronizing their power cycles on some of the incoming events?*

C3-Efficient sensing on intermittent sensors: One of the main factors that determine the intermittency pattern of an energy-harvesting battery-less sensor is the richness of ambient energy. For example, at mid-noon under direct sunlight, even a small solar panel can power a sensor node continuously. In such conditions (i.e., favorable energy conditions), using plenty of intermittent sensors would only result in duplicated work that leads to duplicated messages when the data is being communicated to a sink node: a continuously-powered node acts as a gateway for such sensors to communicate with the outside world. These messages will collide as they will be generated at approximately the same time, and if some of them are received by the sink, then they waste energy as they carry the same information. Thus, the question is, *how to reduce the number of duplicated event detections?*

5.1.3. CONTRIBUTIONS

In this chapter, we tackle the paradox of continuous sensing on intermittently-powered sensors. We studied the relationship between the power cycles of energy-harvesting battery-less devices, the emerging collective behavior, and the effect of the change in ambient energy on this behavior. In particular, we make the following contributions:

- We show how continuous sensing can be approached using multiple intermittently-powered sensors. For that, we **modeled** the collective effective availability—the system availability that leads to successful sensing—of a group of intermittent sensors and **validated** our models using simulation and on real hardware against different ambient energy sources.

- We introduce a new type of virtual sensor, showing its capabilities and limitations. This **Coalesced Intermittent Sensor (CIS)** is the abstraction of a group of energy-harvesting battery-less sensors that achieves maximum statistical availability by exploiting (inherent) randomization to approach uniform distribution of nodes' awake times.
- Contrary to common sense, we show how favorable energy conditions can deteriorate the performance of a CIS. We, therefore, equipped the CIS with **an new algorithm** that makes it ambient-energy aware. This algorithm enables the nodes to determine their own duty cycles (without requiring additional hardware), and the average number of alive nodes (without requiring communication). This information can effectively be used by the nodes to decide when to back off to avoid duplicated event detection and availability interruptions.
- We prototype, evaluate, and demonstrate the feasibility of the CIS concept in the form of a voice-control commands recognizer, the **Coalesced Intermittent Command Recognizer** (**CICR**). We chose to develop a command recognizer as voice is a natural way for the human to interact with small devices. Moreover, words allow us to easily experiment with individual event arrivals and events that arrive in bursts. However, our goal is *not* to present a novel word recognition technique. Instead, we adapts a classical word recognition algorithm to make it power-failure immune. Yet, our CICR prototype is the first intermittent command recognizer, shedding light on the potential of intermittent systems.

5.2. COALESCED INTERMITTENT SENSOR

The Coalesced Intermittent Sensor (CIS) is the abstraction of a group of energy-harvesting battery-less sensor nodes seeking to approximate the continuous sensing availability characteristic of a battery-powered sensor. The design of a CIS needs to consider four main aspects: (i) how the nodes' awake time is distributed; (ii) the consequence of emulating continuous sensing availability by chaining multiple short on-times; (iii) the effect of the environment on the CIS's availability; and (iv) the spatial coverage of the event of interest, which determines the diameter of the CIS.

Let us first characterize the power cycle of an energy-harvesting battery-less device. An energy-harvesting intermittent node frequently switches between off and on, charging energy and operating. We can characterize the time of this charge-discharge (or power) cycle using the following notation, $(t_{on},\ t_p)$, where t_{on} is the node's on-time interval, and $t_p := t_{on} + t_{off}$, where t_{off} is the node's charging time interval.

5.2.1. SENSING

The ability of a CIS to sense depends on the availability of its intermittent nodes and on the characteristics of the event of interest.

COALESCED AVAILABILITY

The CIS's availability is the projection of its underlying intermittent nodes' on-times on the time axis. To determine the expected availability of a CIS, the strategy to distribute its nodes' on-times must be first specified.

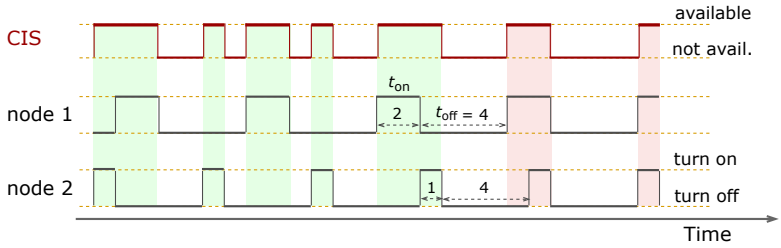


Figure 5.3: A Coalesced Intermittent Sensor's availability is the emerging collective availability of its intermittent nodes. Difference between the power cycles of intermittent nodes lead to constant relative shifts between their duty cycles. This, in turn, causes their on-times to approach uniformly distribution on the CIS power cycle. The red bars indicate a minimum CIS time span—CIS's nodes are overlapping—whereas the green bars show the maximum time span of the CIS.

Explicit on-time division strategy A CIS can build on top of the recent advancements in passive (light or RF) communication [17, 154] and ultra-low-power timers [35] to apply a time-division multiplexing strategy, minimizing on-times overlapping. For example, a node calculates its average on-time $\overline{t_{on}}$ and off-time $\overline{t_{off}}$ for a certain number of power cycles. Then, it encodes the information $(\overline{t_{off}}, \overline{t_{on}})$ in a message and broadcasts it at the beginning of its power cycle. When a node receives this message it will have full knowledge about the transmitting node's power cycle. It can then alter its power cycle, relative to the transmitting nodes cycle, by increasing (or decreasing) its power consumption to shorten (or lengthen) its on-time and subsequently shift its power cycle to a different time slot.

With such explicit on-times control strategy, a CIS of N nodes with on-time of $\overline{t_{on}}$ and off-time of $\overline{t_{off}}$ will have an availability = $\min\left(N\frac{\overline{t_{on}}}{t_p}, 100\%\right)$. However, we expect such an approach to introduce significant overhead as a scattering algorithm (e.g., [155]) must be frequently executed, messages need to be exchanged, and clocks should be synchronized. Therefore, we propose a different on-times spreading strategy.

Implicit on-time division strategy With no information being exchanged between intermittent nodes, the best CIS can do is to uniformly distribute its nodes' on-times and maintaining this distribution over time. The key observation to approach uniform distribution is to ensure that the lengths of the nodes' power cycles are randomized, avoiding nodes being in lockstep indefinitely.

Let us start by assuming that we have a CIS of two nodes with idealized power cycles and these nodes have the same initial conditions. The availability of this CIS equals t_{on} as the nodes are in perfect synchronization (the two nodes wake up and power down together). To extend the availability of this CIS, one of the nodes should shift its on-time away from the other. If one of the nodes sleeps for t units of time, then the on-time of this power cycle will be $t_{on} + \Delta t$. Consequently, the length of this power cycle will be $t_p + \Delta t$, delaying the next awake time by Δt. If the node sleeps only once, then availability of the CIS will equal $\min(2 \times t_{on}, t_{on} + \Delta t)$

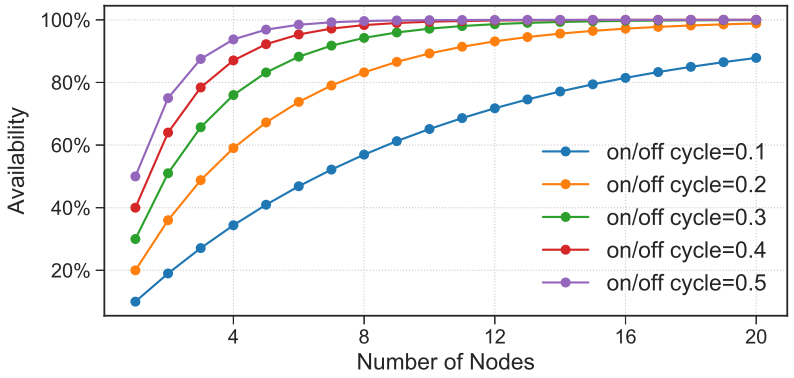
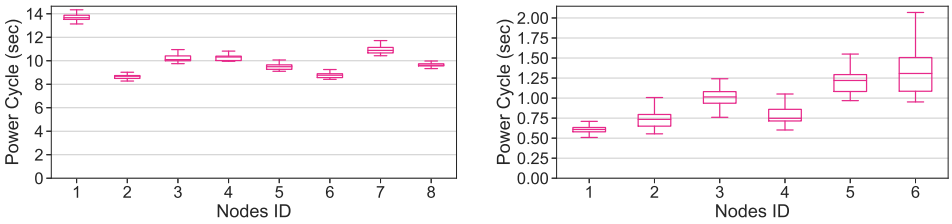


Figure 5.4: Coalesced Intermittent Sensor availability for a different number of nodes and different duty cycles. The nodes are uniformly distributed and the average CIS on-time evolves, when adding new nodes, according to equation 5.1.

However, if the initial conditions are unknown, then shifting a node's on-time a constant number of times may cause the initially desynchronized nodes to become synchronized, collapsing the CIS's availability instead of extending it. Therefore, a safer option is to *constantly* shift the awake time of the node. In this case, the on-time will shift over the entire power cycle of the other node, spending $\frac{t_{off}}{t_p}$ and $\frac{t_{on}}{t_p}$ of the time overlapping with the other node's off-time and on-time, respectively. This behavior is illustrated in Figure 5.3, where node 1 and node 2 have power cycles of (2,6) and (1,5). Following the time axis from the left to the right, we can observe that the position of the on-time of node 2 is shifted by -1 unit of time relative to the on-time of node 1 after each power cycle of node 1. This implies that the on-times of the two nodes are $\frac{1}{3}$ of the time cluster together and $\frac{2}{3}$ of the time they are apart (from an external event standpoint, the on-times are uniformly distributed over the longest power cycle, as they have the same probability to be anywhere when the event arrives). To model the availability of a CIS of N nodes, we first model the nodes' on-times and power cycles. If we represent the on-time of a node with a random variable R_n and find its expected value $E(R_n)$ then we can approximate *any* CIS node's on-time with mean of the expected values of the nodes' on-times, i.e., $t_{on} = \frac{1}{N}\sum_{i=1}^{N} E(R_n)^i$ (intuitively, since we are assume CIS's nodes have the same energy buffer, then their expected on-times should approach the same value). Using a similar analogy, we can define the mean of the expected values of the power cycles lengths as $t_p = \frac{1}{N}\sum_{i=1}^{N} E(R_p)^i$. Now, we can model the availability of a CIS of N nodes as

$$A_v(N) = A_v(N-1) + (1 - A_v(N-1))\,\frac{t_{on}}{t_p}, \tag{5.1}$$

for the initial case where $N = 1$ we define $A_v(0) := 0$. Figure 5.4 shows the availability of CIS when $N \in \{1,2,..,20\}$ and nodes' duty cycles $\frac{t_{on}}{t_p} \in \{10\%, 20\%, .., 50\%\}$. We can



(a) Light (680μF). Each MSP430FR994 lauchPad [156] was powered by a BQ25570 solar power harvester [147] that is connected to an IXYS SLMD121H04L solar cell [148] and a super-capacitor of 470 μF.

(b) RF (47μF). An RFID reader [157] connected to a circular antenna [158] were used to enrgize the WISP tags [77].

Figure 5.5: Nodes' power cycles length for different ambient energy sources, and different energy buffer sizes.

conclude from the above discussion that to approach uniform distribution of nodes' on-times, the lengths of the power cycles need to be randomized [1].

The power cycles of energy-harvesting battery-less devices are inherently randomized and different because the power source (ambient energy) is volatile and the harvesters are not perfect devices (notice that, even battery-powered wireless sensor nodes require a synchronization protocol to correct for the drift in their local clocks). Our own measurements using different energy-harvesting devices and different energy sources, i.e., solar and RF, also confirm that the power cycles of intermittent nodes are different and randomized (Figure 5.5). Therefore, we expect their on-times to approach uniform distribution (we will challenge this expectation in Section 5.4).

EVENTS CLASSIFICATION

The availability of a CIS is not a single stretched interval: it is a chain of short intervals. Therefore, it is important to classify from a CIS perspective which types of events the CIS is best suited for.

- *Short events*: are events that can be captured using a single intermittent node. For example, a spoken word can be seen as a short event if the energy needed to record it is less than what the energy buffer, i.e., the capacitor, can store.
- *Long events*: are events that need more energy to be completely captured than what the energy buffer can store. Long events can be subdivided into three categories:
  - *Simple*: is a long event that can be captured using a single intermittent node—capturing part of it is sufficient to obtain all the information of interest—such as the sound produced by the friction between two moving parts of an engine.

[1] Note that, power cycles of lengths that are multiples of each other is very unlikely because nodes' energy buffers are assumed to be of the same size.

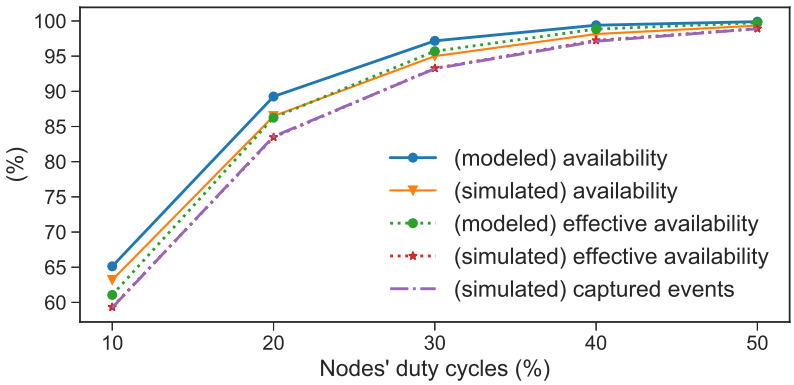


Figure 5.6: Simulating the availability, the effective availability, and successfully captured events of a CIS of 10 nodes with a node duty cycle $\in \{10\%, 20\%, ..., 50\%\}$.

– *Burst*: is a group of short events that requires multiple intermittent nodes to be captured such as a command of a few words (e.g., "room temperature up").

– *Complex*: is a long event that must be fully captured to be recognized. For example, sampling a gyroscope attached to a moving device (e.g., a toothbrush).

Based on the above classification, we can argue that designing a CIS for long events is not like designing it for capturing short ones. For example, while capturing a short event may require continuous CIS availability, capturing a long simple event that is longer than the power cycle t_p does not require extending the availability of a single intermittent node. Furthermore, capturing a long complex event may require data fusion and processing that require the CISs' nodes to communicate the raw data to a more powerful node, which may lead to significant overhead. However, we opt to focus on short and long bust events as they cover a wide range of applications (e.g., voice-controlled human-object interface).

EFFECTIVE AVAILABILITY

Approaching continuous availability does not mean that a CIS can successfully capture all events. It can happen that an event is being only partially captured by one or more nodes, which may lead to unsuccessful event detection. Therefore, it is important to specify the effective availability of a CIS that leads to a successful event capturing (which we assume leads to successful sensing).

Polling-based Sensing Let us assume that we have a CIS of a single intermittent node monitoring a short event of length t_e. For capturing the entire event, the event has to arrive within the interval, $t_{on} - t_e$, which we call, the effective on-time of an intermittent node. Therefore, the effective availability of a CIS of N nodes is the joined effective on-

times of the underlying intermittent nodes, which can be modeled as,

$$A_v(N) = A_v(N-1) + (1 - A_v(N-1)) \frac{t_{on} - t_e}{t_p}. \tag{5.2}$$

Event-driven Sensing An intermittent sensor has a limited energy budget per power cycle. When it is tasked with a polling-based sensing activity, its energy consumption switches between two levels: zero when charging and maximum when sensing. However, in event-based sensing, a node puts its microcontroller into low-power mode and waits (or listens) for an external event to wake up the microcontroller. For example, in our prototype, a voice-controlled command recognizer, we exploit the microphone's wake-on-sound feature to send an interrupt to the microcontroller, which will then start recording the sound samples from the microphone. This wake-on-event style of operation is important as the minimal energy consumption during sleep significantly prolongs the period during which an event can be handled (for example, our prototype consumes 7 times less energy during sleep compared to being active). To model the effective CIS availability when it is tasked with event-based sensing, the change in energy consumption between the sleep and active mode must be taken into account. Since the event itself times when the node changes its energy consumption, we can model the effective availability as

$$A_v(N) = A_v(N-1) + (1 - A_v(N-1)) \frac{t_s - \left(t_e \times \frac{p_a}{p_s}\right)}{t_{sp}}, \tag{5.3}$$

where t_s is the expected sleep time of the CIS's nodes, $t_{sp} := t_s + t_{off}$, and p_a and p_s are the power consumption in active and sleeping mode, respectively. Notice that, there is a subtle point about (5.3) as when an event arrives the node wakes up, consuming more energy. Therefore, its uptime shrinks. For simplicity, we modeled this effect by extending the event time with the same factor. This is sufficient to say if the event will be fully captured or not (effective availability).

SIMULATION

To perform a first sanity check on our models, we developed a Python-based simulator. For each data point presented in Figure 5.6, the simulator generated 10^5 power cycles of a CIS of 10 nodes. We ranged the duty cycles from 10% to 50%, while keeping the length of the event fixed at 3% of the power cycle length, t_p. The on-times and event arrivals were uniformly distributed over the power cycles. The results clearly confirm our models and support our argument about the distinction between CIS's availability and effective availability (notice that the percentage of captured events matches the effective availability). The importance of this distinction—availability versus effective availability—is a function of the value $\frac{t_e}{t_{on}}$; observe the difference between availability and effective availability when nodes' duty cycle is 10% (large effect) and 50% (negligible effect).

5.2.2. ENVIRONMENT

Ambient energy controls the availability of a CIS's nodes. Consequently, it also controls their collective response to external events. When it rises, it extends nodes' on-times that may cause nodes' power cycles to be synchronized on the arrival of some external

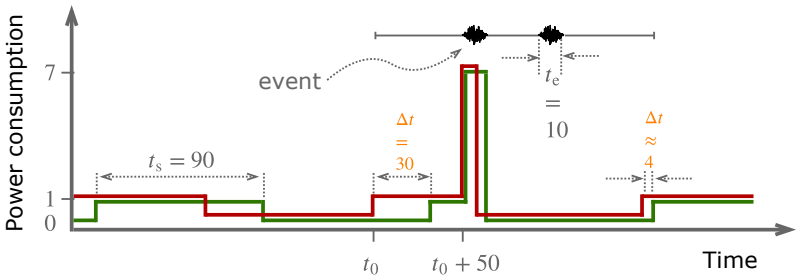


Figure 5.7: Capturing events may lead to power cycles synchronization of nodes that were in low-power mode. In particular, if some of the nodes power down while capturing the event, then this implies that these nodes slept (prior to the event) longer than the nodes that capture the event. This means that the nodes that have died spent their energy slower (they stayed longer in sleep mode); therefore, their overall uptime is longer than the uptime of the node the captured the event. In other words, the nodes that woke up earlier have stated longer on, and therefore, the next power cycles are more synchronized, see the Δt before and after the event. The red and green line graphs represent the power consumption of nodes R and G when they are charging, sleeping and capturing events.

events, compromising the CIS's overall availability. To overcome this problem the CIS's nodes must be power-state aware and able to estimate the number of active nodes in the CIS.

POWER STATES

A CIS can experience a wide range of ambient power intensities. For example, a solar-powered CIS may harvest no energy at night, modest energy from artificial light, and abundant energy from direct sunlight. Generally, we can identify four different CIS powering states:

- *Targeted power state*:- These are the powering conditions that a CIS is designed for. In these conditions, the CIS should work intermittently and have sufficiently randomized power cycles to uniformly distribute its intermittent nodes on-times and meet the desired availability (Figure 5.4). In general, the targeted powering conditions should be near worst energy harvesting conditions to ensure that the system is properly functioning for the majority of the time.

- *Under-targeted power state*—Ultimately, the ambient energy is an uncontrollable power source, and it is not hard to imagine scenarios where a CIS will be under-powered or even comes to a complete and long power down (for example, a solar CIS will come to a perpetual power down in darkness). In general, for under-targeted energy conditions, the CIS behavior can be considered as undefined.

- *Hibernating power state*:- In event-driven sensing, nodes sleep in low-power mode waiting for an event to wake them up. This mode extends CIS's nodes' on-times and makes them overlap significantly. Moreover, the on-times overlap even more, when ambient energy level rises (favorable energy conditions). If an event arrives in such conditions, it will wake up many nodes, causing them to consume their

buffered energy faster. Some of these nodes will power down before capturing the entire event, while others survive. This difference in how much energy is spent in active and sleep mode causes these nodes to tend to synchronize their power cycles after the event. To understand this let us analyze the example presented in Figure 5.7. The figure shows the power traces of two nodes. the nodes consume 7 times more power in active than in sleep mode (our prototype has a similar power consumption ratio, Table 5.3). Further, it shows that a node sustains the low-power mode for 90 units of time, $t_s = 90$; therefore, the maximum buffered energy can be calculated as $E_{buf} = t_s \times p_s$, where p_s is a node's power consumption in sleep mode. If we focus on the power cycle with the first event, then we see that node R powers up at t_0, and it remains in low-power mode for 50 units of time, whereas node G spends only 20 units of time before the event arrives. Now, we can calculate when these two nodes will power down and compare the difference, Δt, before and after the event. A node will turn off when the buffered energy is depleted. This can be expressed as

$$E_{buf} = t_{se} \times p_s + t_{on} \times p_a,$$

where t_{se} is the sleep time of the power cycle that an event arrives in, t_{on} is a node's on-time and p_a is the power consumption in active mode, which can be expressed as $p_a = \frac{\delta}{p_s}$. t_{se} and δ are given and p_s can be eliminated; thus to find when the nodes will power down we need to find t_{on} for both nodes. By substituting the given values we find t_{on} to be 10 and 5.7 units of time for the G and R node, respectively. Therefore, Δt becomes 4.3 while it was 30 before the event (notice, $\frac{30}{7} \approx 4.3$). In general, nodes that die while capturing the event must have started their power cycles *before* the nodes that capture the event. Further, the uptime of the died-while-capturing nodes is *longer* than the nodes that capture the event because they spend less time in active mode. Therefore, the difference, Δt, between the power cycles of a died-while-capturing node and a node the successfully captures the event becomes smaller. This difference shrinks by the factor δ or $\frac{p_a}{p_s}$. When the events arrive in burst this becomes a significant problem, as the CIS will capture multiple copies of the first event, while missing the subsequent ones.

- *Continuous power state*:- Under direct mid-noon sun a tiny solar panel may provide sufficient power to run a sensor node continuously. In such conditions, a CIS's node will be available and able to sense continuously. Therefore, the job of a single node will be repeated N times, and instead of sending a single message to a sink—to push the data to the Internet—N identical messages will be sent. These messages will collide as they are sent at about the same time, causing the information to be lost; if they arrive, however, they—except the first one—will waste energy of the sink as they carry the same information.

The inefficiencies highlighted in the hibernating and continuous power states can be mitigated by enforcing randomization on the response of intermittent nodes: when a node is woken up by an external event it responds to that event with a certain probability. However, if the randomized response is enforced all the time, then the CIS will have a lower probability of catching events during the targeted energy conditions state. Therefore, the CIS has to distinguish between the targeted and above-targeted energy condi-

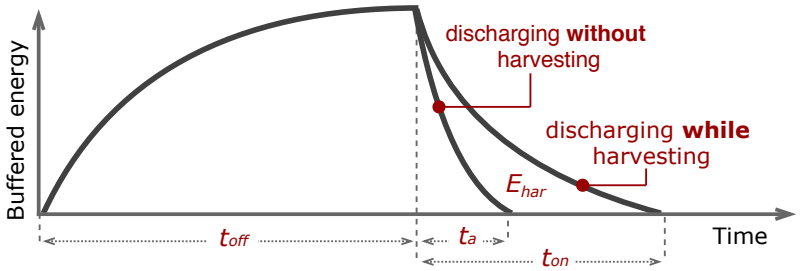


Figure 5.8: The difference in the time of discharging the energy buffer—a node's on-time—when an energy-harvesting device is allowed to charge while operating, and when it is not allowed.

Algorithm 4 off-time estimation

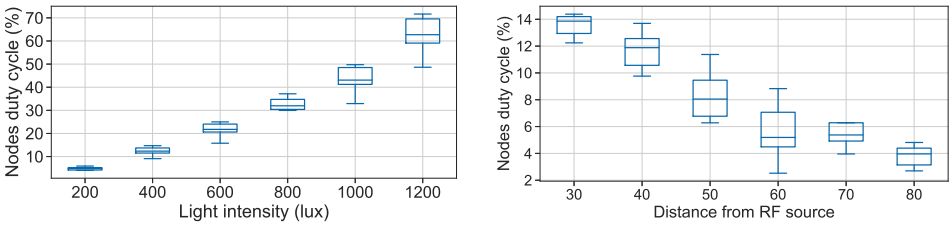
```
1: R_cntr++                                          ▷ reboot counter
2: E_buf                                             ▷ Size of energy buffer
3: t_a                                               ▷ time of discharging E_buf at load a, no harvesting
4: X_cy                                              ▷ time every X power cycles
5: ▷ Code executed on each X power cycles
6: if (R_cntr == X_cy) then
7:     f_LOAD(a)                                     ▷ set node load to a
8:     t_on ← TIME()                                 ▷ measure time until power down
9: ▷ Code executed on each X + 1 power cycles
10: if (R_cntr == X_cy + 1) then
11:    Δt = t_on − t_a                               ▷ time difference due to charging
12:    E_har ← E_buf × t_a/Δt                        ▷ harvested energy
13:    P_in ← E_har ÷ t_on                           ▷ incoming power
14:    t_off ← E_buf ÷ P_in
15:    R_cntr = 0
```

tions and randomize its response only during the hibernating and continuous power states.

Furthermore, responding with a constant probability during the above-targeted energy conditions is inefficient, as the number of active nodes is a function of the total number of intermittent nodes and the power intensity at that time. Therefore, efficient randomization requires intermittent nodes to estimate the number of active nodes and respond proportionally. Our proposed algorithm for estimating the number of active nodes depends on the nodes' ability to measure their on-times and off-times.

INTERMITTENT TIMING

Timing is a key building block of sensing systems. It is, however, missing on intermittent nodes unless an additional dedicated (RC-based) timer is included [35]. Here we propose an alternative that does not require additional hardware. This alternative does not only enable time estimation but also ambient energy richness, which is very important for estimating the number of a node's active neighbors. But, *how a node can time its own on/off cycle?*



(a) Light (b) RF

Figure 5.9: The average duty cycles of 8 solar-powered and 6 RF-powered intermittent nodes for different ambient energy sources and energy intensities. In general, the average duty cycle of a node is a good indicator of the average duty cycle of the other CIS's nodes.

Intermittent nodes fail abruptly; therefore, a persistent timer is needed to measure a node's on-time. A simple way to emulate persistent timer is by using a persistent counter, or sampling the volatile built-in timers of the microcontroller and save the obtained values in the non-volatile memory. To estimate the off-time, t_{off} in Figure 5.8, a node needs to determine the incoming power (harvesting rate). The average harvesting rate can be induced from the on-time as follows. The node measures its on-time while harvesting, see t_{on} in Figure 5.8, and compares it to the time required to drain the energy buffer *without* charging, see t_{a} in Figure 5.8. The additional on-time, Δt, is the result of the energy accumulated while executing. If t_{on} and t_{a} are measured on the same load—thus, they have the same power consumption—then the amount of the energy harvested while the device is on can be calculated as in Algorithm 4, Line 12. And, the average input power can be found as in Line 13 that, in turn, enables the node to estimate its own t_{off} (Line 14). Since calculating the off-time requires constant load, the sensor cannot run arbitrary code during time measurement. Therefore, the sensor needs to sacrifice a certain percentage of its power cycles for measuring time (Line 1-8). Once the on-time and off-time are found the node's power cycle for load a is determined.

Notice that, when the harvested power is very low the accuracy of inferring the charging time from the discharging degrades. However, for the CIS this is not a serious problem as the intermittent nodes need to randomize their response to events only in favorable energy conditions.

ALIVE NODES ESTIMATION

To estimate the number of active nodes, a CIS's node needs to determine the following information: (i) the total number of nodes in its CIS, which is a typically constant value that can be loaded to the device memory; (ii) the on-times distribution, which is uniform in our case; and (iii) its own average $\overline{t_{\text{on}}}$ and $\overline{t_{\text{off}}}$.

Since, we assume that a CIS's nodes have the same energy buffers and are in the vicinity of each other (thus, they are exposed to the same energy conditions) then their duty cycles should approach the same value. Figure 5.9 shows the average duty cycles of the nodes of a solar- and RF-powered CISs. In general, we can conclude that a node's average duty cycle is a good estimator of other CIS's nodes' duty cycles. Now, a node can estimate the maximum time span, t_{max}, of its CIS, which is the total duration of the

Table 5.1: Measuring intermittent nodes overlapping of a CIS of 8 intermittent nodes for different light intensities. $N_{\textbf{active}}$ is the expected number of active nodes and σ is the standard deviation.

Light (lux)	**On/off cycle (%)**	$N_{\textbf{active}}$	σ
300	8	1.01	0.85
500	17	1.63	0.98
800	31	2.88	1.50
1200	62	5.05	1.08

nodes' on-times when they are aligned next to each other, as follows

$$t_{\max} = N \times \overline{t_{\text{on}}}. \tag{5.4}$$

Then, from (5.1), the node calculates the CIS availability, $A_{\text{v}}(N)$. As we argued in Section 5.2.1, nodes on-times are uniformly distributed; therefore, the overlapping on-time is also uniformly distributed. As such, a node can calculate the average number of active intermittent nodes, N_{active}, using,

$$N_{\text{active}} = \frac{t_{\max}}{\overline{t_{\text{p}}} \times A_{\text{v}}(N)}. \tag{5.5}$$

RESPONSE RANDOMIZATION FACTOR

Once a node has estimated the number of active neighbors, N_{active}, it can use the following formula to determine the response probability,

$$P_{\text{resp}} = \begin{cases} \frac{N_{\text{resp}}}{N_{\text{active}}}, & \text{if } \frac{N_{\text{resp}}}{N_{\text{active}}} < 1, \\ 1, & \text{otherwise,} \end{cases} \tag{5.6}$$

where N_{resp} is a system parameter that reflects the desired redundancy factor required by an application.

Table 5.1 shows the average number of active nodes of an 8-nodes CIS for different light intensities. These measurements provide a sanity check of (5.5). For example, at 1200 lux an individual node of our CIS has a duty cycle of $\approx 62\%$, i.e., it is on average $0.62\, t_{\text{p}}$ operating. If we multiply that by the number of nodes (5.4) we get about $5\, t_{\text{p}}$. Figure 5.4 indicates that a CIS with eight nodes of duty cycles above 50% has near 100% availability. From (5.5), we find that the expected number of clustered nodes is 5, confirmed by the measurements presented in Table 5.1.

5.3. PROTOTYPE: COALESCED INTERMITTENT COMMAND RECOGNIZER

The coalesced intermittent command recognizer (CICR) is a prototype of the Coalesced Intermittent Sensor. The CICR consists of eight battery-less intermittent nodes. Each node is capable of performing isolated word recognition.

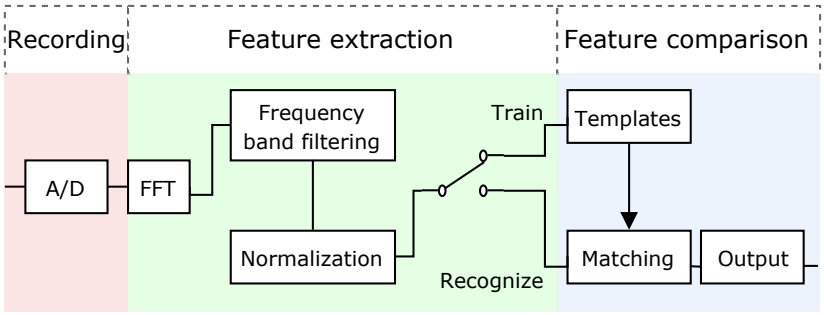


Figure 5.10: Coalesced Intermittent Command Recognizer: an instant of a Coalesced Intermittent Sensor. CICR features a power failure-immune word recognition algorithm. First a word is recorded. Then, its spectral features are extracted. The resulting features vector is compared against previously-stored words' templates for recognition. The comparison is done using a liner distance matching algorithm

5.3.1. HARDWARE

A CICR node consists of three main parts: a microphone, a microcontroller, and a harvester. MSP430RF5994 [156], an ultra-low-power microcontroller, is used for data acquisition and processing. This microcontroller has a 16-bit RISC processor running on 1 MHz, 8 kB of SRAM (volatile), 256 kB of FRAM (non-volatile), and a 12-bit analog to digital converter (ADC). It also features a Low Energy Accelerator (LEA), which offloads the main CPU for specific operations, such as FFT. For recording we use the PMM-3738-VM1010-R piezoelectric MEMS microphone, which features Wake on Sound and ZeroPower listening technologies [159], allowing both the microcontroller and the microphone to sleep in a low-power mode until a sound wave is detected. The microcontroller and microphone are powered by a BQ25570 solar power harvester [147] connected to an IXYS SLMD121H04L solar cell [148] and a super-capacitor of 470 μF. For debugging we used the Saleae logic analyzer [135].

The power usage of a node differs according to its activity. When a node is waiting for a voice event, it is in low-power mode. Recording a voice event activates the microphone, ADC and microcontroller (maximum power consumption). Processing the recorded data requires only the microcontroller to be on. Table 5.3 lists a node's power consumption for each of these states (sleeping, recording, and processing), as measured with a Monsoon power monitor [160].

5.3.2. SOFTWARE

The CICR runs power interrupts immune command recognizer. The recognizer is capable of recognizing the isolated-word type of speech. The main parts of the recognizer are illustrated in Figure 5.10 and explained below:

Data acquisition The *Wake-on-Sound* feature of the microphone triggers the data acquisition process once the energy level in the sound signal crosses a certain level. The ADC samples the output of the microphone at 8 kHz. This sampling rate is sufficient to

Table 5.2: Word testing set for CICR

on	off	stop	clear	load
go	pause	resume	edit	cancel

Table 5.3: Power usage of a CICR's node

State	**Current** (μA)	**Time** (ms)
Sleeping	64 ±20	—
Recording	423 ±20	285
Processing	282 ±20	600

cover most of the frequency range of the human voice. The recording length was set to 285 ms, which suffices to get all the acoustic features needed to recognize the words.

Feature extraction For word recognition, we adopted the method presented in [161]. Here, we briefly describe the algorithm for convenience. The CICR starts by dividing the signal into frames of 256 samples (≈ 33 milliseconds). Then, it computes a 256-point Fast Fourier Transform for each frame. The resulting feature vectors are normalized (by computing the binary logarithm of each entry of that vector) to reduce detection errors that result from differences in the amplitude of the speech input. These feature vectors are the basis for the word-identification process.

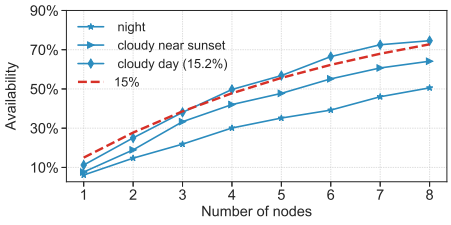
Feature matching Feature matching is achieved by computing the squared Euclidean distance between the normalized feature vectors of the recorded word and the feature vectors of the words stored during the training phase (templates, see Table 5.2). Once the recorded word has been compared to all template words, the template with the smallest distance to the recorded word is considered the correct word. However, if the smallest distance is bigger than a confidence threshold, then the CICR will return "undefined word".

We have experimented with two feature matching algorithms: the Linear Distance Matching (LDM) and Dynamic Time Warping (DTW) algorithm [162]. While LDM compares the feature vectors of two words successively, DTW looks for the minimum distance between the two vectors. In our implementation, the DTW was about 10 times slower than LDM, whereas the detection accuracy was comparable; therefore, we default our implementation to LDM.

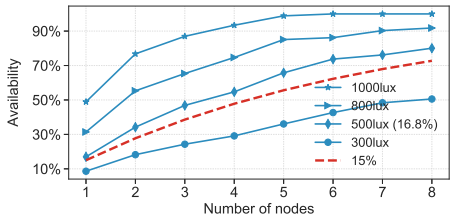
Power Failure Protection In order to preserve the progress state and to protect CICR data against randomly timed power failures, we split the recognition program into 19 atomic regions. We ensured that each of these regions requires less energy than what the energy buffer can provide with a single charge. The program state is checkpointed in the non-volatile memory (FRAM) on the transition between these regions. This prevents the program from falling back to its starting point (`main()`) after each power failure. Data in the non-volatile memory with Write-After-Read dependency is double-buffered to ensure data integrity when the power supply is interrupted.

Code profiling The entire command recognition software was written in C. The total program consists of 973 lines of code, excluding the FFT function, which is imported from the Texas Instrument DSP library. The memory footprint on the microcontroller is 20,064 B of FRAM and 1,134 B of SRAM.

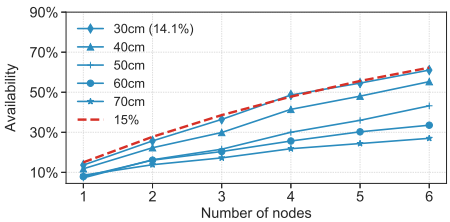
5.4. EVALUATION



(a) The CIS is powered by uncontrollable light sources—artificial light (night) and sunlight (day).



(b) The CIS is powered by a controllable array of LEDs.



(c) The CIS is powered by an RF reader [78] located 30-70 cm away from the RF tags (WISPs [79]).

Figure 5.11: The Coalesced Intermittent Sensor's availability for differed numbers of intermittent nodes and energy sources. The modeled availability—represented by the dashed lines when the average node duty cycle is 15%—approximates the measured availability with high accuracy.

To evaluate the performance (availability) of the Coalesced Intermittent Sensor, we conducted several experiments in different energy conditions and with different event arrivals patterns.

5.4.1. AVAILABILITY

Irrespective of the energy source (RF or light) we showed in Figure 5.5 that the power cycles of a CIS's nodes are different, which leads to a uniform distribution of their on-times, as we argued in Section 5.2.1. We captured the expected joined availability of these nodes in Model 5.1. Here, we validate the model by comparing the modeled availability of a CIS against data captured with different hardware (solar- and RF-powered nodes) in different scenarios.

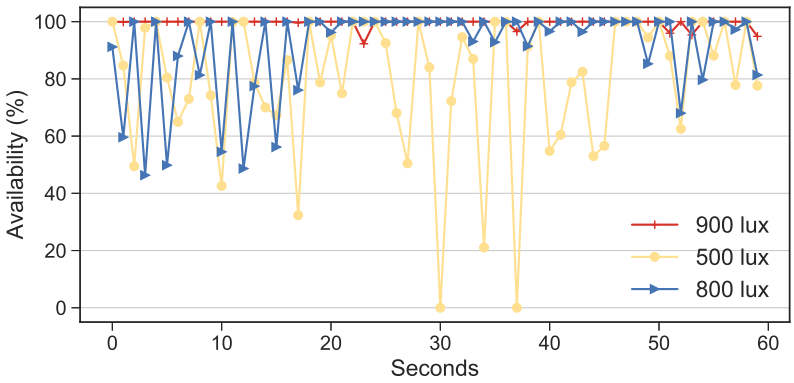


Figure 5.12: CIS availability observed with a 5 seconds time window.

Figure 5.11 shows the availability of three CISs when they are powered by sunlight, artificial light, and RF and for a different number of intermittent nodes. The results clearly confirm our expectation: when the power cycles are slightly different, the on-times approach uniformly distributed. The results also validate our model; the dashed lines represent the modeled availability when nodes' duty cycle is 15%.

AVAILABILITY ON A FINE SCALE

Since the nodes' on-times are in a constant shift relative to each other (Section 5.2.1), the collective availability of the CIS fluctuates when it is observed in a short time window. Figure 5.12 captures CIS availability on a time window of 5 seconds for thee different ambient energy conditions. In these experiments, the average power cycles of the CIS's nodes are (3,18), (3.9,12.3), and (4.3,11.5) seconds when ambient light intensity is 500, 800, and 900 lux, respectively. If we focus on the line graphs associated with 500 and 800 lux and compare the system availability within the interval [20, 50] seconds and the rest, we can observe that the CIS gradually alternates between low and high collective availability; nodes' on-times gradually transition from maximum to minimum separation and vice versa (Section 5.2.1). Notice that, when ambient light intensity was 800 lux the CIS collective availability transited from low to high to low, while this pattern happened to be reversed when light intensity was 500 lux. For the 900 lux the 8-node CIS achieved near-continuous 100% availability.

5.4.2. SENSING

EXPERIMENT SETUP

After validating our observation on different energy sources, we designed a testbed with controllable light intensity for clarity and reproducibility. To this end, we blocked uncontrollable light sources with a box of 60 × 40 × 40 cm. On the box ceiling, we attached a light strip of 2.5 m with 150 LEDs that can produce 15 different light intensities. On the bottom, a CICR of 8 intermittent nodes is placed (see Section 5.3.1 for hardware description).

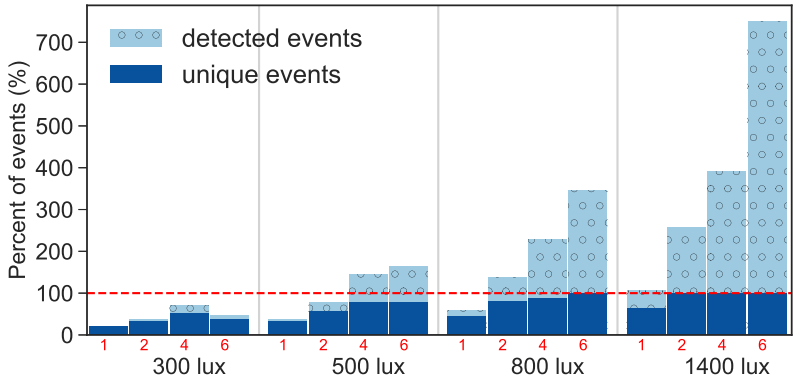


Figure 5.13: Duplicate and unique events captured by a coalesced intermittent command recognizer of eight solar-powered nodes. In general, the number of captured events increases in two cases: when light intensity rises and when inter-event arrival time increases. Red numbers indicate events arrival interval in seconds.

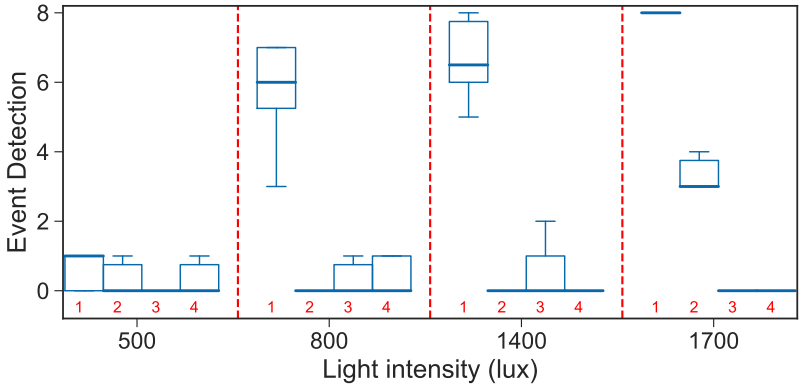


Figure 5.14: The CIS response to events arrive in bursts of 4 *without enabling response randomization*. We see that the majority of the nodes react to the first event in a burst and power down shortly after, missing other events in the burst. Red numbers indicate events indices in a burst.

The events in our experiments are spoken words (Table 5.2). Short events (see events classification in Section 5.2.1) are represented with individual words, while burst events are represented with phrases of a few words. We recorded different patterns of inter-event and inter-bust arriving time. We used a Bluetooth speaker [163] to replay a certain recording. The data were collected using a logic analyzer [135] and processed on a laptop running Ubuntu 16.04 LTS.

EVENTS DETECTION RATE

Here we experiment with the behavior of a CIS when events arrive individually or in bursts *without* enabling randomized response in favorable energy conditions.

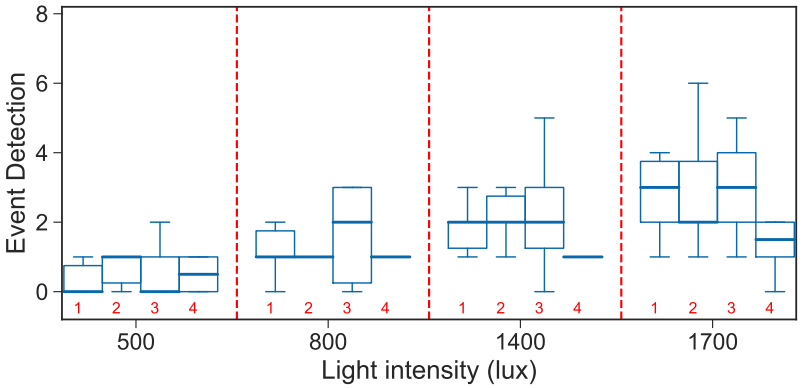


Figure 5.15: The CIS response to events arrive in bursts of 4 *when nodes randomize their response*. We see when response randomization is enabled the CIS capture the entire burst of events with high rates, while reducing the number of duplicated detections. Red numbers indicate events indices in a burst.

Individual events. Figure 5.13 shows the percentages of capturing duplicate and unique events when light intensity varies from 300 lux to 1400 lux and the inter-event arrival time ranges from 1 second to 6 seccond. For each experimental trial 20 words were played, resulting in a total of 240 playbacks.

Figure 5.13 clearly shows a positive correlation between light intensity and the number of detected events. In particular, the number of duplicate detections upsurges when light intensity increases, *demonstrating the overpowering problem* (Section 5.2.2). Moreover, increasing the inter-event arrival time also surges the number of duplicated events. The reason for this phenomenon is that when the time between events increases, the intermittent nodes get the chance to sleep longer in low-power mode, consuming less energy. Therefore, nodes' on-times expand, reducing their inherent randomization, which leads them to be in *hibernating power state* (Section 5.2.2).

Bursty events. Figure 5.14 shows the capturing behavior of a CIS when the events arrive in bursts. A burst of four events with one second between the individual events was fired every 20 seconds. Each burst was repeated 10 times and under four different light intensities. The nodes sleep in a low-power mode when they finish processing an event, waiting for the next one.

In general, we observe that in favorable energy conditions (above 500 lux) intermittent nodes react to the first event of a burst and power down shortly after, missing the rest of the burst. These results confirm our argument about the side effect of the *hibernating power state* of a CIS (Section 5.2.2). These results also demonstrate that the hibernating power problem happens on a wide range of power intensities, showing its significance. Next, we will show how randomizing the response can mitigate the problems generated when ambient energy exceeds the design point.

(lux, second)	(800,6)	(1400,4)	(1400,6)
randomization	205/432	236/675	223/493
no randomization	240/831	240/938	240/1802

Table 5.4: These results are presented in the following format *unique/total* detected events. A CIS's node responds with a probability of 65% in the first two scenarios, **(800,6)** and **(1400,4)**, and 30% for the last one. Randomizing the response reduces the number of duplicated events by 50% while losing only 7% of the unique events.

EVENTS DETECTION RATE WITH RANDOMIZATION

Here, we examine the effect of enabling artificial randomization on the CIS's response.

Individual events. Table 5.4 compares the number of detected events when the CICR's response is randomized and not randomized. When randomization is enabled, nodes respond to events with a probability of 65% for the scenario of (800 lux, 6 seconds) and (1400 lux, 4 seconds), and for the highest energy level and the longest inter-event arrival time the responding probability was set to 30%.

Table 5.4 shows that randomizing the response reduces duplicated events by an average of ≈50%, while only marginally lowers the number of the uniquely detected events (7% on average).

Bursty events. To enable a CIS to capture events arrive in bursts, the response probability for each events in a burst should be different. The CIS should respond with a minimum probability to the first event in a burst and gradually increase the response probability for the subsequent events in the burst (we assume that between the bursts the CIS resumes to its expected collective availability). This gradual increment to the responding probability is motivated by the observation that when a node captures an event it becomes unavailable for the subsequent ones in the burst. In this experiment, the nodes reacted with a probability of 40%, 50%, 70% and, 100% on the first, second, third and fourth event, respectively. Since the event distribution is known these probabilities were fixed during the development stage.

Figure 5.15 shows how randomizing the CIS response spreads the nodes' awake times, as compared to Figure 5.14, and enables the CIS to capture the entire burst with a high probability, i.e., above 85%. Additionally, we also observe a positive impact of randomizing the response when the system is under-powered (500 lux).

5.5. CONCLUSION AND FUTURE WORK

This chapter addresses the availability problem of intermittent sensors that fail to capture (and process) events while charging their energy buffer. As the power to drive a node is much higher than what can be harvested from ambient sources, the chance of capturing an event can be as low as just 8% (sunlight) and 4% (RF) (cf. the duty cycles reported in Figure 5.9). To address this problem of missing most events we presented the Coalesced Intermittent Sensor (CIS), which is the abstraction of a group of intermittently-powered sensors, whose collective duty cycle (on-time) can approach the desired 100% availability. The inherent differences in the powering subsystem of intermittent sensors

result in (slight) differences in the sensor nodes' power cycles causing the nodes' on-times to be uniformly distributed. This implies that simply selecting the right number of nodes is all that is required. To this end, we have modeled the (effective) availability of a CIS and validated its accuracy against data collected on real hardware.

Experimentation with an 8-node prototype CIS, a basic voice-control application recognizing up to 4-word commands, showed that the inherent randomization in the power cycles can easily be disrupted. In case the ambient power exceeds the (worst-case) design point and nodes employ an efficient wake-on-event sleep mode, all nodes wake-up on the same (rare) event. If the energy buffer is small then they all enter the charging state at approximately the same time (unwanted synchronization) and subsequent events (words) will be missed (compromising availability). To counter this unwanted behavior, we proposed to use a probabilistic approach in which the number of active neighbors is determined and nodes respond proportionally to an event. This approach was shown to be effective for our prototype, capturing burst events with above 85% detection accuracy.

Similar to battery-powered sensors, intermittent sensors will need to communicate their data to the external world. Backscattering is a promising candidate to drive the communication between energy-harvesting devices as it does not require them to emit signals. However, sensor-to-sensor backscatter networks suffer from dead spots that prevent a senor from reaching all its potential neighbors. This limitation is investigated in the next chapter.

6

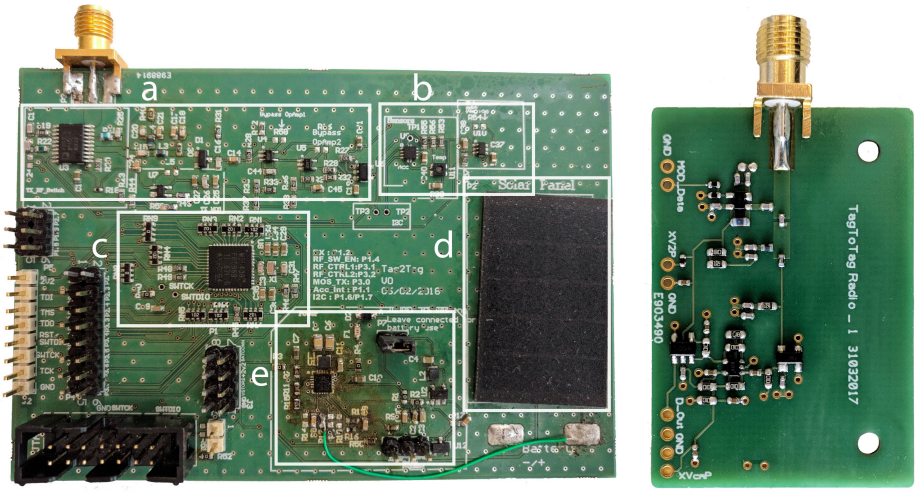
MULTI-HOP BACKSCATTERING FOR TAG-TO-TAG NETWORKS

6.1. INTRODUCTION

Energy-harvesting battery-less sensors need to communicate their sensory data to other layers in the system to take proper actions. However, their extremely limited energy budget calls for energy-efficient communication. Backscatter technology provides ultra-low-power communication [165]. It enables sensors to communicate their data by inducing changes to ambient signals, instead of emitting signals. Despite its energy efficiency, however, sensor-to-senor (or tag-to-tag) backscattering has a number of drawbacks. First, backscatter signals are weak as compared to their carrier signals. Therefore, they have short propagation ranges. Second, due to the dis-locality between the carrier signal generator (e.g., an RF reader) and the information modulator (a backscattering tag), the communication links are non-symmetric, which restricts the tag-to-tag range of communication to the weaker link [37]. Finally, this dis-locality between the energy and information sources causes permanent dead spots in tag-to-tag backscatter networks, which further limits the communication range [37].

In this chapter, we study the effect of multi-hopping on the range of backscatter networks and the dead spots in them. To this end, we developed a discrete component-based backscatter T2T transceiver and a communication protocol suite composed of (i) flooding-based link control tailored towards backscatter transmission, and (ii) low-power listening MAC. Our MAC design is based on a new insight that backscatter reception is *more energy costly* than transmission. We show that the multi-hop benefits the backward T2T link (tag far from the exciter sending to the tag close to the exciter) much more than the forward link. Furthermore, we show experimentally that the decode-and-relay T2T network improves the robustness of single-hop T2T transmission as good as state-of-the-art phase-shifted message repetition mechanism, while (i) being capable of

Parts of this chapter have been published in IEEE INFOCOM'19, Paris, France [164].



(a) Standalone backscatter T2T tag

(b) T2T transceiver

Figure 6.1: Two prototypes of backscatter T2T tag. Figure 6.1a: Integrated version (dimensions: 6.9 cm×10.1 cm) with main sections marked as **a**: T2T transceiver, **b**: sensors, **c**: digital section with TI MSP430-family MCU [80], **d**: solar panel, **e**: power and energy harvesting. Figure 6.1b: T2T backscatter transceiver board only (dimensions: 3 cm×4.4 cm) as an add-on for embedded platforms.

extending the T2T communication range by a *factor of two*, already at four hops transmission range, and (ii) enabling connecting T2T networks served by separate exciters.

6.2. MULTI-HOP BACKSCATTER T2T NETWORK ANALYSIS

We now proceed with a set of theoretical results that show the benefit of multi-hop T2T networks. This analysis will provide fundamental insights that will be observed in real T2T network implementation, presented in Section 6.6.

6.2.1. BACKSCATTER T2T MODEL DEFINITION

Let us define a flat, square area of length S_a with one exciter E at position (x_E, y_E) and N static tags located at uniform random positions within the area. The exciter transmits a non-modulated signal of wavelength λ_c and power P_E, which tags can backscatter to communicate with other tags with reflection coefficients k_0, k_1 for symbols 0, 1, respectively. The receiver sensitivity threshold for all tags is P_s and the available power to backscatter at tag n is [37, eq. (2)]

$$P_n = P_E G_E(\theta_{E,n}) G \lambda_c^2 \left(4\pi d_{E,n}\right)^{-2} \tag{6.1}$$

where $d_{k,l}$ is the distance from device k to l (tag or exciter), $G_E(\theta_{E,n})$ is the antenna gain of the exciter in the angular direction to tag n, $\theta_{E,n}$, and G the antenna gain of the tags.

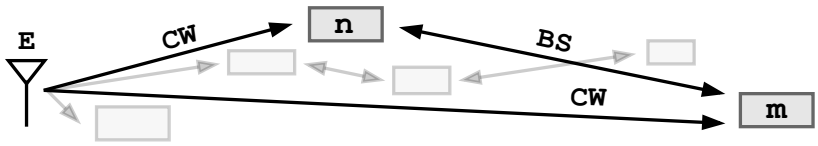


Figure 6.2: Illustration for received power analysis of T2T network; CW: carrier wave, E: exciter, BS: backscatter.

Then, the power received at tag m from tag n is [37, eq. (6)]

$$P_{n,m} = P_n \left(k_0 G \lambda_c\right)^2 \left(4\pi d_{n,m}\right)^{-2}. \tag{6.2}$$

For simplicity, we consider only line of sight, free-space propagation with no fading or noise, as our goal is to provide the simplest bounds on network performance. As $k_0 < k_1$, (6.2) serves as a lower bound to the received power.

The T2T network is modeled as a weighted graph with a set of nodes $\{1, 2, \ldots, N\}$ and a set of directional links $\{L_{n,m}\}$. The weight of a link $L_{n,m}$ from tag n to tag m is defined as $L_{n,m} = P_{n,m}$ iff $P_{n,m} \geq P_s$ and $L_{n,m} = \varnothing$, otherwise. A single-hop T2T network is connected if $L_{n,m} \neq \varnothing, \forall n, m$. Conversely, a multi-hop T2T network is connected if there exists a path $\forall n, m$ node pairs.

6.2.2. ANALYSIS OF BACKSCATTER T2T NETWORK

NON-SYMMETRIC T2T LINKS

A tag's available power depends on its distance from the exciter. This implies the following simple observation.

Observation 1. *The ratio of received power between tags n and m in the forward link (from n to m, Figure 6.2) over the power received in the backward link (from m to n) is quadratically proportional to* $d_{E,m}/d_{E,n}$.

Proof. Directly from (6.2) we can write

$$\frac{P_{n,m}}{P_{m,n}} = \frac{G_E(\theta_{E,n})}{G_E(\theta_{E,m})} \left(\frac{d_{E,m}}{d_{E,n}}\right)^2. \tag{6.3}$$

This implies that in the useful case where tags are spread away from each other, each T2T link is non-symmetric, with much higher reception probability at the forward link (i.e., a link from the tag closest to the exciter to a tag further away), meaning that $P_{n,m}/P_{m,n} \gg 1$. □

MULTI-HOP RANGE

Next, let us analyze the range capabilities of T2T multi-hop networks in two ways: comparing received power with single-hop, and computing the maximum distance such networks can cover.

Observation 2. *Considering tags 1 and N (Figure 6.3), the received power from the transmitting tag using multi-hop is always greater than that using single-hop in the* backward link *(from the most distant tag from the exciter to the tag closest to exciter).*

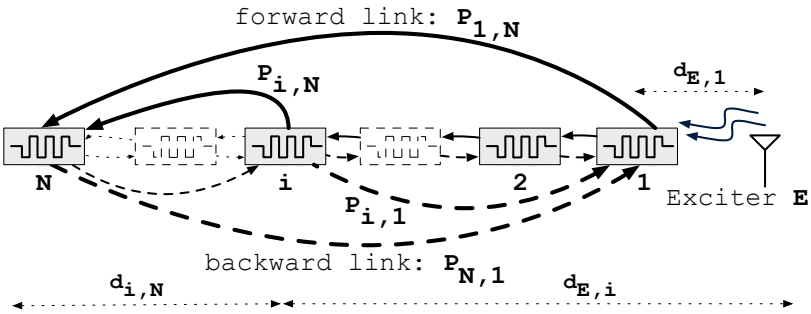


Figure 6.3: Multi-hop T2T received power analysis scenario.

Proof. Consider the linear topology depicted in Figure 6.3. With a little abuse of notation, let $d_{E,1} = d_1$ and $d_{k-1,k} = d_k, \forall k \in [2, N]$. As shown in this figure, the received power in the backward link is $P_{N,1}$ for single-hop and $P_{i,1}$ for multi-hop, which depends on the tag i over which the last hop is performed. For presentation compactness, define $\ell_a^b \triangleq \sum_{k=a}^{b} d_k$ that denotes the length of the path between tag a and tag b on the line topology. Using (6.2) again we can compute the power ratio

$$\frac{P_{i,1}}{P_{N,1}} = \frac{P_i (d_{N,1})^2}{P_N (d_{i,1})^2} = \left(\frac{d_{E,N} d_{N,1}}{d_{E,i} d_{i,1}} \right)^2 = \left(\frac{\ell_1^N \ell_2^N}{\ell_1^i \ell_2^i} \right)^2 . \tag{6.4}$$

Note that (6.4) greater than one, as $i \in (1, N)$. □

Corollary 1. *Assume that tags are equally spaced on the line topology; i.e.,* $\forall k \in [2, N]$: $d_k = d_1 = d$. *Then, node* i *that maximizes the received power in the backward link is* 2.

Proof. For the backward link (N to 1)

$$\arg\max_i \frac{P_{i,1}}{P_{N,1}}\Big|_{d_k = d} = \arg\max_i \frac{(N(N-1))^2}{(i(i-1))^2} = 2 \tag{6.5}$$

□

Let us now analyze the network's maximum range.

Observation 3. *Consider the multi-hop network of* N *tags placed on a line. The optimal value of the distance between any two tags* i *and* $i-1$ *that maximizes range and ensures communication in both ways is given by*

$$d_i^* = \frac{1}{2} \left(\sqrt{\left(\ell_1^{i-1}\right)^2 + 4\epsilon} - \ell_1^{i-1} \right) \tag{6.6}$$

with $\epsilon \triangleq \lambda^2/(4\pi)^2 G k_0 \sqrt{P_E G_E(\theta_{E,i}) G / P_s}$.

Proof. As shown in Observation 1, the forward link (from $i-1$ to i) is less costly in power. Then, it suffices to guarantee communication in the backward link (from i to $i-1$) by

Table 6.1: Backscatter T2T Network Efficiency Analysis Results; 'MHR': multi-hop range, 'SHR': single-hop range

		Case A: phase shifting	**Case B: multi-hop flooding**
Known topology	MHR	$E[m] = 2$ $E[t] = 2t_f$ $\Pr(s) = 0$	$E[m] = H$ $E[t] = H(t_f + t_p) - t_p$ $\Pr(s) = 1$
	SHR	$E[m] = 2$ $E[t] = 2t_f$ $\Pr(s) = 1$	$E[m] = 1$ $E[t] = t_f$ $\Pr(s) = 1$
Unknown topology	MHR	$E[m] = 2$ $E[t] = 2t_f$ $\Pr(s) = 0$	$E[m] = M + 1$ $E[t] = M\left(t_f + t_p\right) + t_f$ $\Pr(s) = 1 - p_c^{M+1}$
	SHR	$E[m] = 2$ $E[t] = 2t_f$ $\Pr(s) = 1$	$E[m] = M + 1$ $E[t] = M\left(t_f + t_p\right) + t_f$ $\Pr(s) = 1 - p_c^{M+1}$

ensuring $P_{i,i-1} = P_s$ so that the received power is greater than the receiver sensitivity threshold. Therefore, it follows

$$d_{E,i} d_{i-1,i} - \epsilon = d_i^2 + \ell_1^{i-1} d_i - \epsilon = 0,$$

from which $d_i^* = d_i$ can be solved. □

As more tags are added, they are placed closer together and eventually d_i^* becomes small enough so that tags would be placed in the near field of their antenna. Therefore, we set a minimum inter-tag distance (equal to the Fraunhofer distance) as $d_{\min} = d_F = 2D^2/\lambda \leq d_i^*$ with D being the largest linear dimension of tag's antenna.

COMBATING PHASE CANCELLATION

Stemming from the fact that the energy source is dislocated from the T2T transmitter, it is possible that the backscatter signal and the un-modulated exciter signal arrive at a receiver with opposite phases and interfere destructively. This phenomenon is called *phase cancellation* [37] and depends on the phase difference of arrival $\theta_{d,n}$: the difference of the phases of the signals arriving at T2T tag n coming from the exciter and the transmitter. Phase cancellation effectively creates blind energy spots in the network and happens when [37, Eq. (18)]

$$\theta_{d,n} = \theta_c = \cos^{-1}\left(-\frac{(k_0 + k_1) d_{E,m} \lambda_c G_n}{8\pi d_{E,n} d_{n,m}}\right). \tag{6.7}$$

The state-of-the-art solution to combat phase cancellation in T2T networks consists of sending every packet twice with a phase offset between them to ensure correct reception [37, 123]. We will assess the performance of the network when the phase-shift technique is applied and when our multi-hop protocol is used.

Analysis Let us look at the average number of messages $E[m]$ and average transmission time $E[t]$ it takes to deliver a frame for any source-destination T2T pair, as well as the probability of successfully doing so, $\Pr(s)$. Then, let us consider two network typologies: (i) a *known network topology* in which all nodes have complete information of the

Table 6.2: Realistic model of T2T network used in numerical example

Symbol	Value	Units	Description
f_c	868	MHz	Center frequency
$(k_0; k_1)$	(0.4; 0.9)	—	Reflection coefficients for 0 and for 1
$(P_E; P_s)$	(33; -50)	dBm	Output power of exciter, sensitivity of tags
$(G; G_E)$	(0; 4)	dBi	Antenna gain: tags; exciter
$(\theta_E; \theta_c)$	(-45; 40)	°	Exciter antenna beam direction; beam width
(x_E, y_E)	(0; 3)	m	Cartesian coordinates of exciter
S_a	30	m	Side of the square area of the T2T network
d_1	3	m	Distance from exciter to tag 1 (in Figure 6.4b)

network connections, and (ii) an *unknown topology*. These two cases are, in turn, split into cases where the source and the destination tags are within single-hop range or not (denoted as 'SHR' and 'MHR', respectively).

Table 6.1 collects all combinations for cases A and B. Case A is the single-hop phase shifting solution—repeating the same message twice with 90°-shifted phase, as proposed in [37, 123] and case B is the multi-hop protocol. When network topology is known the optimum path can be computed, which also enables avoiding links that are down due to phase cancellation effect (which we assume to happen with probability p_c) by switching the phase of the transmitted frame. Furthermore, the minimum hop count, H, is reached and $E[t]$ is minimized, which is composed of the time-of-flight of a frame, t_f, and the processing time taken by a node to forward a frame, t_p. In the case of unknown topology, we assume an average number of relay nodes M which are able to forward the message and reach the destination.

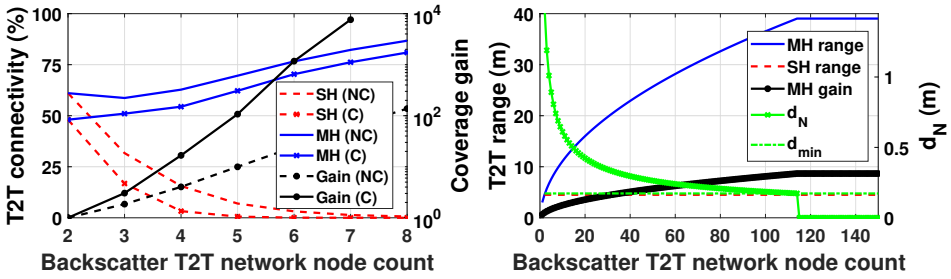
Result Comparing cases A and B, Table 6.1, we see that our solution, i.e., Case B, improves network utilization or communication success probability in most cases. However, if source and destination happen to be in range at unknown topology, it is possible that the usage of the T2T network is increased and/or the success rate is reduced depending on M and p_c.

6.2.3. BACKSCATTER T2T NETWORK: NUMERICAL RESULTS

We illustrate core analytical results with a numerical example. We simulate an instance of a backscatter T2T network with the parameters given in Table 6.2. Simulation code [166] is written in Matlab. We generate each instance of a T2T network by randomly placing nodes on a square of side S_a and checking connectivity at every iteration. Each simulation point is an average of 10000 runs. The results are given in Figure 6.4.

Result 1—*Multi-hop coverage.* We define coverage when *all* T2T tags are connected with each other. Backscatter T2T single-hop coverage does rapidly decrease with N, see Figure 6.4a, and more so when accounting for phase cancellation, improving the multi-hop gain. Also, the phase cancellation effect becomes less relevant with a denser T2T network.

Result 2—*Maximum multi-hop range.* Figure 6.4b presents the maximum communication range as a function of the number of nodes achieved by a one-dimensional backscatter T2T network while ensuring two-way communication. The maximum range was computed according to the analysis presented in Observation 3. When $d_N < d_{\min}$,



(a) T2T multi-hop coverage (b) T2T maximum multi-hop range

Figure 6.4: Backscatter T2T network numerical example with simulation parameters listed in Table 6.2. Figure 6.4a: multi-hop versus single-hop full connectivity (probability that *all* nodes are connected with each other, either only directly, SH, or through hop, MH) with and without phase cancellation; Figure 6.4b: multi-hop maximum range in one-dimensional T2T network; *NC*: no phase cancellation, *C*: phase cancellation, *MH*: multi-hop, *SH*: single-hop.

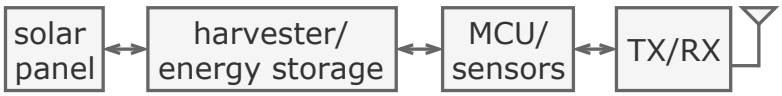


Figure 6.5: Backscatter T2T tag block diagram.

T2T network range can no longer be increased, but until then, the multi-hop range increases logarithmically, opposite to inter-tag distance d_N.

Remark:—*Limitation of T2T network.* The main bottleneck of T2T network is the small coverage, caused by the low signal detection threshold of the tags, P_s. Precisely because of this, traditional tag-to-reader networks would outperform a T2T counterpart in the range of communication.

6.3. BACKSCATTER T2T TAG DESIGN: HARDWARE

We proceed with the description of backscatter T2T tag architecture, see Figure 6.5. This section will introduce the T2T tag hardware, while Section 6.4 will introduce the supporting carrier generator and design choices on the tags deployment.

6.3.1. BACKSCATTER T2T TAG: BACKSCATTER TRANSCEIVER

Our T2T backscatter transceiver is designed with energy consumption reduction in mind. It, therefore, avoids all energy-hungry components such as ADCs or multi-stage power amplifiers. Its complete hardware design, introducing novel adaptations with respect to [17, Figure 6], [124, Figure 4], [167, Figure 3] is presented in Figure 6.6 and discussed stage-by-stage below. We note that to design and simulate the transceiver's (i) RF (including on-PCB antenna matching circuit) and (ii) the low-frequency part (including filter and amplifier parameters), ADS version 2012 [168] and OrCAD Capture version 17 [169] software suites were used.

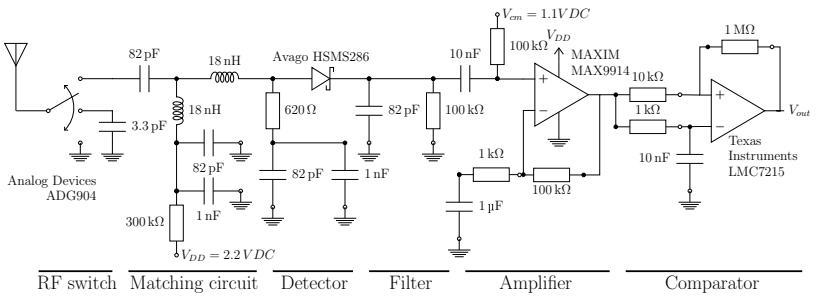


Figure 6.6: Backscatter T2T transceiver; its implementation is shown in Figure 6.1b.

ANTENNA MATCHING

The transceiver is preceded by 50 Ω antenna matching circuit tuned at 868 MHz center frequency, i.e., working within 863–870 MHz (SRD860) band. Matching is built with a transmission line and passive SMD components.

BACKSCATTER TRANSMITTER

It is composed of an Analog Devices ADG904 SP4T RF switch inducing three modulation states (reflecting, reflecting with 90°phase shift, and non-reflecting an impinged radio frequency signal from external transmitter: non-dedicated as in [124, 170], or dedicated). The phase shift enables the tag to combat the phase cancellation problem of a T2T link [37] discussed also in Section 6.2.2. The phase-shift technique implementation is based on a design proposed in [37, Sec. IV-A], i.e., phase change-inducing switch connected in-between antenna and the SMA port of the transceiver to reflect the same bit with an inverted phase. The embedded version of the tag, Figure 6.1a, uses an SMD version of the switch, while transceiver-only version, Figure 6.1b, used ADG904 RF SP4T switch evaluation board connected via the antenna port (not shown on the photograph). The RF switch is controlled digitally by an embedded microcontroller (MCU), which is described in Section 6.3.3.

BACKSCATTER RECEIVER

It is composed of (i) an envelope detector (to filter out the mix of carrier and other tag's backscatter signal), (ii) a low-pass/high-pass filtering block, (iii) a baseband amplifier, and (iv) a Texas Instruments LMC7215 comparator (i.e., a one-bit ADC).

Envelope Detector To filter out the RF components of the carrier, a rectifier diode (Avago HSMS286) followed by a parallel RC network are used. We enhance the voltage swing of the detector input RF signal by biasing the diode to overcome its threshold voltage. This distinguishes our design from the envelope detector introduced in [167, Figure 4] which uses voltage doubler. The benefit of biasing is a lower junction resistance of the diode, which enables easier matching to a 50 Ω transmission line [171, p. 6]. A shunt resistor is added before the diode for improved matching (reducing reflection losses) for a desirable $S_{11} < -10$ dB. For our design $S_{11} = -32.31$ dB. This result was obtained with ADS.

High-Pass Filter The output of the envelope detector is passed to a high-pass filter to block the DC component (since there is information in it) of the carrier wave.

Ultra-Low-Power Amplifier We propose to amplify the output of the high-pass filter[1]. Although the diode is biased to improve the baseband signal swing, this voltage swing is still in μV range at low power levels. If the comparator would be directly connected to the envelope detector, such as in [17, Figure 6], [124, Figure 4], the total sensitivity would be limited due to the comparator offset of typically few millivolts. In our design we use the off-the-shelf amplifier, Maxim Integrated MAX9914, (Figure 6.6), instead of a custom-build one like in [167, Sec. III]. This way, we keep the monetary costs low, enabling widespread use of our platform among the research community [166][2]. The use of an amplifier is being traded for increased energy consumption compared to existing state-of-the-art tags—nevertheless, it is still many times lower than low-power active radios [36, Figure 3]. A gain of 100 was chosen for amplification.

Comparator The output of the amplifier is fed into the inputs of the comparator. The comparator's inverting terminal includes an averaging circuit to specify the right threshold for bits detection[3]. Finally, to minimize the number of false triggers due to noise, a hysteresis of 10 mV is added. The used comparator was Texas Instruments LMC7215.

6.3.2. BACKSCATTER T2T TAG: POWER SUPPLY

The communication distance of a passive backscatter tag, i.e., powered by the RF energy of a carrier generator, is limited by the minimum power that needs to be supplied at the RF harvester (which should be greater than -25 dBm [172, Sec. 2]). Therefore, to extend tags communication range, and to simplify the design, we choose for semi-passive (energy-assisted) architecture [173, Sec. II] by powering tags from the non-RF energy source, i.e., a solar panel.

Specifically, to power the transceiver and the MCU of our tags, two monocrystalline 2 V/44.6 mA IXOLAR SLMD121H04L solar panels [148] were connected in series to TI BQ25570EVM-206 evaluation board [147] (in case of transceiver design only) hosting TI BQ25570 [174] harvester power management circuit. Finally, we note that the parameters of the regulating nano-power management circuit of TI BQ25570 were set following TI BQ25505/70 Design Help V1.3 tool [174]. The energy is stored in a 470 μF supercapacitor, while for prototyping and measurement tags were powered directly via the USB interface of the MCU or via a battery.

6.3.3. BACKSCATTER T2T TAG: COMPUTING ENGINE

The backscatter transceiver is connected to TI MSP430FR5969 MCU, i.e., with non-volatile memory. MCU's role is to encode and decode incoming frames of the protocol introduced in Section 6.5. In the case of transceiver-only fabrication of the T2T tag (Fig-

[1] This idea was independently proposed in [167, Sec. III-B].

[2] We note that up to now regrettably *no tag-to-tag hardware*, including [17, 124, 167, 170] *has been open-sourced.*

[3] Potential improvements such as dynamic tracking of a threshold for comparator and self-interference cancellation circuit advocated in [165, Sec. 2.2] were left out for future design.

ure 6.1b), the transceiver is connected to a launchpad [175] (for ease of prototyping), in contrary to the self-contained fabricated tag (Figure 6.1a).

6.3.4. BACKSCATTER T2T TAG: COMPLETE DESIGN

We have fabricated two types of T2T tags, both presented in Figure 6.1. The first one is a transceiver board only which can be connected to any MCU, energy harvester or power supply of choice (Figure 6.1b). All components of this transceiver were hand-soldered on a two-layer 3 cm×4.3 cm printed circuit board. The second design is an all-in-one tag (Figure 6.1a), containing all required components on a single PCB. Both types of fabricated tags used 868 MHz 3 dBi gain 50 Ω omni-directional whip antenna connected through an SMA connector to the input port. In the rest of the chapter, we have experimented only with the first option due to cost considerations. The total cost of the T2T tag transceiver, based on standard hardware suppliers, did not exceed 25 €.

6.4. BACKSCATTER T2T TAG DESIGN: TAGS AND CARRIER GENERATORS

T2T Tag Population. We locate our T2T network indoors. Tags can be placed anywhere—either in line-of-sight to the exciter or not. Each T2T tag has the same level of hierarchy, master-master, in contrast to architecture of [122].

Number of Carrier Generators. Favoring mono-static setup against complex/costly multi-static one [173] we use $N_g = 1$ carrier generators (with no connectivity options either with other T2T tags or the outside world). Naturally, in any T2T network $N \gg E$ is desired.

Carrier Generator. In our experiments, we used two carrier generators: Agilent E4438C ESG and Hewlett-Packard HP8648C (use of which specific generator will be reported explicitly in the respective experimental results in Section 6.6)[4]. Any signal generator is connected to Liard S928PCR 902–928 MHz 8 dBic, right-hand circularly polarized antenna. The maximum output power of the generator is set to 20 dBm (for comparison, a 13 dBm output power carrier generator was used in [173]). We do not perform any duty cycling, which is allowed by the regulatory bodies for some RFID spectrum ranges, i.e., 915–921 MHz [176, Sec. 4.3.7.3].

6.5. BACKSCATTER T2T TAG DESIGN: PROTOCOL SUITE

We proceed with the introduction of our protocol suite of the backscatter T2T network. In particular, we introduce a MAC design that perfectly fits with the energy constraints of the T2T tags, followed by the link and application layer.

6.5.1. BACKSCATTER T2T MEDIUM ACCESS CONTROL

The MAC layer of the studies [17, 123] implements a continuous carrier sensing mechanism to detect incoming bits as well as to avoid interference between neighboring tag-

[4]We note that although our tags were designed to work with ambient carrier signals, due to extremely weak signal levels and lack of control over the location of the carrier wave, we have tested our network with signal sources we had full control of.

Table 6.3: Energy consumption of T2T Tag; Transceiver (as in Figure 6.1b) connected to MSP430 evaluation board [175]

Operation mode	Energy consumption (mW)
Reception only	1.3
Transmission only⋆	0.7
MCU only (Low Power Mode 0)†‡[80]	2.2

⋆ T2T transmitter set to transmit continuous stream of bits
† Measurements using [160], powered at 3.6 V; 16 MHz MCU clock cycle
‡ Includes powering all on-board peripherals, including USB controller

to-tag links. This approach demands the receiver and the MCU to be always kept on in order to receive incoming packets—leading to considerable energy overhead.

Observation 1—**Backscatter reception is more costly.** Backscatter transmission is cheap, since it is performed only by toggling the MCU port connected to the RF switch (see Figure 6.6 or [36, Figure 3]). However, contrary to active radios[5], backscatter reception is far more energy costly as compared to backscatter transmission. This is due to the additional energy cost of the receiver circuitry and the uncontrollable MCU false interrupts during the reception window. This is proven by our example T2T tag energy consumption measurements, see Table 6.3: reception is almost *twice as expensive* as transmission. All in all, the MCU is the most power-hungry component of the tag *requiring sleep scheduling.*

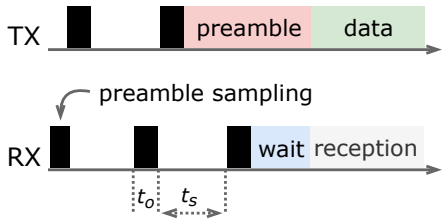
Observation 2—**Ambient noise prevents energy-efficient wake-up radio operation.** Since the comparator circuit triggers MCU at each bit transition, the backscatter receiver can be seen as a *wake-up* radio, which eliminates *idle listening* [178]. However, due to ambient noise, the MCU can also be triggered in the absence of any backscatter transmission. These *false triggers* are dependent on the selection of the comparator threshold and will wake-up the MCU from the sleeping state quite frequently—leading to wasted energy due to false wake-up and listen periods.

Observation 3—**Packet detection using carrier sense is inefficient for low data rates.** Employing carrier sense to detect if there is an incoming packet, as e.g. in [17, Sec. 4.2.1], requires the backscatter receiver to be *always* kept on in order to process incoming bits. However, this leads to significant overhead due to the energy cost of the reception (*vide* Observation 1) and false triggers (*vide* Observation 2).

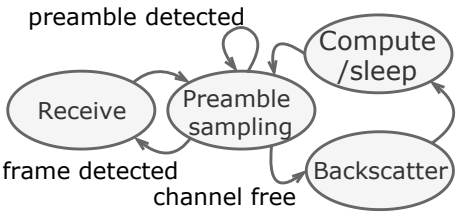
Observation 4—**Synchronous operations are infeasible.** Introducing synchronous protocols; e.g. forcing tags to wake-up their backscatter receivers just before their assigned slots as in TDMA, is not feasible as of now due to the reasons being [179, Sec. 3]: (i) backscatter tags would not have timekeeping mechanism with frequent power losses; (ii) there is no central entity in the network which is continuously powered, assigning transmission schedules (such as RFID reader)—making approaches such as [36, Sec. 3.5] inapplicable.

Proposition—**MAC Paradigm for Backscatter Tags: Low-Power Listening** Given *Observations 1–4* we advocate for *asynchronous* MAC design based on *low power listening* inspired by [180, Sec. 3].

[5] Refer to, e.g., LTE Cat NB1 radio [177, Table 4.2.3], where transmission expenditure increases with transmission output power.



(a) Low power listening (for t_s and t_o see Table 6.4).



(b) T2T MAC finite state machine (for both transmitter and receiver)

Figure 6.7: Schematic representation of the proposed MAC protocol.

Table 6.4: Selected T2T MAC parameters; see [166] for details

Parameter	Description	Value	Unit
b_r, b_t	RX and TX buffer size	8, 8	Frames
l_b	Bit length	Section 6.5.1	CC*
t_s	Sleep period	26.5	ms
t_x	Frame reception timeout	15	ms
t_p	Frame preamble length	36	ms
t_s, t_g	Inter-frame time	0.25	ms
t_l, t_h	Timer jitter low and high	25, 37.5	μs
t_o	Channel observation period	6.1	ms
n_o	Min bit transitions within t_o	8	Bits

* CC: cycles of SMCLK clock of MCU [175] at 16 MHz

LOW POWER LISTENING AND MAC STATE MACHINE

High-level design of our MAC is illustrated in Figure 6.7a, while MAC finite state machine is presented in Figure 6.7b. Briefly, a T2T node wakes up periodically every 26.5 ms (see Table 6.4), detects the preamble (by reading bits at the comparator output port and searching for the predefined preamble, Table 6.4) and synchronizes to the delimiter of the received frame. A frame validation is started by calculating the CRC of the received frame and comparing it to the CRC bytes of the frame. If they match the freshness of the frame is checked by searching a ring buffer that keeps track of the most recent y (in our implementation $y = 10$) frames. If the frame is new and the frame `Receiver ID` equals the node `ID`, then the payload is saved in the memory and made accessible to the application layer. However, if the `ID`s mismatch the frame will be saved in the transmission buffer to be forwarded.

In order to avoid collision each data transmission is preceded by channel observation (preamble sampling) and a long preamble (see Table 6.4) is transmitted to wake up surrounding nodes. Once a frame is received or discarded the MCU transitions to the low-power mode (i.e., Low Power Mode 0 of the MSP430 MCU used [80]) turning off the T2T backscatter receiver until the next wake-up. Furthermore, the MAC cycle is randomized to prevent nodes from waking up and estimating the channel at the same time, which may lead to concurrent transmission, see Figure 6.7b. Our MAC enables nodes to transmit with and without a phase shift, of 90°, to enable nodes to cope with the dead spots in backscatter networks [37].

Table 6.5: Frame Structure of the Implemented T2T MAC Protocol

Field	Length (B)	Notes
Preamble	Time based	`0xBB`*
Start Frame Delimiter	1	`0xAA`*
Sender ID	1	Node ID
Receiver ID	1	Node ID
Message Type†	1	`Broadcast` ID Byte = `0xFF`
Message ID	1	—
Payload	4	—
CRC	2	CRC-CCITT hardware-based [175]

* Values obtained experimentally
† In experiments only `Broadcast` used; other types like `ACK`, `Beacon`, `Data` possible

MAC FRAME STRUCTURE

We implement a frame structure described in Table 6.5. Bits in the frame are coded with FM0 modulation. Finally, we note that bit lengths can vary in a set [1600, 16000, 31250] of SMCLK clock cycles that corresponds to respective [10 k, 1 k, 512] bps data rates. In the actual implementation, we have chosen for the highest clock rate/bit rate, i.e., 1600 clock cycles/10 kbps, respectively.

T2T TAG ADDRESSING

All tags have a predefined addresses of 1 B long (Table 6.5). We do not implement any tag discovery protocol at this stage.

6.5.2. BACKSCATTER T2T LINK LAYER

To increase the robustness of the T2T network we chose to implement a flooding mechanism to forward the messages. Flooding also implicitly eliminates the need for implementing error correcting mechanism, since multiple copies of a frame are forwarded. To limit the number of the messages being forwarded each node is allowed to backscatter a new frame z times (in our implementation $z = 1$). Furthermore, to reduce the probability of collisions each tag observes the channel before backscattering and its wake-up cycles are randomized within a range of [0, 5] ms.

6.5.3. BACKSCATTER T2T PROTOCOL IMPLEMENTATION DETAILS

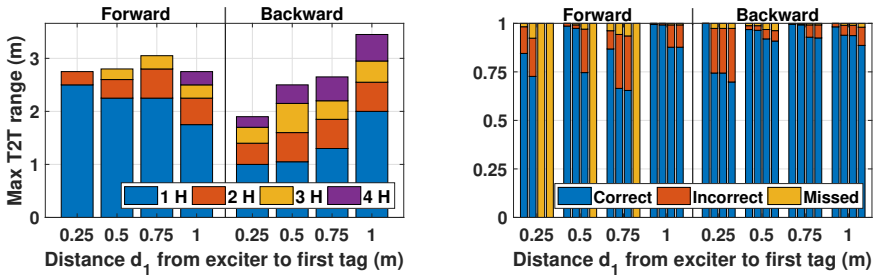
The footprint of our protocol implementation was 8 kB (program memory) and 606 B (RAM). The complete protocol was implemented in C language, amassing to 2113 lines of code.

6.6. BACKSCATTER T2T NETWORK: EVALUATION

We now proceed with describing the experimental results of our backscatter T2T network. We start by introducing the T2T network setup and measurement methodology.

6.6.1. BACKSCATTER T2T NETWORK: EXPERIMENT SETUP

We performed all experiments indoors, in a large office space with many metallic shelves and concrete walls of mixed thickness. A total of seven tags were used in the experiments. Tags and the exciter's antenna were mounted on tripods, elevated 1 m above ground.



(a) T2T multi-hop range at hop H

(b) Received frame distribution

Figure 6.8: T2T link metrics for forward and backward link. Figure 6.8a: the backward link has a higher multi-hop gain, *doubling the range* as receiving tag moves away from the exciter; Figure 6.8b: frame distribution per hop (ordered from left to right in each bar group)—backward link is less prone to errors .

6.6.2. RESULT 1—T2T RANGE AND MULTI-HOP PERFORMANCE

Measurement Methodology. Each measurement (data point in Figure 6.8) is the average of five runs, each of them consisting of a hundred frames sent by the source tag, see Table 6.5. The maximum distance of a hop is assumed to be reached when 75% of the frames are correctly received at the destination.

Figure 6.8 presents the core results on the multi-hop T2T link properties. Measurements of the range improvement that multi-hop brings in backscatter T2T networks are given in Figure 6.8a, both for forward and backward links. The figure shows the maximum distance that can be covered in a one-dimensional (line) multi-hop backscatter T2T network as a function of the number of hops and the distance d_1 from the exciter to the first T2T tag. The maximum number of hops is shown, meaning that, for instance for $d_1 = 0.25$ m only two hops were achievable in the forward link.

The core observation is that the ratio of first versus subsequent hops distance (i.e., multi-hop gain) increases with d_1 mainly because the first hop becomes shorter as less power to backscatter is available at the first tag. Furthermore, when T2T tags move away from the exciter, equivalent changes in the distance have less impact on available carrier power due to the logarithmic nature of path loss, increasing both the number and the length of the subsequent hops. As predicted by Observation 2, the multi-hop gain is higher in a backward link, increasing the range by about a factor of two already at four hops.

In Figure 6.8b the cumulative received frame distribution per hop is shown as a function of d_1, for both forward and backward links. Bars are grouped in order of hops, such that the leftmost bar of a group refers to the first hop, and the rightmost to the last possible hop for that distance. The main observation here is that the *backward link has a more stable* behavior.

6.6.3. RESULT 2—PHASE CANCELLATION AND NETWORK ROBUSTNESS

Measurement Methodology. A 2 m×2 m area is divided into a grid of 0.5 m increments. The exciter and source tag are positioned at relative location (0,0) m and (0.5,0.5) m, respectively. Each data point is the average of three runs of twenty-five frames each.

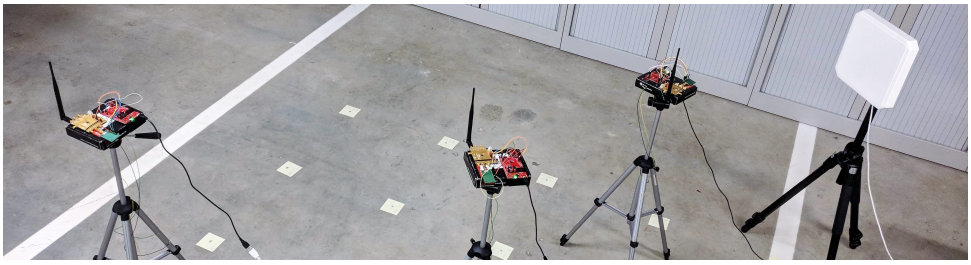
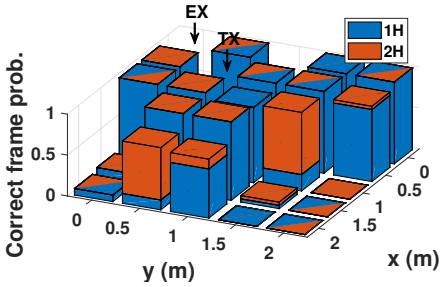
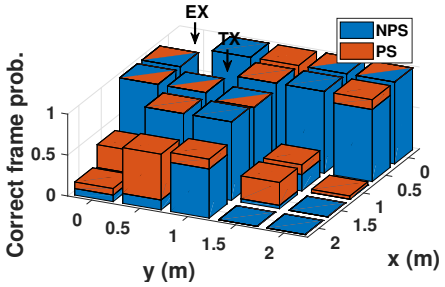


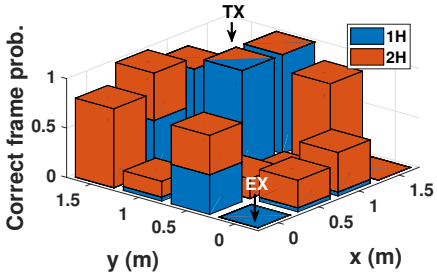
Figure 6.9: Photograph of the setup to evaluate phase cancellation countermeasures in T2T networks, discussed in Section 6.6.3: Three backscatter T2T tags on tripods, next to white panel antenna; grid coordinates marked with yellow squares on the floor.



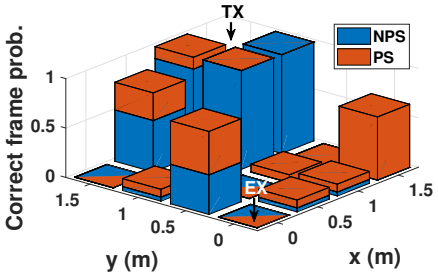
(a) Forward link multi-hop



(b) Forward link phase shifting



(c) Backward link multi-hop



(d) Backward link phase shifting

Figure 6.10: Backscatter T2T network phase cancellation experiment. *EX* and *TX* mark the positions of the exciter and transmitter, respectively. *(N)PS*: (no) phase shifting. Figure 6.10a and 6.10b: forward link. Both methods yield comparable results. Figure 6.10c and 6.10d: backward link. Multi-hop is superior in network coverage. Note: *1H* and *NPS* is the same case.

We now cover the performance of multi-hop as a solution against phase cancellation and compare it to phase shifting for the same purpose (i.e., sending every frame twice with a phase offset of 90°). This experiment also shows the robustness increase in the T2T network by the use of multi-hop. Furthermore, adding or subtracting relaying tags simulates the case of message forwarding during interference and network reshaping, e.g. from T2T tag mobility.

Table 6.6: T2T Network Coverage: Summary of Figure 6.10 results

	Vanilla*(%)	Phase shifting†(%)	Multi-hop flooding†(%)
Forward link	58.1	67.3	65.0
Backward link	28.0	40.9	54.3

* No phase cancellation fighting mechanism, i.e., traditional single-hop T2T
† As analyzed in Section 6.2.2

We proceed by placing a destination tag and measuring the rate of correct frame reception at each coordinate of the grid. Then, we compare this benchmark with the two approaches. The relaying tags are added at the closest grid coordinates to the middle point between source and destination tags, forming a two-dimensional network, see Figure 6.9. For the sake of comparison fairness, this approach only uses one relaying tag (e.g. two hops) so that both solutions are balanced in network utilization: twice the original number of frames in both cases.

Figure 6.10 presents the results of the experiment, while global (average) coverage values are shown in Table 6.6. Note that for the backward link the network area was reduced to 1.5 m×1.5 m due to the weaker nature of this link. Points of weak reception in the area could be caused by phase cancellation or by multi-path fading due to the unfavorable testing environment. This hypothesis gains relevance when we look at the cases in which reception is greatly improved by adding a relay tag or by using phase shifting. However, other points could not be enhanced by either method.

In the forward link, multi-hop and phase shifting as means to fight phase cancellation report close results, improving coverage moderately by about 1.13×. On the other hand, the backward link is better handled by the multi-hop solution, almost doubling T2T network coverage. Multi-hop is therefore preferred in either case, as it also adds range extension and robustness to the T2T network.

6.6.4. CASE STUDY: JOINING TWO T2T BACKSCATTER CLUSTERS

As a final experiment, we present a case study showcasing the junction of two tag clusters. This study displays the possibility of joining distinct T2T networks with their own exciters by positioning a middle tag connecting both groups [6].

The first cluster consists of three tags, spanning a total distance of 2.5 m from their exciter. At the other side of the laboratory, there is a second cluster, also composed of three tags, reaching 1.8 m in the other direction. Figure 6.11 depicts this scenario. By placing the seventh tag in the middle of the two groups, we are able to join them and successfully establish communication from one side to the other. There is a forward link in the first cluster, followed by a backward one in the second cluster of tags, covering a total distance of 5.65 m.

6.7. LIMITATIONS OF THIS WORK

Hardware Improvements. Beyond the improvement to increase the receiver sensitivity of the T2T tag, which would increase the T2T communication distance, other develop-

[6] Note that, the exciters should not transmit at the same time, e.g., they need to apply a Time-Division-Multiple-Access (TDMA) technique or randomize their transmission slots.

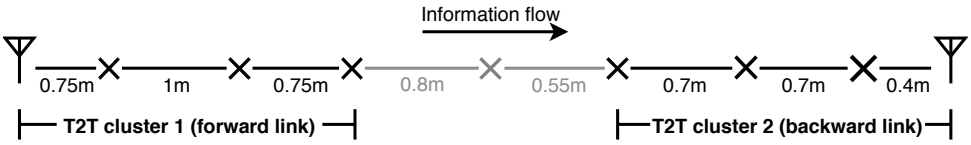


Figure 6.11: A scenario of T2T clusters junction by adding a bridging T2T node. Antenna symbols represent exciters and crosses represent tags.

ments are required. These include building a full-duplex or multi-antenna transceiver (enabling space-time coding, for instance).

Protocol Improvements. T2T network was not designed with security in mind (to speed up the design time), i.e., transmitted data is not obfuscated. Other improvements include a better (non-flooding) routing algorithm and link adaptation mechanism (based on link packet error rate measurements, for instance). Moreover, we have not performed experiments on harvested power only, so the consideration of T2T networking under intermittent power [18] is another critical research step.

6.8. CONCLUSIONS

In this chapter, we built, characterized and evaluated a network composed of backscatter tag-to-tag (T2T) links. In short, we reported the first successful demonstration of the largest multi-hop decode-and-relay T2T network using distributed embedded backscatter transceivers. Our results show that the T2T multi-hop approach is superior to the phase-shifted frame repetition technique in mitigating the effect of dead spots in T2T backscatter networks.

7

CONCLUSIONS

The main obstacles to achieve truly ubiquitous sensing are (i) the limitations of battery technology—batteries are short-lived, hazardous, bulky, and costly—and (ii) the unpredictability of ambient power. The latter causes energy-harvesting sensors to operate intermittently, breaking an underpinning assumption of current software architectures—a power failure is a rare event—and violating the availability requirements of many real-world applications. This intermittent power supply necessitates rethinking classical software architectures and the design of energy-harvesting battery-less sensors.

Reactive, Dynamic Software Architecture will Drive Future Intermittent Platforms. Prior intermittent systems considered the challenges associated with intermittent-power supply mainly from a computing standpoint. As a result, these systems are inherently static: they do not enable intermittent systems to react to environmental changes. This thesis envisions that the primary application of intermittent platforms is sensing. Sensing applications are inherently reactive: sensor nodes sleep in low-power mode waiting for an event to wake them up and to trigger the corresponding computation thread. Therefore, it proposes InK, a reactive intermittent kernel. InK features a power-failure immune scheduler to enable event-driven intermittent execution. Further, it extends `C` with new language abstractions to enable safe data gathering despite frequent power failures. This wake-on-event style of operation is particularly important for intermittent systems as they have limited energy budget per power cycle. Therefore, putting sensors in low-power mode waiting for events prolongs their on-times, which increases the probability of successful event capturing.

The variation in the ambient energy and differences buffer sizes—when heterogeneous devices are considered—translate into varying uptimes. *Dynamic* uptime creates opportunities to execute more instructions before suspending application execution to commit and protect the computation progress. Static protection approaches are agnostic to these opportunities, and therefore, they are doomed to be overprotecting, wasting energy and thereby effective on-time. Furthermore, if a static task demands more energy than what the buffer can provide, the system will get stuck in this non-terminating

task. This thesis introduces Coala to address the limitations of static task-based systems. Coala uses efficient, energy-aware task coalescing strategy to amortize static task overheads and capitalize on the variation of devices on-times. Further, it uses a timer-based task-splitting mechanism to avoid non-termination of tasks too long for a device's energy buffer.

Intermittent Sensors Approximate Continuous Availability when Grouped Together. The unique characteristics of energy-harvesting battery-less devices do not only create new challenges, but also new opportunities for new design dimensions. Prior to this thesis, to meet a certain on/off duty cycle of an intermittent device, the energy harvester needed to be custom designed for particular ambient energy conditions. For example, given certain light intensity, a solar panel must be of a particular size to deliver certain output power that results in a particular charge-discharge cycle.

This thesis proposes an alternative way to meet the availability required of an application. It exploits the little differences between the charge-discharge (or power) cycles of energy-harvesting battery-less sensors to arrive at a uniform distribution of a group of intermittent nodes' on-times (Chapter 5). The emerging joined availability of these sensors is a function of their *number*. As such, the targeted system availability can be met *not* only by changing sensor hardware (e.g., a bigger solar cell), but also by increasing the number of intermittent sensors.

However, the variation in energy-harvesting rate and power consumption complicate the design of the proposed Coalesced Intermittent Sensor (CIS): a group of energy-harvesting battery-less sensors. In event-driven sensing nodes typically employ sleep mode to minimize energy consumption. As a consequence, nodes become available on longer time intervals, and therefore, nodes' awake times overlap more. Favorable energy conditions extend the overlapping intervals even further. In such situations, many nodes will respond to the first incoming event, and consequently, power down at roughly the same time as event processing consumes the rest of their buffered energy quickly. This unwanted synchronization minimizes the overall availability of the system as nodes power up and down together. To overcome this limitation energy-harvesting battery-less sensors must be to estimate the number of active nodes and respond proportionally to incoming events. For that, intermittent nodes need to know the total number of nodes in their CIS; their current power cycles, which can be estimated by measuring the on-time on a fixed load; and nodes' on-time distribution, which is naturally uniform.

Multi-hop is resilient to Dead Spots in Backscatter Networks. From a sensor perspective, backscattering is much more energy-efficient than active communication. This is because backscatter communication allows sensors to reuse existing-in-the-air signals instead of emitting their own. However, due to the dis-locality between the energy source (the carrier generator) and the information source (the sensor), backscatter sensor-to-sensor networks have persistent dead spots. The reason behind the discontinuous coverage is a phenomenon called *phase cancellation*; at certain locations, the backscatter signal representing the logic "1" and "0" induces the same change to the carrier signal, and therefore, it is indistinguishable. Multi-hop backscattering (Chapter 6) is naturally resilient to the phase-cancellation problem, as it allows a message to arrive at the re-

ceiver from different paths with different *phase shifts*. Furthermore, multi-hopping is extremely important for backscatter networks as backscatter signals are weak as compared to their carrier signals. Thus, by chaining these short backscatter sensor-to-sensor links, multi-hopping allows the sensors to exploit the full range of the ambient signals, instead of being limited to the ranges of the weak backscatter signals.

To conclude, this thesis detailed the way from intermittent computing to reliable sensing on intermittent power. For that, it introduced an intermittently-powered kernel that enables reactive and dynamic execution (Chapter 3 and 4), and a virtual sensor that abstracts a group of intermittently-powered sensors to provide reliable and continuous sensing (Chapter 5). Further, this dissertation advocated for a passive communication means between intermittent sensors, and therefore, it proposed a multi-hop backscatter sensor-to-sensor network architecture (Chapter 6).

FUTURE WORK

Here, we speculate on potential future research directions of intermittent sensors and their impact on everyday life.

Coalesced Intermittent Sensors Networks Intermittent sensors eventually need to communicate the processed data. From the energy perspective, backscatter communication is a promising technology to drive the communication between intermittent nodes. However, backscatter links are short; therefore, these nodes need to be cooperative, i.e., forward the messages of other nodes, to deliver sensory data to a sink. Given the limited energy per power cycle, it is interesting to investigate the cooperative strategy that maximizes nodes' probability of successful sensing while preserving network connectively.

Intermittent Systems and Machine Learning Intermittent sensors can partially capture events. Classical algorithms for recognizing and classifying events might have a hard time dealing with partially captured data. Thus, follow-up work can investigate if and how much machine learning algorithms can improve the quality of intermittent sensing.

From Intermittent Systems to Smart Buildings We think that our work forms a basis for developing smart and energy-autonomous skins, wallpapers, etc., which will have a great social impact. For example, walls that can hear and see (through smart wallpapers) might enable instant and anywhere in-building communication (without smartphones). Furthermore, the smart wall concept, in turn, might require to re-think the architectural design of the future smart buildings.

Propositions

accompanying the dissertation

CONTINUOUS SENSING ON INTERMITTENT POWER

by

Amjad Yuosef MAJID

1. Intermittent kernels should be developed as reactive systems (Ch. 3).
2. Intermittent kernels can estimate their uptime by counting executed tasks (Ch. 4).
3. The on-times of a group of intermittent sensors are uniformly distributed (Ch. 5).
4. Multi-hop backscattering is resilient to the phase-cancellation problem (Ch. 6).
5. Active and passive communication can co-exist.
6. A computer scientist works either with or for a computer.
7. There is a great similarity between a scientific researcher and a mountaineer.
8. Time and immortality are mutually exclusive.
9. Worthless content on social media costs society more than fighting cancer.
10. The biggest problem humanity can face is that there is no problem.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotor Prof. Dr. K. Langendoen. and the co-promotor Dr. P. Pawełczak.

REFERENCES

[1] R. P. Dameri and C. Rosenthal-Sabroux, *Smart city: How to create public and economic value with high technology in urban space* (Springer, 2014).

[2] L. Atzori, A. Iera, and G. Morabito, *Understanding the internet of things: definition, potentials, and societal role of a fast evolving paradigm,* Ad Hoc Networks **56**, 122 (2017).

[3] B. Silva, M. Khan, C. Jung, J. Seo, D. Muhammad, J. Han, Y. Yoon, and K. Han, *Urban planning and smart city decision management empowered by real-time data processing using big data analytics,* Sensors **18**, 2994 (2018).

[4] J. Polastre, R. Szewczyk, and D. Culler, *Telos: enabling ultra-low power wireless research,* in *Proceedings of the 4th international symposium on Information processing in sensor networks* (IEEE Press, 2005) p. 48.

[5] D. C. Daly, P. P. Mercier, M. Bhardwaj, A. L. Stone, Z. N. Aldworth, T. L. Daniel, J. Voldman, J. G. Hildebrand, and A. P. Chandrakasan, *A pulsed uwb receiver soc for insect motion control,* IEEE Journal of solid-state circuits **45**, 153 (2010).

[6] Department of Labor, The US Goverment, *Occupational safety and health administration,* (2019).

[7] S. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K.-S. Kwak, *The internet of things for health care: a comprehensive survey,* IEEE Access **3**, 678 (2015).

[8] R. Margolies, G. Grebla, T. Chen, D. Rubenstein, and G. Zussman, *Panda: Neighbor discovery on a power harvesting budget,* (IEEE, 2016) pp. 3606–3619.

[9] M. Gorlatova, J. Sarik, G. Grebla, M. Cong, I. Kymissis, and G. Zussman, *Movers and shakers: Kinetic energy harvesting for the internet of things,* (IEEE, 2015) pp. 1624–1639.

[10] M. Gorlatova, P. Kinget, I. Kymissis, D. Rubenstein, X. Wang, and G. Zussman, *Energy harvesting active networked tags (enhants) for ubiquitous object networking,* IEEE Wireless Communications **17**, 18 (2010).

[11] S. Gollakota, M. S. Reynolds, J. R. Smith, and D. J. Wetherall, *The emergence of rf-powered computing,* Computer **47**, 32 (2014).

[12] H. T. Friis, *A note on a simple transmission formula,* Proceedings of the Institute of Radio Engineers **34**, 254 (1946).

[13] M. Minami, T. Morito, H. Morikawa, and T. Aoyama, *Solar biscuit: A battery-less wireless sensor network system for environmental monitoring applications,* in *The 2nd international workshop on networked sensing systems* (Citeseer, 2005) p. 2007.

[14] H. Zhang, J. Gummeson, B. Ransford, and K. Fu, *MOO: A batteryless computational RFID and sensing platform (technical report um-cs-2011-020),* `https://web.cs.umass.edu/publication/docs/2011/UM-CS-2011-020.pdf` (2011), last accessed: Mar. 30, 2018.

[15] R. J. Vyas, B. B. Cook, Y. Kawahara, and M. M. Tentzeris, *E-wehp: A batteryless embedded sensor-platform wirelessly powered from ambient digital-tv signals,* IEEE Transactions on microwave theory and techniques **61**, 2491 (2013).

[16] H. Aantjes, A. Y. Majid, and P. Pawełczak, *A testbed for transiently powered computers,* arXiv preprint arXiv:1606.07623 (2016).

[17] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, *Ambient backscatter: wireless communication out of thin air,* in *ACM SIGCOMM Computer Communication Review*, Vol. 43 (ACM, 2013) pp. 39–50.

[18] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, *Intermittent computing: Challenges and opportunities,* in *2nd Summit on Advances in Programming Languages (SNAPL 2017)* (Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017).

[19] V. S. Rao, *Ambient-energy powered multi-hop internet of things,* (2017).

[20] A. Parks, I. in 't Veen, S. Naderiparizi, and J. Tan, *WISP 5.0 firmware git,* `https://github.com/wisp/wisp5` (2019), last accessed: Oct. 10, 2019.

[21] J. Hester and J. Sorber, *Flicker: Rapid prototyping for the batteryless internet-of-things,* in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems* (ACM, 2017) p. 19.

[22] TI Inc., *MSP430FR59xx mixed-signal microcontrollers (Rev. F),* `http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf` (2017), last accessed: Jul. 26, 2017.

[23] Texas Instruments, Inc., *FRAM faqs,* `http://www.ti.com/lit/ml/slat151/slat151.pdf` (2014), last accessed: Mar. 30, 2018.

[24] B. Ransford, J. Sorber, and K. Fu, *Mementos: system support for long-running computation on rfid-scale devices,* in *ACM SIGARCH Computer Architecture News*, Vol. 39 (ACM, 2011) pp. 159–170.

[25] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, *Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems,* IEEE Embedded Systems Letters **7**, 15 (2014).

[26] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, *Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices,* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**, 1968 (2016).

[27] N. Bhatti and L. Mottola, *HarvOS: Efficient code instrumentation for transiently-powered embedded devices,* in *Proc. IPSN* (ACM/IEEE, Pittsburgh, PA, USA, 2017).

[28] B. Lucia and B. Ransford, *A simpler, safer programming and execution model for intermittent systems,* in *ACM SIGPLAN Notices*, Vol. 50 (ACM, 2015) pp. 575–585.

[29] J. Van Der Woude and M. Hicks, *Intermittent computation without hardware support or programmer intervention,* in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016) pp. 17–32.

[30] H. Jayakumar, A. Raha, and V. Raghunathan, *Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers,* in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems* (IEEE, 2014) pp. 330–335.

[31] M. Hicks, *Clank: Architectural support for intermittent computation,* in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (IEEE, 2017) pp. 228–240.

[32] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola, *Efficient intermittent computing with differential checkpointing,* in *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems* (ACM, 2019) pp. 70–81.

[33] A. Colin and B. Lucia, *Chain: tasks and channels for reliable intermittent programs,* in *ACM SIGPLAN Notices*, Vol. 51 (ACM, 2016) pp. 514–530.

[34] K. Maeng, A. Colin, and B. Lucia, *Alpaca: intermittent execution without checkpoints,* (ACM, 2017) p. 96.

[35] J. Hester, K. Storer, and J. Sorber, *Timely execution on intermittently powered batteryless sensors,* in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems* (ACM, 2017) p. 17.

[36] V. Talla, M. Hessar, B. Kellogg, A. Najafi, J. R. Smith, and S. Gollakota, *Lora backscatter: Enabling the vision of ubiquitous connectivity,* Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **1**, 105 (2017).

[37] Z. Shen, A. Athalye, and P. M. Djurić, *Phase cancellation in backscatter-based tag-to-tag communication systems,* IEEE Internet of Things Journal **3**, 959 (2016).

[38] A. Colin and B. Lucia, *Termination checking and task decomposition for task-based intermittent programs,* in *Proceedings of the 27th International Conference on Compiler Construction* (ACM, 2018) pp. 116–127.

[39] J. P. Lynch and K. J. Loh, *A summary review of wireless sensors and sensor networks for structural health monitoring,* Shock and Vibration Digest **38**, 91 (2006).

[40] P. P. Ray, M. Mukherjee, and L. Shu, *Internet of things for disaster management: State-of-the-art and prospects,* IEEE Access **5**, 18818 (2017).

[41] S. H. Shah and I. Yaqoob, *A survey: Internet of things (iot) technologies, applications and challenges,* in *2016 IEEE Smart Energy Grid Engineering (SEGE)* (IEEE, 2016) pp. 381–385.

[42] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler, *A building block approach to sensornet systems,* in *Proceedings of the 6th ACM conference on Embedded network sensor systems* (ACM, 2008) pp. 267–280.

[43] J. Hester, T. Peters, T. Yun, R. Peterson, J. Skinner, B. Golla, K. Storer, S. Hearndon, K. Freeman, S. Lord, *et al.*, *Amulet: An energy-efficient, multi-application wearable platform,* in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM* (ACM, 2016) pp. 216–229.

[44] C. Park, J. Liu, and P. H. Chou, *Eco: an ultra-compact low-power wireless sensor node for real-time motion monitoring,* in *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* (IEEE, 2005) pp. 398–403.

[45] M. P. Andersen, G. Fierro, and D. E. Culler, *System design for a synergistic, low power mote/ble embedded platform,* in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)* (IEEE, 2016) pp. 1–12.

[46] K. S. Adu-Manu, N. Adam, C. Tapparello, H. Ayatollahi, and W. Heinzelman, *Energy-harvesting wireless sensor networks (eh-wsns): A review,* ACM Transactions on Sensor Networks (TOSN) **14**, 10 (2018).

[47] R. V. Prasad, S. Devasenapathy, V. S. Rao, and J. Vazifehdan, *Reincarnation in the ambiance: Devices and networks with energy harvesting,* IEEE Communications Surveys & Tutorials **16**, 195 (2013).

[48] A. P. Sample, B. H. Waters, S. T. Wisdom, and J. R. Smith, *Enabling seamless wireless power delivery in dynamic environments,* Proceedings of the IEEE **101**, 1343 (2013).

[49] K. Huang and X. Zhou, *Cutting the last wires for mobile communications by microwave power transfer,* IEEE Communications Magazine **53**, 86 (2015).

[50] H. J. Visser and R. J. Vullers, *Rf energy harvesting and transport for wireless sensor network applications: Principles and requirements,* Proceedings of the IEEE **101**, 1410 (2013).

[51] P. Kamalinejad, C. Mahapatra, Z. Sheng, S. Mirabbasi, V. C. Leung, and Y. L. Guan, *Wireless energy harvesting for the internet of things,* IEEE Communications Magazine **53**, 102 (2015).

[52] M.-L. Ku, W. Li, Y. Chen, and K. R. Liu, *Advances in energy harvesting communications: Past, present, and future challenges,* IEEE Communications Surveys & Tutorials **18**, 1384 (2015).

[53] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, *Design considerations for solar energy harvesting wireless embedded systems,* in *Proceedings of the 4th international symposium on Information processing in sensor networks* (IEEE Press, 2005) p. 64.

[54] T. Xiang, Z. Chi, F. Li, J. Luo, L. Tang, L. Zhao, and Y. Yang, *Powering indoor sensing with airflows: a trinity of energy harvesting, synchronous duty-cycling, and sensing,* in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems* (ACM, 2013) p. 16.

[55] Y. Lee, G. Kim, S. Bang, Y. Kim, I. Lee, P. Dutta, D. Sylvester, and D. Blaauw, *A modular 1mm 3 die-stacked sensing platform with optical communication and multi-modal energy harvesting,* in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International* (IEEE, 2012) pp. 402–404.

[56] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, *Power management in energy harvesting sensor networks,* ACM Transactions on Embedded Computing Systems (TECS) **6**, 32 (2007).

[57] O. Cetinkaya, E. Dinc, C. Koca, G. V. Merrett, and O. B. Akan, *Energy-neutral wireless-powered networks,* IEEE Wireless Communications Letters (2019).

[58] B. Buchli, F. Sutton, J. Beutel, and L. Thiele, *Dynamic power management for long-term energy neutral operation of solar energy harvesting systems,* in *Proceedings of the 12th ACM conference on embedded network sensor systems* (ACM, 2014) pp. 31–45.

[59] Panasonic, *Eneloop rechargeable batteries,* `https://www.panasonic.com/global/consumer/battery/eneloop/whats.html` (2020), last accessed: Jan. 15, 2020.

[60] M. R. Palacín and A. de Guibert, *Why do batteries fail?* Science **351**, 1253292 (2016).

[61] D. N. Perkins, M.-N. B. Drisse, T. Nxele, and P. D. Sly, *E-waste: a global hazard,* Annals of global health **80**, 286 (2014).

[62] F. C. McMichael and C. Henderson, *Recycling batteries,* IEEE Spectrum **35**, 35 (1998).

[63] J. Camp and D. M. Camp, *Battery disposal system,* (2013), uS Patent App. 13/587,815.

[64] J. Gummeson, S. S. Clark, K. Fu, and D. Ganesan, *On the limits of effective hybrid micro-energy harvesting on mobile crfid sensors,* in *Proceedings of the 8th international conference on Mobile systems, applications, and services* (ACM, 2010) pp. 195–208.

[65] A. Erturk and D. J. Inman, *Piezoelectric energy harvesting* (John Wiley & Sons, 2011).

[66] S. Rodriguez, S. Ollmar, M. Waqar, and A. Rusu, *A batteryless sensor asic for implantable bio-impedance applications,* IEEE transactions on biomedical circuits and systems **10**, 533 (2015).

[67] G. Chen, H. Ghaed, R.-u. Haque, M. Wieckowski, Y. Kim, G. Kim, D. Fick, D. Kim, M. Seok, K. Wise, *et al.*, *A cubic-millimeter energy-autonomous wireless intraocular pressure monitor,* in *2011 IEEE International Solid-State Circuits Conference* (IEEE, 2011) pp. 310–312.

[68] P. Nadeau, D. El-Damak, D. Glettig, Y. L. Kong, S. Mo, C. Cleveland, L. Booth, N. Roxhed, R. Langer, A. P. Chandrakasan, *et al.*, *Prolonged energy harvesting for ingestible devices,* Nature biomedical engineering **1**, 0022 (2017).

[69] S. DeBruin, B. Campbell, and P. Dutta, *Monjolo: An energy-harvesting energy meter architecture,* in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems* (ACM, 2013) p. 18.

[70] S. Sudevalayam and P. Kulkarni, *Energy harvesting sensor nodes: Survey and implications,* IEEE Communications Surveys & Tutorials **13**, 443 (2010).

[71] EnOcean, *EnOcean energy harvesting wireless technology,* `https://www.enocean.com/` (2019).

[72] P. Corp., *Powercast hardware,* `http://www.powercastco.com` (2014), last accessed: Mar. 30, 2018.

[73] NOWI, *NOWI energy harvesting power management unit,* `https://www.nowi-energy.com/` (2019).

[74] A. Y. Majid, C. Delle Donne, K. Maeng, A. Colin, S. Yıldırım, P. Pawełczak, and B. Lucia, *Dynamic task-based intermittent execution for energy-harvesting devices,* ACM Transitions on Sensor Network (2019).

[75] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith, *Wispcam: A battery-free rfid camera,* in *2015 IEEE International Conference on RFID (RFID)* (IEEE, 2015) pp. 166–173.

[76] G. Vougioukas and A. Bletsas, *24μ watt 26m range batteryless backscatter sensors with fm remodulation and selection diversity,* in *2017 IEEE International Conference on RFID Technology & Application (RFID-TA)* (IEEE, 2017) pp. 237–242.

[77] 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**, 2608 (2008).

[78] Impinj Inc., *Impinj speedway R420 RFID reader product information,* `https://www.impinj.com/platform/connectivity/speedway-r420/` (2018), last accessed: Apr. 8, 2018.

[79] J. R. Smith, A. P. Sample, P. S. Powledge, S. Roy, and A. Mamishev, *A wirelessly-powered platform for sensing and computation,* in *International Conference on Ubiquitous Computing* (Springer, 2006) pp. 495–506.

[80] Texas Instruments, *MSP430FRXXXX 16 MHz ultra-low-power microcontroller product page,* (2017).

[81] Texas Instruments, *MSP-EXP432P401R datasheet,* (2019).

[82] STMicroelectronics, *STM32F415RG datasheet,* (2019).

[83] A. N. Parks, A. P. Sample, Y. Zhao, and J. R. Smith, *A wireless sensing platform utilizing ambient rf energy,* in *2013 IEEE Topical Conference on Biomedical Wireless Technologies, Networks, and Sensing Systems* (IEEE, 2013) pp. 154–156.

[84] A. Dementyev, J. Gummeson, D. Thrasher, A. Parks, D. Ganesan, J. R. Smith, and A. P. Sample, *Wirelessly powered bistable display tags,* in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing* (ACM, 2013) pp. 383–386.

[85] A. Saffari, M. Hessar, S. Naderiparizi, and J. R. Smith, *Battery-free wireless video streaming camera system,* in *2019 IEEE International Conference on RFID (RFID)* (IEEE, 2019) pp. 1–8.

[86] V. Talla, B. Kellogg, S. Gollakota, and J. R. Smith, *Battery-free cellphone,* Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **1**, 25 (2017).

[87] Z. Manchester, *Kicksat,* `https://www.kickstarter.com/projects/zacinaction/kicksat-your-personal-spacecraft-in-space` (2017), last accessed: Jul. 7, 2017.

[88] J. A. Paradiso and M. Feldmeier, *A compact, wireless, self-powered pushbutton controller,* in *International Conference on Ubiquitous Computing* (Springer, 2001) pp. 299–304.

[89] M. E. Karagozler, I. Poupyrev, G. K. Fedder, and Y. Suzuki, *Paper generators: harvesting energy from touching, rubbing and sliding,* in *Proceedings of the 26th annual ACM symposium on User interface software and technology* (ACM, 2013) pp. 23–30.

[90] S. S. Baghsorkhi and C. Margiolas, *Automating efficient variable-grained resiliency for low-power iot systems,* in *Proceedings of the 2018 International Symposium on Code Generation and Optimization* (ACM, 2018) pp. 38–49.

[91] A. Colin, E. Ruppel, and B. Lucia, *A reconfigurable energy storage architecture for energy-harvesting devices,* in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18 (ACM, New York, NY, USA, 2018) pp. 767–781.

[92] A. Gomez, L. Sigrist, M. Magno, L. Benini, and L. Thiele, *Dynamic energy burst scaling for transiently powered systems,* in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe* (EDA Consortium, 2016) pp. 349–354.

[93] J. Hester, T. Scott, and J. Sorber, *Ekho: realistic and repeatable experimentation for tiny energy-harvesting sensors,* in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems* (ACM, 2014) pp. 330–331.

[94] A. Colin, G. Harvey, B. Lucia, and A. P. Sample, *An energy-interference-free hardware-software debugger for intermittent energy-harvesting systems,* (ACM, 2016) pp. 577–589.

[95] H. Aantjes, A. Y. Majid, P. Pawełczak, J. Tan, A. Parks, and J. R. Smith, *Fast downstream to many (computational) rfids,* in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications* (IEEE, 2017) pp. 1–9.

[96] 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*, edited by W. Weber, J. M. Rabaey, and E. Aarts (Springer, Berlin, Germany, 2005) pp. 115–148.

[97] A. Dunkels, B. Gronvall, and T. Voigt, *Contiki-a lightweight and flexible operating system for tiny networked sensors,* in *29th annual IEEE international conference on local computer networks* (IEEE, 2004) pp. 455–462.

[98] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, *A dynamic operating system for sensor nodes,* in *Proceedings of the 3rd international conference on Mobile systems, applications, and services* (ACM, 2005) pp. 163–176.

[99] L. Gu and J. A. Stankovic, *t-kernel: Providing reliable os support to wireless sensor networks,* in *Proceedings of the 4th international conference on Embedded networked sensor systems* (ACM, 2006) pp. 1–14.

[100] M. Buettner, B. Greenstein, and D. Wetherall, *Dewdrop: an energy-aware runtime for computational rfid,* in *Proc. {USENIX} NSDI* (2011) pp. 197–210.

[101] P. Zhang, D. Ganesan, and B. Lu, *Quarkos: Pushing the operating limits of micropowered sensors,* in *Proceedings of the 14th USENIX conference on Hot Topics in Operating Systems* (USENIX Association, 2013) pp. 7–7.

[102] R. Vullers, R. van Schaijk, I. Doms, C. Van Hoof, and R. Mertens, *Micropower energy harvesting,* Solid-State Electronics **53**, 684 (2009).

[103] F. Yildiz, *Potential ambient energy-harvesting sources and techniques.* Journal of technology Studies **35**, 40 (2009).

[104] K. Maeng and B. Lucia, *Supporting peripherals in intermittent systems with just-in-time checkpoints,* in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (ACM, 2019) pp. 1101–1116.

[105] Y.-C. Lin, P.-C. Hsiu, and T.-W. Kuo, *Autonomous i/o for intermittent iot systems,* in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)* (IEEE, 2019) pp. 1–6.

[106] A. Branco, L. Mottola, M. H. Alizai, and J. H. Siddiqui, *Intermittent asynchronous peripheral operations,* in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (2019) pp. 55–67.

[107] A. Rodriguez Arreola, D. Balsamo, G. Merrett, and A. Weddell, *Restop: Retaining external peripheral state in intermittently-powered sensor systems,* Sensors **18**, 172 (2018).

[108] B. Ransford and B. Lucia, *Nonvolatile memory is a broken time machine,* in *Proceedings of the workshop on Memory Systems Performance and Correctness* (ACM, 2014) p. 5.

[109] S. T. Sliper, D. Balsamo, N. Nikoleris, W. Wang, A. S. Weddell, and G. V. Merrett, *Efficient state retention through paged memory management for reactive transient computing,* in *Proceedings of the 56th Annual Design Automation Conference 2019* (ACM, 2019) p. 26.

[110] T. D. Verykios, D. Balsamo, and G. V. Merrett, *Selective policies for efficient state retention in transiently-powered embedded systems: Exploiting properties of nvm technologies,* Sustainable Computing: Informatics and Systems **22**, 167 (2019).

[111] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawełczak, and J. Hester, *Ink: Reactive kernel for tiny batteryless sensors,* in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (ACM, 2018) pp. 41–53.

[112] K. Ma, X. Li, K. Swaminathan, Y. Zheng, S. Li, Y. Liu, Y. Xie, J. J. Sampson, and V. Narayanan, *Nonvolatile processor architectures: Efficient, reliable progress with unstable power,* IEEE Micro **36**, 72 (2016).

[113] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. O. Eversmann, *An 82a/mhz microcontroller with embedded feram for energy-harvesting applications,* in *2011 IEEE International Solid-State Circuits Conference* (2011) pp. 334–336.

[114] S. Khanna, S. C. Bartling, M. Clinton, S. Summerfelt, J. A. Rodriguez, and H. P. McAdams, *An fram-based nonvolatile logic mcu soc exhibiting 100% digital state retention at* vdd = *0 v achieving zero leakage with < 400-ns wakeup time for ulp applications,* IEEE Journal of Solid-State Circuits **49**, 95 (2014).

[115] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang, *A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops,* in *2012 Proceedings of the ESSCIRC (ESSCIRC)* (IEEE, 2012) pp. 149–152.

[116] S. Senni, L. Torres, G. Sassatelli, and A. Gamatie, *Non-volatile processor based on mram for ultra-low-power iot devices,* ACM Journal on Emerging Technologies in Computing Systems (JETC) **13**, 17 (2017).

[117] Y. Liu, Z. Wang, A. Lee, F. Su, C. Lo, Z. Yuan, C. Lin, Q. Wei, Y. Wang, Y. King, C. Lin, P. Khalili, K. Wang, M. Chang, and H. Yang, *4.7 a 65nm reram-enabled nonvolatile*

processor with 6Œ reduction in restore time and 4Œ higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic, in *2016 IEEE International Solid-State Circuits Conference (ISSCC)* (2016) pp. 84–86.

[118] F. Su, Y. Liu, Y. Wang, and H. Yang, *A ferroelectric nonvolatile processor with 46µs system-level wake-up time and 14µs sleep time for energy harvesting applications,* IEEE Transactions on Circuits and Systems I: Regular Papers **64**, 596 (2016).

[119] H. Stockman, *Communication by means of reflected power,* Proceedings of the IRE **36**, 1196 (1948).

[120] K. Finkenzeller, *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication* (John Wiley & Sons, 2010).

[121] P. V. Nikitin, K. Rao, and S. Lam, *Rfid paperclip tags,* in *2011 IEEE International Conference on RFID* (IEEE, 2011) pp. 162–169.

[122] P. V. Nikitin, S. Ramamurthy, R. Martinez, and K. Rao, *Passive tag-to-tag communication,* in *2012 IEEE International Conference on RFID (RFID)* (IEEE, 2012) pp. 177–184.

[123] J. Ryoo, J. Jian, A. Athalye, S. R. Das, and M. Stanaćević, *Design and evaluation of bttn: A backscattering tag-to-tag network,* IEEE Internet of Things Journal **5**, 2844 (2018).

[124] A. N. Parks, A. Liu, S. Gollakota, and J. R. Smith, *Turbocharging ambient backscatter communication,* in *ACM SIGCOMM Computer Communication Review*, Vol. 44 (ACM, 2014) pp. 619–630.

[125] J. F. Ensworth and M. S. Reynolds, *Every smart phone is a backscatter reader: Modulated backscatter compatibility with bluetooth 4.0 low energy (ble) devices,* in *2015 IEEE International Conference on RFID (RFID)* (IEEE, 2015) pp. 78–85.

[126] B. Kellogg, A. Parks, S. Gollakota, J. R. Smith, and D. Wetherall, *Wi-fi backscatter: Internet connectivity for rf-powered devices,* ACM SIGCOMM Computer Communication Review **44**, 607 (2015).

[127] B. Kellogg, V. Talla, S. Gollakota, and J. R. Smith, *Passive wi-fi: Bringing low power to wi-fi transmissions,* in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)* (2016) pp. 151–164.

[128] D. Bharadia, K. R. Joshi, M. Kotaru, and S. Katti, *Backfi: High throughput wifi backscatter,* ACM SIGCOMM Computer Communication Review **45**, 283 (2015).

[129] *Ink website,* `https://github.com/tudssl/ink` (2018), last accessed: Sep. 20, 2018.

[130] J. Hester, L. Sitanayah, and J. Sorber, *Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors,* in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* (ACM, 2015) pp. 5–16.

[131] J. Hester, N. Tobias, A. Rahmati, L. Sitanayah, D. Holcomb, K. Fu, W. P. Burleson, and J. Sorber, *Persistent clocks for batteryless sensing devices,* ACM Transactions on Embedded Computing Systems (TECS) **15**, 77 (2016).

[132] T. Instruments, *Msp430fr5969 launchpad development kit,* `http://www.ti.com/tool/msp-exp430fr5969` (2015), last accessed: Apr. 30, 2018.

[133] Sparkfun, *Analog devices ADXL345 breakout board,* `https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf` (2009), last accessed: Apr. 1, 2018.

[134] Adafruit, *Silicon SPW2430HR5H-B MEMS microphone breakout board (SPW2430),* `https://www.adafruit.com/product/2716` (2016), last accessed: Apr. 1, 2018.

[135] Saleae, *Saleae logic pro 16 analyzer,* `http://downloads.saleae.com/specs/Logic+Pro+16+Data+Sheet.pdf` (2017), last accessed: Mar. 30, 2018.

[136] J. Hester, T. Scott, and J. Sorber, *Ekho: realistic and repeatable experimentation for tiny energy-harvesting sensors,* in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems* (ACM, 2014) pp. 330–331.

[137] IXYS, *IXOLAR high efficiency SolarBIT solar panel,* `http://www.ti.com/lit/ug/tidu383/tidu383.pdf` (2011), last accessed: Apr. 2, 2018.

[138] A. Dementyev, H.-L. C. Kao, I. Choi, D. Ajilo, M. Xu, J. A. Paradiso, C. Schmandt, and S. Follmer, *Rovables: Miniature on-body robots as mobile wearables,* in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (ACM, 2016) pp. 111–120.

[139] M. Le Goc, L. H. Kim, A. Parsaei, J.-D. Fekete, P. Dragicevic, and S. Follmer, *Zooids: Building blocks for swarm user interfaces,* in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (ACM, 2016) pp. 97–109.

[140] J. Y. Kim, T. Colaco, Z. Kashino, G. Nejat, and B. Benhabib, *mroberto: A modular millirobot for swarm-behavior studies,* in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2016) pp. 2109–2114.

[141] D. Pickem, M. Lee, and M. Egerstedt, *The gritsbot in its natural habitat-a multi-robot testbed,* in *2015 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2015) pp. 4062–4067.

[142] M. Rubenstein, C. Ahler, and R. Nagpal, *Kilobot: A low cost scalable robot system for collective behaviors,* in *2012 IEEE International Conference on Robotics and Automation* (IEEE, 2012) pp. 3293–3298.

[143] R. Brühwiler, B. Goldberg, N. Doshi, O. Ozcan, N. Jafferis, M. Karpelson, and R. J. Wood, *Feedback control of a legged microrobot with on-board sensing,* in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2015) pp. 5727–5733.

[144] K. Schaper, *Intermittently-powered robot website,* https://github.com/tudssl/iprobot (2018), last accessed: Sep. 20, 2018.

[145] K. Schaper, *Transiently-powered Battery-free Robot*, Master thesis, Delft University of Technology, Delft, The Netherlands (2017).

[146] Texas Instruments, Inc., *MSP-EXP430FR5969 launchpad,* http://www.ti.com/tool/MSP-EXP430FR5969, last accessed: Apr. 13, 2018.

[147] Texas Instruments, *Ultra low power management IC, boost charger nanopowered buck converter evaluation module,* (2018).

[148] IXYS Corporation, *IXOLAR™ high efficiency SLMD121H04L solar module,* (2018).

[149] Texas Instruments Inc., *Overview for MSP430FRxx FRAM,* http://ti.com/wolverine (2014), last accessed: Jul. 10, 2017.

[150] RFMAX, *S9028 Technical Specification,* http://rfid.atlasrfidstore.com/hubfs/Tech_Spec_Sheets/RFMAX/ATLAS_RFMax_S9028.pdf (2015), last accessed: Apr. 13, 2018.

[151] TI Inc., *Ti gcc,* http://www.ti.com/tool/MSP430-GCC-OPENSOURCE (2018), last accessed: Apr. 17, 2018.

[152] A. Y. Majid, D. Carlo, K. Maeng, A. Ytidtnm, Colin, K. Sinan, P. Pawetczak, and B. Lucia, *Coala website,* https://github.com/TUDSSL/Coala (2018), last accessed: Oct. 12, 2019.

[153] J. P. Aditya and M. Ferdowsi, *Comparison of nimh and li-ion batteries in automotive applications,* in *2008 IEEE Vehicle Power and Propulsion Conference* (IEEE, 2008) pp. 1–6.

[154] R. Bloom, M. Zamalloa, and C. Pai, *Luxlink: Creating a wireless link from ambient light,* (2019).

[155] A. Giusti, A. L. Murphy, and G. P. Picco, *Decentralized scattering of wake-up times in wireless sensor networks,* in *European Conference on Wireless Sensor Networks* (Springer, 2007) pp. 245–260.

[156] TI Inc., *EXP430FR5994 launchpad development kit),* http://www.ti.com/tool/MSP-EXP430FR5994 (2020).

[157] Impinj Inc., *Impinj speedway R1000 RFID reader data sheet,* https://cdn.barcodesinc.com/themes/barcodesinc/pdf/Impinj/speedway.pdf (2015), last accessed: Jul. 27, 2017.

[158] A. R. Store, *RFMAX circular polarity S9028PCRJ RFID panel antenna,* https://www.atlasrfidstore.com/rfmax-s9028pcrj-s8658prj-rhcp-indoor-rfid-antenna-fcc-etsi (2016), last accessed: Apr. 30, 2018.

[159] pui audio, vesper, *Pmm-3738-vm1010-r: A zeropower listening piezoelectric mems microphone,* (2019).

[160] Monsoon Solutions Inc., *High voltage power monitor,* (2018).

[161] G. Hopper and R. Adhami, *An FFT-based Speech Recognition System,* Journal of the Franklin Institute **329**, 555 (1992).

[162] P. Senin, *Dynamic time warping algorithm review,* Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA **855**, 40 (2008).

[163] JBL, *JBL Go+ bluetooth speaker,* (2019).

[164] A. Amjad Yousef Majid, M. Jansen, G. Ortas Delgado, K. Yildirim, and P. Pawelczak, *Multi-hop backscatter tag-to-tag networks, INFOCOM 2019-IEEE Conference on Computer Communications,* **2019** (2019).

[165] P. Zhang, M. Rostami, P. Hu, and D. Ganesan, *Enabling practical backscatter communication for on-body sensors,* in *Proceedings of the 2016 ACM SIGCOMM Conference* (ACM, 2016) pp. 370–383.

[166] *Hardware/software source files of T2T backscatter network,* (2019).

[167] Y. Karimi, A. Athalye, S. R. Das, P. M. Djurić, and M. Stanaćević, *Design of a backscatter-based tag-to-tag system,* in *2017 IEEE International Conference on RFID (RFID)* (IEEE, 2017) pp. 6–12.

[168] Keysight, *Advanced Design System website,* (2018).

[169] Cadence Design Systems, *OrCAD Capture application website,* (2018).

[170] C. Yang, J. Gummeson, and A. Sample, *Riding the airways: Ultra-wideband ambient backscatter via commercial broadcast systems,* in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications* (IEEE, 2017) pp. 1–9.

[171] Hewlett-Packard Co., *Impedance matching techniques for mixers and detectors (application note 963),* (1980).

[172] J. D. Griffin and G. D. Durgin, *Complete link budgets for backscatter-radio and rfid systems,* IEEE Antennas and Propagation Magazine **51**, 11 (2009).

[173] P. N. Alevizos, K. Tountas, and A. Bletsas, *Multistatic scatter radio sensor networks for extended coverage,* IEEE Transactions on Wireless Communications **17**, 4522 (2018).

[174] Texas Instruments, *Ultra low power harvester power management IC with boost charger, and nanopower buck converter,* (2018).

[175] Texas Instruments, *MSP430FR5969 launchpad development kit,* (2018).

[176] ETSI, *Radio frequency identification equipment operating in the band 865 MHz to 868 MHz with power levels up to 2 W and in the band 915 MHz to 921 MHz with power levels up to 4 W,* (2016).

[177] u blox, *Sara-n2 power-optimized nb-iot (lte cat nb1) modules data sheet,* (2018).

[178] L. Chen, S. Cool, H. Ba, W. Heinzelman, I. Demirkol, U. Muncuk, K. Chowdhury, and S. Basagni, *Range extension of passive wake-up radio systems through energy harvesting,* in *2013 IEEE International Conference on Communications (ICC)* (IEEE, 2013) pp. 1549–1554.

[179] K. S. Yıldırım, H. Aantjes, A. Y. Majid, and P. Pawełczak, *On the synchronization of intermittently powered wireless embedded systems,* (2016).

[180] P. Joseph, H. Jason, and C. David, *Versatile low power media access for wireless sensor networks,* in *Proceedings of the 2nd international conference on Embedded networked sensor systems* (2004) pp. 95–107.

LIST OF PUBLICATIONS AND CONTRIBUTIONS

TOPIC-RELATED PUBLICATIONS INCLUDED IN THE DISSERTATION

1. S. K. Yıldırım, **A. Y. Majid**, D. Patoukas, K. Schaper, P. Pawełczak, J. Hester, *InK: Reactive Kernel for Tiny Batteryless Sensors*, in Proc. ACM SenSys (Nov. 3-5) 2018, Shenzhen, China (Chapter 3).
   I contributed to the InK's architecture design and to the experiments design and implementation. Furthermore, I was a project lead and co-supervisor of a master student (Koen Schaper) who worked on a case study application (a transiently powered robot). Finally, I was on the writing team.

2. **A. Y. Majid**, C. Delle Donne, K. Maeng, A. Colin, S. K. Yıldırım, P. Pawełczak, Brandon Lucia, *Dynamic Task-Based Intermittent Execution for Energy-Harvesting Devices*, ACM Transactions on Sensor Networks, 2019 [accepted for publication] (Chapter 4).
   I proposed the main ideas of the system (these ideas, however, have been sharpened by constant feedback from co-authors). I implemented the system, conducted the experiments, and collected, analyzed, and plotted the data [a master student (Carlo Delle Donne] improved the last version of the system and conducted the final set of the experiments). Finally, I was on the writing team of the paper.

3. **A. Y. Majid**, P. Schilder, K. Langendoen, *Continuous Sensing on Intermittent Power*, in Proc. ACM/IEEE IPSN (Apr. 21-24) 2020, Sydney, Australia, (Chapter 5).
   I proposed the original idea of the project and co-supervised a master student (Patrick Schilder) who developed a voice recognition code for embedded systems. Also, I developed the simulation and the mathematical models presented in the paper. Additionally, I designed the experiments and analyzed, processed, and plotted the data. Finally, I was on the writing team of the paper.

4. **A. Y. Majid**, M. Jansen, G. Delgado, S. K. Yıldırım, P. Pawełczak, *Multi-hop Backscatter Tag-to-Tag Networks*, in Proc. IEEE INFOCOM (29 Apr.-2 May) 2019, Paris, France (Chapter 6).
   I contributed to the development of the firmware code of the tags. For that, I developed a specialized debugging systems to test the implementation of the protocol and the performance of the network. The debugging tool consisted of USRP, GNURadio software, and Python code. I set a new research goal and designed and conducted the initial set of experiments (a master student [Guillermo Ortas Delgado] repeated these experiments to collect the final set of data). I reviewed and corrected the mathematical analysis included in the paper. Finally, I was on the writing team of the paper.

TOPIC-RELATED PUBLICATIONS NOT INCLUDED IN THE DISSERTATION

1. H. Aantjes, **A. Y. Majid**, P. Pawełczak, J. Tan, A. Parks, J. R. Smith, *Fast Downstream to Many (Computational) RFIDs*, in Proc. IEEE INFOCOM (May 1-4) 2017, Atlanta, GA, USA.
2. H. Aantjes, **A. Y. Majid**, P. Pawełczak, *A Testbed for Transiently Powered Computers*, in Proc. ASPLOS Hilariously Low Power Computing Workshop (Apr. 2) 2016, Atlanta, GA, USA.
3. S. K. Yıldırım, H. Aantjes, P. Pawełczak, **A. Y. Majid**, *On the Synchronization of Computational RFIDs*, IEEE Transactions on Mobile Computing, vol. 18, no. 9, pp. 2147-2159 (Sept. 1) 2019.
4. D. Patoukas, S. K. Yıldırım, **A. Y. Majid**, J. Hester, P. Pawełczak, *Feasibility of Multitenancy on Intermittent Power*, ACM ENSsys Workshop (Nov. 4) 2018, Shenzhen, China.
5. C. Delle Donne, S. K. Yıldırım, **A. Y. Majid**, J. Hester, P. Pawełczak, *Backing Out of Backscatter for Intermittent Wireless Networks*, ACM ENSsys Workshop(Nov. 4) 2018, Shenzhen, China.