Delft University of Technology
Master’s Thesis in Embedded Systems

Intermittent Kernel: A First Attempt

Dimitrios Patoukas

Intermittent Kernel: A First Attempt

Master's Thesis in Embedded Systems

Embedded Software Section
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Dimitrios Patoukas
patoukas@gmail.com, d.patoukas@student.tudelft.nl

8th November 2018

Author
Dimitrios Patoukas (patoukas@gmail.com, d.patoukas@student.tudelft.nl)
Title
Intermittent Kernel: A First Attempt
MSc presentation
15 November 2018

Abstract

Energy harvesting and battery-free sensing devices show great promise for revolutionizing computing in every known area while expanding to non-conventional use-cases. The promise of cheap, dense, and ubiquitous sensing technology brings new applications for the Internet of Things. However, the future programming model is blurry and complex. With a potential for trillions of devices, and thousands of devices per person on earth, programming languages and associated operating systems must be usable, flexible, and resource efficient. Although transiently powered computing is an area of ample research, no model presented so far has been widely adopted, hindering widespread use. Because of the thousands of applications and differences in requirements, a kernel that abstracts the intricacies of intermittency may be a part of the solution. This thesis explores key concepts that push intermittent systems closer to traditional embedded programming while examining resources costs, feasibility, and motivation for a kernel for intermittent systems. As a result a novel persistent micro-kernel is presented which supports multi-tenancy of applications, a persistent scheduler, networking and remote updating. In the experiments performed it was observed that predictive value-based scheduling can be up to 40% better in terms of total throughput, applications successfully completed per time unit, and that a node can be updated remotely with maximum speed of 9.13 bytes/s. In the thesis the full system design of the micro-kernel is presented along with quantified results for its performance.

Preface

This thesis presents the final step towards obtaining my Master's degree in Embedded Systems from Delft University of Technology.

I would like to thank my supervisor Przemysław Pawełczak for his support and patience throughout this project on both personal and academic matters. I also would like to thank Amjad Majid for his continuous support and engagement. Moreover, I would like to thank Sinan Yıldırım for the trust he showed me while working for the publication of InK. Additionally, I want to thank Prof. Koen Langendoen for hosting me at the Embedded Systems group and Maurício Aniche for participating as member of my graduation committee.

Most importantly I would like to thank my good friend Carlo Delle Donne for being a close companion throughout this journey of my life and my family for their unending support.

Dimitris Patoukas

Delft, The Netherlands
8th November 2018

Contents

Chapter 1

Introduction

Ubiquitous computing and ambient intelligence demand un-tethered nodes that can sense and compute in deeply embedded applications. However, the major constraint of un-tethered nodes is their finite battery capacity. This bounds the lifetime of the deployed device, not only hindering deployment flexibility, but also introducing additional cost and complexity of recharging, replacing and maintaining batteries. Although low power processors and new battery materials are increasing the lifespan of embedded systems, energy density does not scale exponentially like computing power traditionally has, but linearly [1] bounding the future of battery deployed nodes. Luckily, research progress on energy harvesting [2] (converting ambient energy into electrical current powering embedded devices) and transient computing [3] (executing computing tasks on embedded device that frequently loses power due to unpredictable energy harvesting source) show the path to a new era of energy-neutral battery-less computers.

1.1 Intermittent Execution

One of the main areas where battery-free devices could have immediate impact is in self-sustainable sensor nodes [4]. The main components of such a system are: an energy harvester that will convert ambient energy (solar, RF, kinetic, thermal etc.) into electric current, low-consumption computing and sensing electronics along with the needed peripherals and a low-consumption communication module. Currently a number of devices that can operate on harvested energy exist in the market like: the TI MSP430 series [5], WISP [6] with a number of different variants and applications [6, 7, 8, 9, 10], as well as commercial platforms like Pyros [11] and Bio-Thermo [12].

Harvested energy is used to fill a typically small energy buffer which is expended once the capacitance reaches a certain level. Charging times are generally much slower from discharging times, at the same time ambient energy is uncertain and harvested energy usually suffers from low conversion

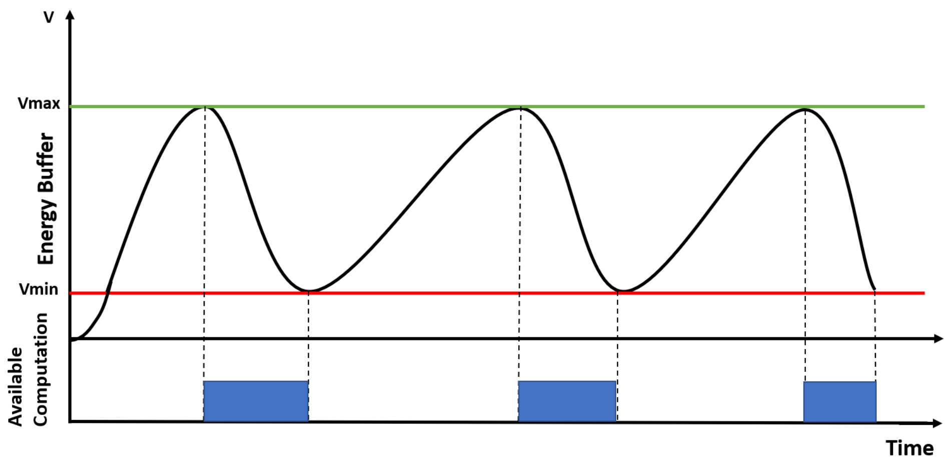

Figure 1.1: Intermittent execution during energy buffer discharge.

rates. As a result, harvester powered nodes experience frequent power interrupts (see Figure 1). The frequency of the power outages depends on various factors, for example: the size of the energy buffer, current consumption, ambient energy available and harvester conversion rate. Thus harvesting systems operate intermittently, computing for the operating period where energy is available and remain off while recharging [13, 14, 15, 16]. The software that is hosted on such systems runs in the same intermittent fashion changing the classical paradigm of line-by-line code execution. Once the node loses power it resets, losing the contents of memory, the state of the registers and the program counter, forcing the running code to start from the beginning the next time it is charged.

Although energy harvesting has been explored extensively we can see that widespread adoption of micro-controllers powered by such systems is still lacking. The reality being that developing reliable applications for systems that experience random power interrupts is cumbersome and can be proven extremely tricky. Developing cost increases as the programmer has to address the two key issues that are inherent to the nature of the system, namely:

1. **Forward Progress**: Ensuring that the executing program will resume operation from the latest safe point of execution allowing for the executing program to complete.

2. **Data Consistency**: Ensuring that the data processed and written will be kept consistent between power failures and the memory space will accurately reflect the proper state of execution.

Additionally, key concepts of embedded software like timekeeping and interrupts need to be re-addressed. Timekeeping without additional hardware is impossible in the traditional sense, as with every energy outage the content of the registers clear and no information is retained. Interrupts can break the memory model as modifying the contents of global memory without proper versioning can have a detrimental effect to data consistency.

An ample body of work is currently tackling the intricacies of intermittent computing, for example InK [17], Alpaca [15], Chain [18], DINO [13], Hibernus [19, 20] and Mementos [21]. A number of solutions have been proposed that ensure forward progress of program execution and maintain data consistency. Each system takes a different approach providing mixed results on performance, features and design constraints. These systems focus on designing *monolithic software systems* that are being deployed under intermittent conditions with no guarantee over their performance or longevity. They present simplistic programming models that confine the creation space for modern software as support for key features is lacking.

1.2 Problem Statement

The trend of software development mandates that hardware acts as a hosting node where multiple actors develop and operate software to different ends, leaving monolithic designs behind and embracing multi-tenancy. This is demonstrated in maturity of computing systems from sensors, phones, desktops, and datacenters. Much like other compute paradigms, the process and resource management *as well as scheduling* have a significant impact on the behavior, efficiency, and quality of application of deployed battery-less embedded systems. However, the design space for enabling multi-tenancy on intermittent computers has not been explored.

Multi-tenant systems allow multiple software constructs to operate on the same platform sharing the same resources. When software is executed, system constraints might force tasks to wait for I/O operations or interaction with other peripherals (ADC, DAC, sensors, actuators, communication modules etc.). In the case of embedded sensor nodes interaction with the environment is crucial. As such, stable powered nodes are usually designed to respond to events by interrupts which trigger the appropriate computations or by constantly polling the input channels. In intermittent systems such a capability does not exist as the system is inherently out of sync with the environment.

As an example consider a wearable motion-sensor node which is triggered by an interrupt once the detected activity surpasses an acceleration threshold. In a stable powered node once the interrupt occurs the node samples the movement for a time window and then process the data. In intermittent systems if a power failure occurs after the interrupt, sampling for movement might be irrelevant as the movement window has passed when energy becomes available again. As a result the motion sensing has to wait for the event to happen again to capture it. On this downtime, another application could be executed exploiting the energy availability while the motion sensing task waits for an input. This thesis theorizes that by allowing multiple tenants to operate on the same node we increase the chance to utilize more of the available harvested energy, increasing the overall performance and responsiveness of the system.

Furthermore, self-sustaining systems can be best exploited in applications where human access is limited or cumbersome. Sensor nodes can be placed in remote environments [22], on insects [23], or be embedded in building materials [24]. For example, in the case of sensor nodes embedded in building infrastructure use-cases and requirements might change multiple times throughout the life-cycle of the building. As a result multiple operators may want to modify, maintain, update or enrich the software that is running on this unaccessible node. It becomes apparent that being able to run and update different applications of remote nodes fits the current direction of the moder IoT.

This thesis examines the feasibility of a system that can ensure that a transient node can operate reliably with scalable software while being able to inter-connect and be updated as modern practices mandate.

Therefore, the research question this work addresses is:

Is it possible for a transiently powered node to host, operate and update multiple applications effectively?

1.3 Contributions

Current programming models for intermittent nodes only deal with abstracting the difficulties of transient computing for single execution programs not addressing aspects that will allow the widespread adoption of transient computers. This research explores the feasibility of a rudimentary operating system, allowing multi-tenacy of applications, interrupt handling, persistent scheduling and networking.

The list of contributions is as follows:

1. The design of a battery-free system that hosts multiple applications and ensures their operation under power failures.
2. The design of a battery-free system that can be updated by a novel tag to tag network.

1.4 Thesis Outline

The rest of the thesis is organized as follows: Chapter 2 provides background information and 3 introduces the related work. The design of the intermittent kernel along with the software implementation and programming model are presented in Chapter 4. This is followed by a performance evaluation of the intermittent kernel in Chapter 5. Finally, Chapter 6 concludes this thesis and proposes potential future work.

Chapter 2

Preliminaries

As mentioned in Chapter 1, research in the domain of energy harvesting and as well as developments in low-power hardware and accessible FRAM controllers [25] are enabling self-sustained nodes that can operate relying solely on harvested energy.

The emergence of such technologies opens the path for a new vision for IoT devices that do not rely on battery power or their ability to be attached to a power source. Consequently battery-free devices that are capable to collect, process and transmit data relying solely on harvested energy, could be realized. In this reality not only we can monitor in spaces were it was previously impossible, like building materials, wildlife, remote and inaccessible environments [24], but we effectively reduce the energy footprint of the IoT ecosystem as part computation is offloaded to millions of harvesting nodes and not in energy expensive datacenters [26].

On a high level approach such systems contain four key elements that enable them to operate under intermittent conditions:

- energy harvester (RF, solar, kinetic, thermal etc.);
- the microprocessor along with needed peripherals;
- A capacitor needed for collecting energy from the harvester and able to hold enough energy for the micro-controller and the attached peripherals to operate for an arbitrary amount of time.
- An amount of non-volatile memory so that data can be preserved even without any available energy.

In a typical operating cycle the device is powered-on and thus computation-capable once the available energy stored in the capacitor surpasses a certain threshold. After that point, the system executes until the energy storage is depleted and the system ceases operation. After that point the system remains in off state until the energy harvester collects enough energy to surpass the set threshold [27, 28]. Harvested energy is a highly unpredictable energy supply as it depends on environmental availability [24], hardware configuration and conversion rate [29]. Consequently the amount of available computation per charge varies highly among systems. Although the size of the capacitor can lead to larger execution times at the same time charging time grows proportionally, as a result the pattern of execution can vary among systems. In small sensory nodes where form factor is considered critical, small fast charging capacitors are used leading to frequent power failures at a rate of ≈10–100 times per second [30].

Depending on hardware configuration, some systems are capable of supplying harvested energy directly to the discharging capacitor. Discharging rates are far bigger than harvesting rate in most cases as electronic component out-consume small energy buffers [31]. Moreover, extra hardware or added software logic can be used to notify the system for imminent death [32], but for small and constrained devices that this thesis examines the amount of energy and computation time that is needed for such operations is significant and our intuition is every available resource should be utilized for the computational load of the system and not for auxiliary functions.

2.1 Challenges

It becomes apparent that embedded systems that operate solely on harvested energy do not follow the traditional operational scheme and thus intermittent execution introduces a new set of challenges that need to be addressed when designing software.

2.1.1 Forward Progress

As the system experiences frequent deaths a purely sequential code execution cannot guarantee that computation will successfully terminate. At every power failure the volatile state is lost and the contents of the registers are cleared.

Consider a transiently powered node with software that takes 100 cycles to reach successful termination. If the energy provided from the energy buffer to the node is enough for a maximum of 50 cycles the code will never reach the termination point, unless some mechanism allows the program to be split in smaller pieces which can be completed with multiple expenditures of the energy budget. In this case the node will resume execution from the last non-executed piece until all are concluded (see Figure 2.1).

2.1.2 Memory Consistency

Although the state of non-volatile memory can be retained the non-sequential execution of code might lead to memory dependency hazards, most commonly write after read dependencies.

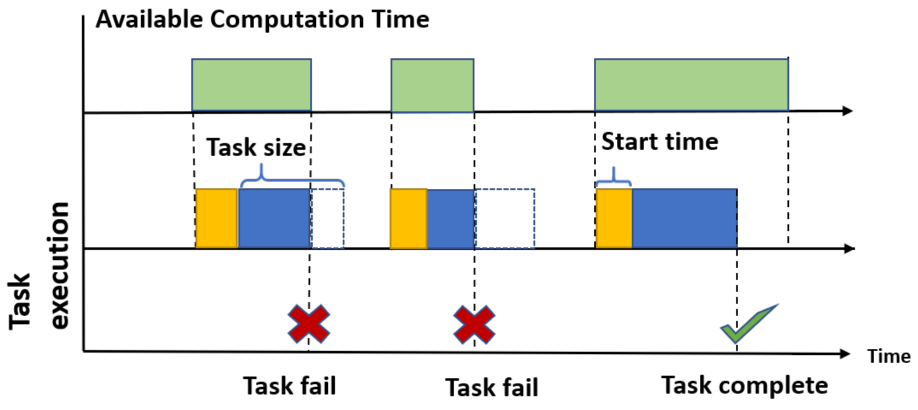

Figure 2.1: Successful completion of a task over the period of three intermittent execution cycles.

For instance, consider two operations commonly found in software: an increment operation, (`i++;`) and using the increment result as an array index (`array[i]=value;`). Now consider that a power failure occurs after `i` was incremented. When the code restarts (`i++;`) will be executed for a second time and the index will be incremented twice before writing into the array, leading to an inconsistency.

Additionally, interrupt service routines that operate on non-volatile data might create inconsistencies since the re-executing part of the code is unaware of which is the right version of data to access. Moreover if a power failure occurs while operating in the non-volatile memory the data retained might be corrupted or incomplete. Since the power outage of the microcontroller might occur at any point of its operation it is possible that a death will occur during a write operation. For example, a power failure during a register write operation might lead the register to retain either `1` or `0`. This unspecified behavior might lead to inconsistencies.

Chapter 3

Related Work

To address the issues mentioned in Section 2.1 a number of different approaches were used yielding widely different outcomes. Checkpointing of the volatile state was one remedies that tried to address the problem of forward progress, by periodically saving the state of execution to the non-volatile memory. Such programming models, e.g., Mementos [21], Hibernus [19], HarvOS [33] save the volatile in non-volatile memory to ensure that the progress of software will resume from the last saved checkpoint. Unfortunately they do not provide adequate support against inconsistencies that might occur from re-execution of code [34]. DINO [13] overcomes this problem by selectively versioning part of the non-volatile memory and other implementations achieve the same with additional hardware support [20]. By checkpointing the whole system state these runtime environments run into the danger of becoming un-scalable due to overheads. After a certain size of volatile memory the energy and time cost might overcome the available budget [17]. Similar scalable approaches include QuickRecall [35], Clank [14] and Rachet [36] which are designed for micro-controllers where all memory is non-volatile and where light checkpointing is applied on the state of the system. As a result these implementations are considered niche and not realistically applied in widely adopted market.

Other approaches use task based systems that avoid the un-sustainable overhead from checkpointing by dividing the executable program in *atomic* tasks. Alpaca [15] and Chain [18] introduced a task based control flow where the program progress to the next task only when the previous has been competed successfully. Memory consistency is guaranteed by versioning the non-volatile variables to ensure that the consistent version is used when a task restarts.

On the same note InK [17] introduced the concept of event-driven execution from traditional embedded systems to the intermittent domain, by supporting encapsulated interrupts that do not violate the consistency of the memory space.

Task based approaches appear to be superior to checkpointing in terms of scalability and overhead but they introduce new complexities.

During development the executable program is decomposed in a collection of tasks which are executed in explicit order. The software progresses only after a task has been completed within one "power-on" period. In reality complex software is not always decomposable into a set of tasks. Moreover using popular programming abstractions like pointers and recursion might not be an option when all computation has to occur within one defined task.

Additionally in order to achieve forward progress each task has to be completed. For this to happen all tasks have to be able to be executed within the available energy budget, which requires accurately profiling and sizing tasks. Large tasks run into the danger of never terminating and very small tasks significantly increase the overhead from task-switching too frequently.

On the sensory node domain extensive research has been conducted covering many of aspects of embedded nodes but without addressing intermittent execution and the complexities of harvester powered systems. Popular systems in the embedded domain like tinyOS [37], Contiki [38] and T-kernel [39] provide an introduction to operating kernels in such constrained environments. These systems apply operating system techniques to small embedded systems but their architecture if far for applicable to the transient power domain. Multi-tenancy has been explored[40] for non-intermittent systems like MantisOS [41] and SOS [42].

Updating remote sensor networks is an active research area with multiple scopes. In the area of embedded nodes various architectures have been proposed for kernels that contain update support like Tock [43] whereas others implementations propose architectures that enable dissemination in dense sensory networks [44] or transmitting code instructions in tiny capsules [45]. The scope of this research focus on systems that are are connected to a stable supply and thus cannot be adopted for intermittent nodes.

Updating transiently powered systems has been explored mainly in the scope of computational RFIDs with protocols like Wisent [46] and Stork [47] where the EPC [48] protocol is exploited to re-flash the transient node with new firmware. These systems only apply to the case of an RF powered node and they do not update parts of the system, but the entirety of it. Such an approach requires large enough memory to host the new uploaded image before the update, making it difficult for memory constrained devices. Moreover by transmitting the whole software image makes the update process longer than needed in cases where only parts of the hosted software needs to be replaced.

Networking under intermittency remains an undetermined field where no definitive solution has been presented. Traditional network stacks like BLE [49] can be deployed if the transient system can sustain enough computation time to successfully meet the protocols requirements. But those requirements are unrealistic to be met by tiny harvesting platforms. Backscatter networking [50] has been proposed as a possible solution. It uses ambient signals to pass messages between intermittent nodes. The low energy consumption of backscatter networks [51, 52] make them match the low energy profile that transient nodes need to operate successfully. No such implementation so far covers the scope of networking for intermittent systems in a definitive way and it remains an open research area.

Chapter 4

System Design

This thesis work aims to implement a system that not only addresses the key problems of intermittent execution but expands on the functionality of existing systems in a way that provides developers the space to develop real-world applications on battery-less systems.

4.1 Requirements for Intermittent Kernel

Except from abstracting the key problems of transiently powered computers (forward progress, memory hazards) we want a system that is capable of adhering to modern embedded software principles, to the extent that this is possible.

Namely, we want a kernel that is able to host multiple applications and can safely and effectively share the constrained resources of a battery-less system. Moreover is crucial for the usability and longevity of the embedded node for the kernel to be to remotely updatable either by allowing for new applications to be loaded or for older applications to be maintained. Additionally hosting multiple application in the same energy constraint platform can be proved detrimental for the hosted software from the point of performance.

In intermittent systems no guarantees can be provided either for the available energy or for the time that every application is allowed to be executed. This is an inherent problem of the executing environment since harvesting patterns can be highly unpredictable and accurate timekeeping is currently unfeasible for most systems.

The problems that arise due to those effects are:

- *Resource contention*: an application that is significantly large and occupies the current execution order is able to indefinitely hold the CPU leading to starvation. Task size plays a crucial role in the behavior of task based systems and their negative effects can have a detrimental

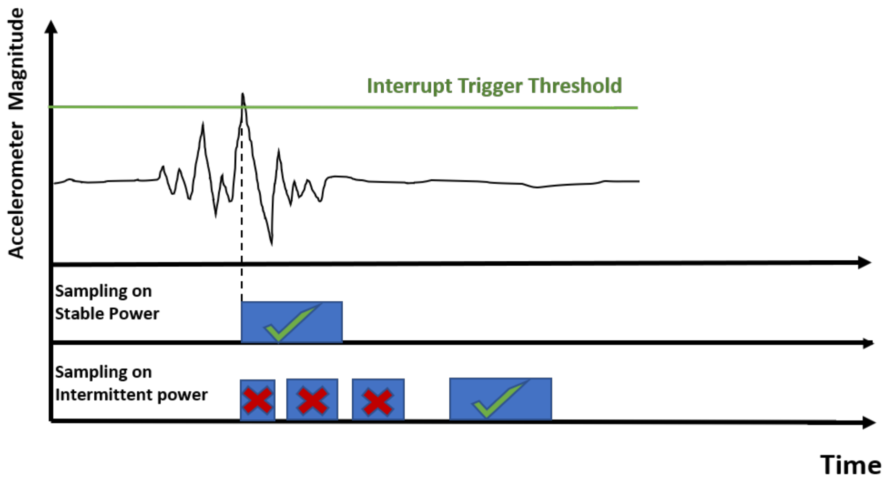

Figure 4.1: Sampling occurs out of sync of the event, leading to resource waste and incorrect sampling.

role on their performance. Large tasks tend to take more rebooting cycles to complete and could lead to delays pushing queued tasks further forcing the system to respond very late to any external input.

- *Resource waste*: an application that is extending its execution over a time where is irrelevant is wasting resources that could be used for useful computation. Embedded nodes usually interact with the environment by interrupts which are triggered from outside events. For example in activity recognition systems, the embedded accelerometers start sampling only when the acceleration overpasses a set threshold and lasts for the duration of the movement. If we adapt this example in the intermittent domain we could experience the possibility that the accelerometer triggers a measurement but since no guarantees exist for timing, sampling might occur in a moment irrelevant to the actual movement we want to observe. Moreover data may sampled which are irrelevant to our application leading to false results (see Figure 4.1). The energy expended at the same time could be used in another application or conserved for a later event, leading to a more responsive and reliable system. This problem was initially addressed in InK where interrupt service routine support for intermittent systems was found to reduce wasteful computation and as a result increase the reactivity of the system by almost 14 times [17].

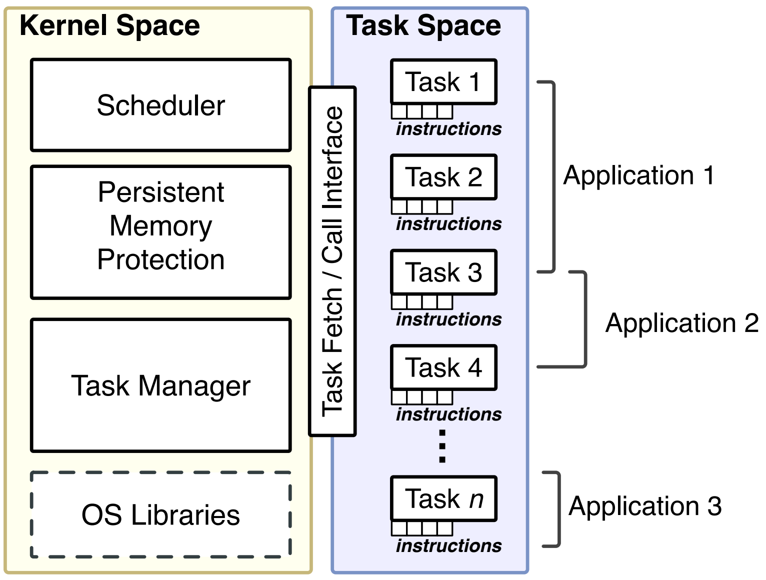

Figure 4.2: High level task based system architecture and memory space supporting multi-tenancy. Applications are made up of tasks. Tasks use OS level libraries and can be shared among applications.

4.2 Architecture

The kernel was designed with intermittency in mind. Thus to be able to guarantee forward progress and data consistency as well as address the challenges mentioned earlier a fitting execution and programing model should be adapted. The system proposed is composed of two distinct layers: (i) the micro-kernel layer and (ii) the application layer (see Figure 4.2). The layered design enables a degree of task isolation and protection as in modern operating systems and provides design space for features such network stack, time-tracking and remote update.

4.2.1 Application layer

This section contains the abstractions that enable the programmer to develop applications to be run on our kernel. Each application is defined as a graph of tasks which will be executed sequentially and in an order provided by the programmer. Tasks, and as an extension applications, are fairly easy to write as they are just C functions which accept no argument and return nothing. The declaration of such a task is done in the relevant translation unit along with the information about which application it belongs to and the order of its execution. A task can be used by different applications while keeping the application data protected from inter-application accesses. A task is atomic in the context of intermittency in the sense that in order for the system to progress to the next task the whole task has to be executed to completion, unless the scheduler evicts the task. Data consistency is preserved only for variables and arrays that are explicitly defined as protected by the appropriate abstractions. Access is also performed through relevant notation which is an abstraction layer to the Virtual Memory manager which is responsible for keeping data consistent through power failures. The number of tasks available for programming is fixed, currently 12 are supported, and their code size is bounded to a fixed size (1 KB currently). This enables the micro-kernel to be easily updatable as the scheduler knows beforehand where in memory to locate the tasks once a task is enabled. Moreover, this limit also bounds the amount of data that are transmitted over the air for a possible update.

Knowing that networking is inherently unstable and poor performing under intermittency it is preferable to have smaller packets traveling over the network to increase the probability of successful completion in a reasonable time-frame. The task code size can be altered without significant effort by modifying the linker file, although small task sizes yielded better results as it is generally easier to complete smaller tasks, especially under very frequent power failures.

4.2.2 Kernel Layer

This section of the system contains the infrastructure needed to guarantee the execution of the applications under intermittency and provide extra functionality. More specifically: scheduler, virtual memory manager, networking stack, and update system are part of the kernel.

Virtual Memory Management

The Virtual Memory Management (VMM) system handles protected variables and data sharing between tasks. The VMM abstracts the physical address space of the non-volatile memory into two regions: 'safe' and 'volatile':

- *Safe*: holds the data for which the system can guarantee their consistency.
- *Volatile*: acts as a buffer where an image of the safe data is manipulated during execution.

On compile time the available non-volatile memory address space is partitioned into the an area on which the VMM is operating. During the execution of a task, the volatile address space is populated with new data and upon successful termination the volatile buffer is committed to the Safe area where they can be accessed by the next task. If a death occurs before the successful termination, the buffers are discarded preserving the "correct" state of operating memory. When a task is activated the VMM maps the variables into the correct address space taking also application context into account. For example, if Application 1 and Application 2 are sharing `SharedTask`, when `SharedTask` attempts to access protected data under the context of Application 1 the VMM is responsible for addressing the memory space referring to that application, thus providing not only memory protection under power failures but also some memory isolation between applications. Additionally if a new task is inserted through the update mechanism the VMM maps the new protected variables in the next available address space in the partitioned memory.

Scheduler

The Scheduler operates simply by executing tasks which are recovered from a buffer, namely the ready queue. The ready queue is populated by tasks which are inserted as soon as they become available according to the scheduling policy. Tasks are executed consecutively and task preemption is allowed, for example in the case a task has exceeded its available computation time.

Persistent Scheduling

Although scheduling is a well researched topic, implementing a scheduler that performs under frequent interruptions is a challenging task. As with every intermittent program the flow of execution must respect the constrains of intermittency. In a typical scheduling scenario a process is released and posted to a queue. Once the CPU becomes available the next task in the scheduler queue is executed according to the scheduling algorithm that is specified. In the intermittent domain in every step of the scheduling process we run into the hazard of losing power. As a result a process might be released but never added to the queue because of a power interrupt. Moreover when the scheduler calculates the next task that is to be executed a power failure might incur faults in the execution order. Write after read dependencies might produce undesired and unpredicted behaviors leading to undefined behavior of the system (see Figure 4.3). The same risk is introduced by data inconsistencies that might corrupt the schedule leading the system to an unknown state.

As a result this thesis implements a persistent scheduler that tries to mitigates the aforementioned risks. The scheduling part of the kernel operates in two states:

- *Volatile schedule*: In the volatile state the scheduling module adds processes in the ready queue and calculates the executing order based on the desired algorithm;

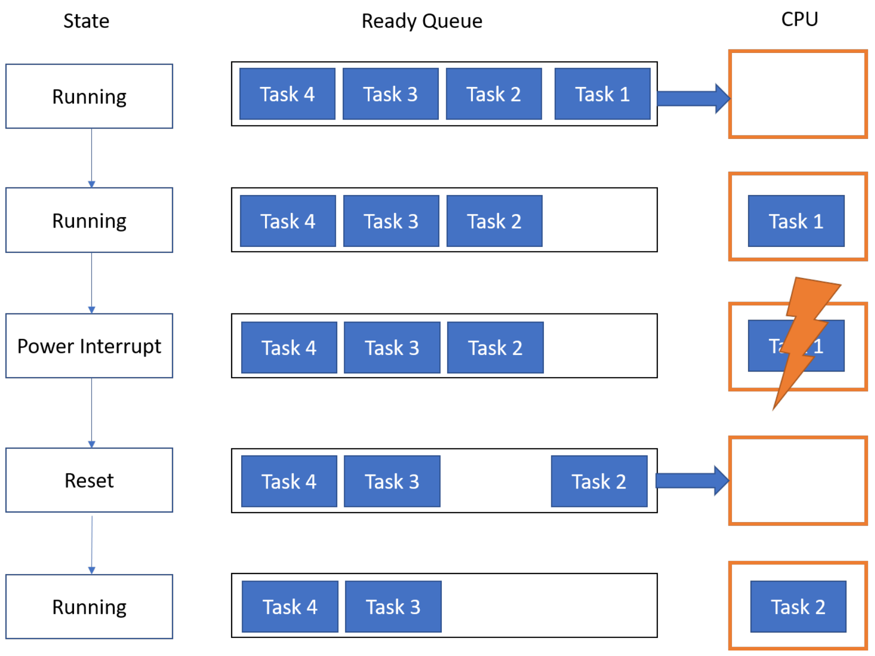

Figure 4.3: If a power interrupt occurs after a task was removed from the ready queue, after the reboot the schedule will be inconsistent. Task1 is removed without having completed.

- *Persistent schedule*: If the volatile state is concluded safely the scheduling module commits the result to a non-volatile buffer. This buffer is read only by the scheduler and written only by the commit mechanism.

By creating this level of isolation we can be reassured about the validity and consistency of our schedule. In that way we can guarantee that things that are scheduled for execution are going to be processed in the desired order. As a result selecting appropriate algorithms can lead to a stable system that will always perform on the desired manner as undefined states will be avoided by the schedule.

Scheduling Algorithms

Scheduling under intermittency is an active area of research which focuses on the optimality of schedules under uncertain executing conditions. Research mainly focus on optimality of the schedule by either applying complex prediction techniques that try to fit the executing load on the predicted available computational period or try to fit the executed load to the available stored energy. Both of those techniques cannot be easily adopted under the context of tiny battery devices that are examined in this thesis. Executing times of tens of milliseconds make the available executing budget very constrained. As a result the priority of the scheduler should be to execute the computational load and to not waste unnecessary operating time in the scheduling itself. In this context of intermittency where the operating overheads are already high due to the the persistent protection mechanisms, expensive FRAM memory accesses and low processing speeds the sustainability of such an approach is questionable. Moreover measuring the amount of available energy requires to devote resources on additional sensing which can be costly in terms of computational resources or area since extra hardware is usually required. Thus, in this thesis we approach scheduling in a naive manner where common scheduling algorithms with reasonable overhead are evaluated for their performance under transient power conditions.

One important aspect to be considered is that by the nature of intermittency, scheduling events that rely in input from the environment, namely interrupt triggered actions, are prone to missed events. As an example a sensory node which relies to an interrupt from a sensory component to initiate a measurement might miss the input event as at the time the event occurred the node could be in a non-powered state. The effects and possible solutions to this problem were presented in InK [17] where there is a strong indication that interrupt enabled computation in transient environments significantly improve the reactivity of the system to outside events. Although such an approach mitigates the problems introduced it does not provide any guarantees that the node will successfully capture all environmental events. As a result it becomes apparent that intermittent systems are operating in a *best effort* fashion which consequently applies to the implementation approach of this thesis.

The selected algorithms for the kernel presented are:

- *First Come First Serve (FCFS):* Non-preemptive scheduling of the applications as soon as they are released. FCFS was selected based on the intuition that an applications should be executed as soon as they are released as this can potential lead to lower response times. Since the execution environment is unpredictable and unstable taking the opportunity to compute as soon as possible could provide benefits to the execution of the application.

- *Shortest Job First (SJF):* Non-preemptive scheduling based on the size of each application. Following the same reasoning as in FCFS, this thesis evaluates if prioritizing applications based on their size yields improvement in the performance of the system. One of the limitations of this approach is that the size of each application has to be specified and evaluated on compile time increasing the complexity to the programmer. Additionally the scheduling overhead increases significantly as the scheduler has to involve more logic when selecting the next application in the ready queue, increasing at the same time the probability of a power failure happening during scheduling.

- *Value Based Scheduling (VBS):* Non-preemptive scheduling based on values derived by the behavior of the applications. The value of each application is increasing whenever a task is completed within one discharging cycle. Based on the same principle the value of each application is decreasing every time a task fails to complete. Prioritizing successful tasks the system increases the probability to increase the number of successful task completions. At the same time, overhead is increased as selecting the next application takes more computational time along with keeping track successful completions. Additionally, the system runs into the potential hazard to only execute high value tasks as in environments where energy becomes scarce only small task might be able to execute. As a result a small task might be indefinitely prioritized to the expense of bigger tasks that might never get the chance to execute.

- *Round Robin (RR):* Preemptive scheduling circularly switching among applications when their available time quantum expires. Although opportunistic scheduling might yield better results when the applications are able to successfully end within few execution cycles, the system faces the risk of starvation if an application is unable conclude. Round robin provides a sense of fairness as it enables applications to run for a predefined time quantum. If the application fails to conclude in the allocated time it get evicted from the execution queue and the next application is inserted. That way there is a fair allocation of computational resources to the system. Moreover, by allowing preemption the system deals with handling smaller execution units, tasks, which are easier to complete in the limited computational time that is available in transient conditions.

Network Stack

The network stack is adopted from a novel tag-to-tag network protocol developed in the TU Delft Embedded Systems group. Tag-to-tag networks intuitively are linked with intermittent nodes as they provide a highly energy efficient way of communicating by using ambient electromagnetic waves as carrier signals. In principal the protocol floods the channel with messages which are relayed by receiving nodes to effectively increase the overall range of the network. Frames have fixed length and deliver a payload of 4 bytes per frame. The operation of the network relies on ISRs to complete a successful transmission or reception and the only requirement for such is that sufficient execution time is available (Section 5.5). The details of the network implementation is beyond of the scope of this work and will be covered in future publication.

Update System

The update system module is implemented on the application layer of the tag to tag network stack. Although the current implementation of the network is not optimized for intermittent operation we implement an application capable of exploiting the flooding operation of the protocol. The update mechanism relies on the successful transmission of a task over the air. Once the new task is received it is incorporated in the application layer of the kernel and is treated as part of the executing schedule.

The transmission occurs in three stages:

1. *Stage 1*: An initial message informs the system to be updated about the incoming task. Upon reception the node pauses the execution of the schedule and allocates a buffer according to the size of the incoming task.

2. *Stage 2*: The node is now on the receiving state where it populates the buffer with the incoming messages. Since we consider intermittent operation the messages might arrive in non-serial manner, this is why each message indicates its place in the buffer, effectively reducing the usable payload of the frame from 4 to 3 bytes. After a number of sending iterations and once the receive buffer contains all the messages that are expected the system waits for a finalizing message to indicate the stop of the transmission process.

3. *Stage 3*: The finalizing message contains information about where in memory the new task will be stored and about how its going to be executed. More specifically, in which application it belongs and what is its order of execution relative to the already existing tasks. After finalization, the node returns to the normal state of execution where the schedule is processed.

It is also important to note that in this research the communication is considered to be one way, from the transmitter node to the transient receiving node. As a result the transmitter has no way to verify if the all the messages have been received and as a consequence multiple retransmissions of the same message are needed in order for one successful transmission to be completed.

Chapter 5

Evaluation

Evaluating a complex system as a whole is difficult, as no definitive answer can be easily produced that accurately encapsulates the overall performance. Moreover since no other such system exists that follows the architecture of our kernel it is not possible to quantify it in relative terms by comparing to an existing implementation.

5.1 Use Case Application

Benchmarking is performed by developing an experimental application as a model use-case for such an intermittent node. As mentioned in Chapter 1 harvester powered nodes could be potential used in deeply embedded applications in building materials. As an example case, we present a system that could be embedded inside double glass panes. These insulating panes are used in windows and are consisted from two glass panes attached with a light aluminum frame. The frame is hollow with opening of 3 mm height and 4–25 mm width (see Figure 5.1). The frame is enclosed with insulating glue, but during assembly a small sensor node can be inserted. The node can monitor the state and the environmental status of each window.

Depending on the requirements of the application the node can be powered either by a small solar panel positioned in the internal metal frame of the glazing unit or by an RF harvester. That way the sensing node can be completely disengaged from the need of a power supply. No special window frames or cabling is required as the sensor can be self sustainable. Moreover with the added update ability the node can be maintained without the need for physical access.

An update could be required changing the environmental monitor from summer to winter calibration. Additional software could be added later, if new monitoring needs occur. For example using the sound sensor to detect high wind bursts that could damage the window if it is not secured.

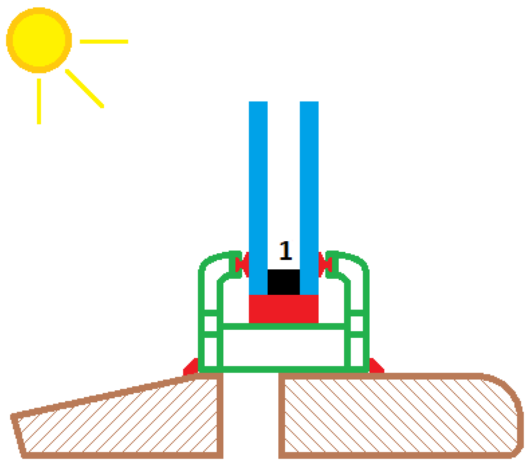

Figure 5.1: A sectioned diagram of a fixed insulating glass unit. Number 1 indicates the hollow aluminum frame.By NcLean - Own work, CC BY-SA 3.0

5.2 Testing Environment

Our prototype which consists the system under test in comprised of a TI MSP430FR5969 MCU [5] which hosts the intermittent kernel. The MCU is connected to a sound sensor, an accelerometer and environmental sensors which provide temperature and humidity values. We develop three applications:

1. Sound Detection: This application is used to detect glass breakage by sensing the ambient sound. The application is triggered by an interrupt signal from the accelerometer. The application is split in three tasks: A sampling task which used the ADC to sample 128 samples. A filtering task that performs Fourier transform on the sample. Lastly a a task which classifies the filtered sample as glass breakage or not.

2. State detection: This application is used to determine if the window is closed or open. Using an accelerometer triggered interrupt the application classifies the window in one of the two states. The first task samples the accelerometer. The second task applies averaging filter and the third task classifies the window to the correct state (open or closed).

3. Environment Sensing: This application gathers data for temperature and humidity inside the double glazing unit and determines if they have exceeded or fallen below a certain limit. That way we can monitor the thermo-insulating performance of the glazing pane and replace it once it degrades. The first task samples the sensor and collects data

for temperature and humidity. The data are sorted and the two limit values are stored. The third task classifies the sample to acceptable or not according a preset limit.

Testing the current setup in a concise and repeatable manner requires a predefined execution environment. To simulate intermittency, it is required to allow the microcontroller to execute for a limited period of time before forcing the system to reboot. To that purpose a software library was developed that could simulate the energy environment under which the system operates. In reality, harvesting environments contain a huge range of cases under which the system can be evaluated. More specifically the amount of execution time available, time spent while dead, recharge time, energy availability are variables that will change depending on hardware design, deployed environment, time of the day and a number of unaccounted factors. On compile time, an array of pseudo-random generated values define the available executing time after each reset. The timing values are produced by a python script for random number generation [53].

For each iteration of the experiment for the same scheduling policy the minimum and maximum range of the random values is modified to simulate more energy-constrained or energy-rich environments. The following cases where evaluated:

- *Normal Execution*: The node runs on stable power supply. This is the base performance of the systems and the results are used to normalize the intermittent values. Moreover from this case metrics are extracted for overheads as well as baseline application profiling when determining the size and the expected behavior of the system.

- *Intr-18*: The node experiences power interrupts at a random moment in time, allowing for execution periods in the range 1–30 ms with average period of 18.03 ms.

- *Intr-26*: The node experiences power interrupts at a random moment in time, allowing for execution periods in the range 1–50 ms with average period of 26.81 ms.

- *Intr-32*: The node experiences power interrupts at a random moment in time, allowing for execution periods in the range 1–65 ms with average period of 32.69 ms.

The interrupt that occurs after the available time expires incurs a brown-out reset (BOR) which leads to a power on reset (POR) and power up clear (PUC) which puts the micro-controller to its initial power-on state with the contents of its registers cleared [5]. After the reset the library sets the new power interrupt to occur by picking a new random value.

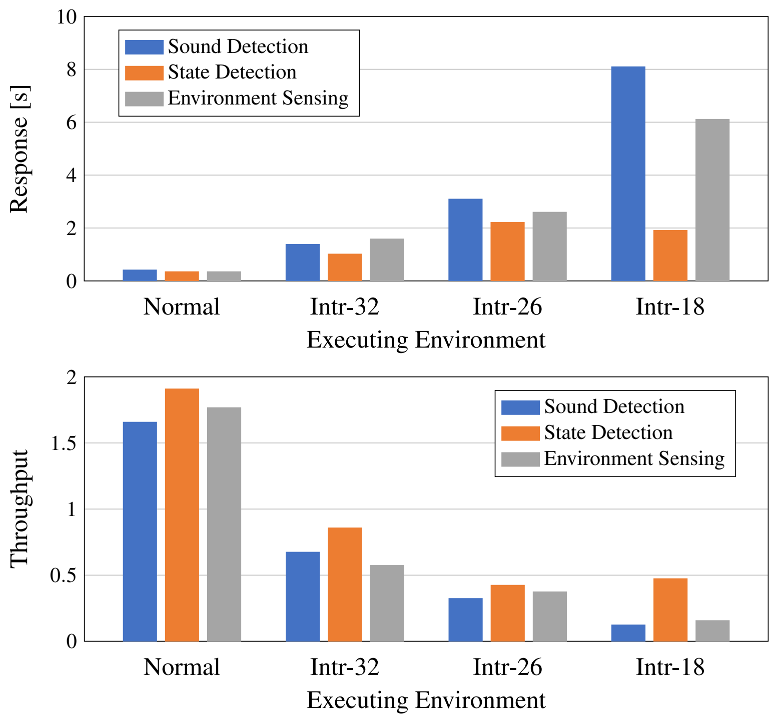

Figure 5.2: Round Robin: response time and throughput for all three hosted applications for each of the simulated environments.

5.2.1 Performance Metrics

For examining the implications of power failures in scheduling algorithms we consider the following performance metrics:

- *Response Time*: the time it takes for an application to finish from its release;
- *Throughput*: the amount of successful completions per unit of time;
- *Wasted Time*: the amount of time spent on computation that is lost due to power failures.

The results vary per different execution environment. In Figure 5.2 the different results per simulated environment are demonstrated, for all three applications in the case of Round Robin scheduling. The three simulated environments are denoted by the average on-time that the microcontroller executes. The trend that appears is that, as expected the response time of an application is increasing as the available energy becomes more scarce. Additionally throughput decreases as less applications successfully terminate per time unit.

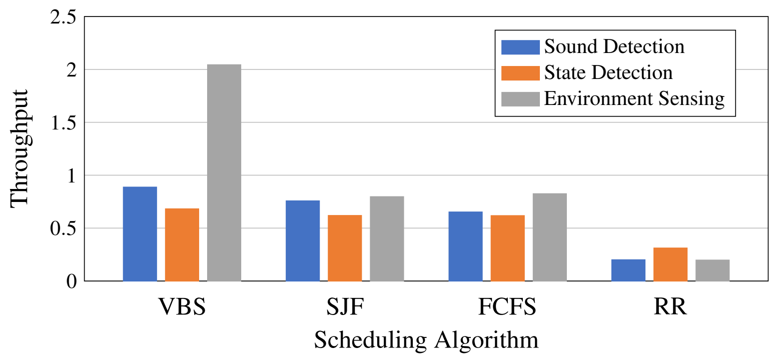

Figure 5.3: Average throughput (successful completions per time unit) per application for each evaluated algorithm.

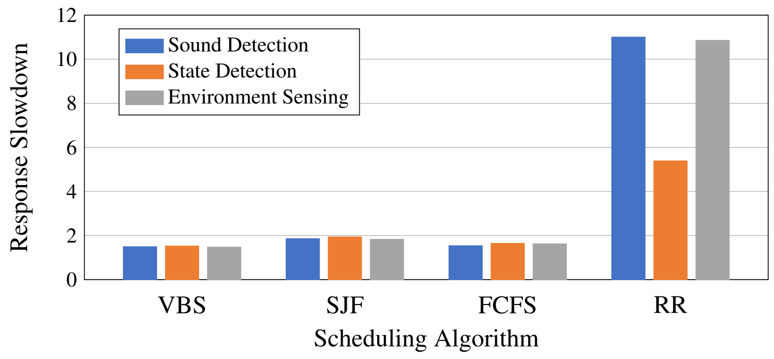

Figure 5.4: Average slowdown in response time per application for each evaluated algorithm.

5.3 Scheduling Results

The results of the three different energy environments are averaged and normalized over the non-interrupted execution. The end result provides an overview of the systems performance based on the three harvesting patterns mentioned in Section 5.2.

Attempting to better quantify the overall performance of the system we measure the overall throughput of the system as the total amount of successful application completions per time unit (Figure 5.5).

5.4 Overhead

Performance metrics provide a clear indicator for the performance of the system as is. As mentioned earlier though, overheads in intermittent operation is crucial to determine if such a system is feasible and able to scale into a more usable use-cases. As it appears in Figure 5.7 the scheduling

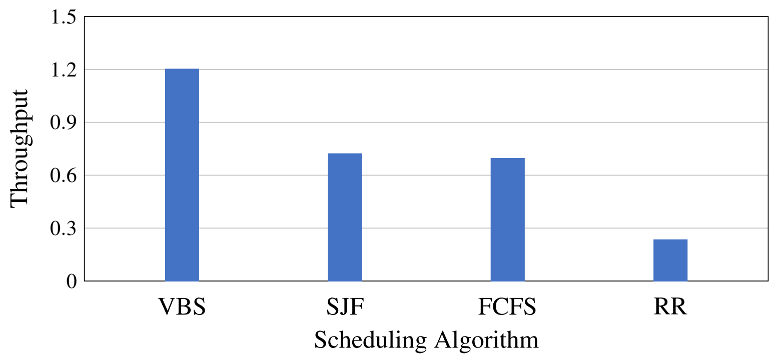

Figure 5.5: Overall throughput of the transient node.

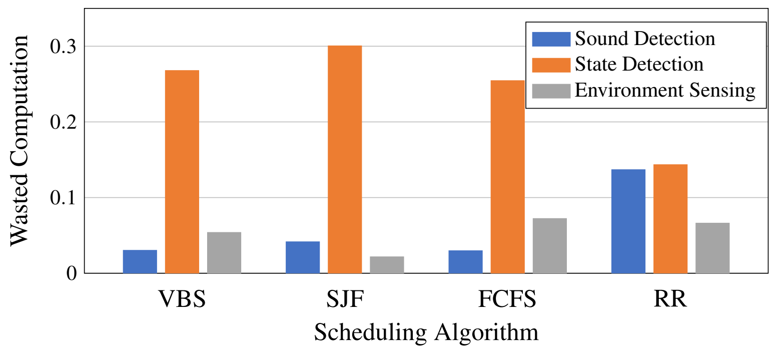

Figure 5.6: Wasted computation per application for each evaluated algorithm.

overhead increases as the complexity of the policy increased.

5.5 Over-the-Air Update

Key aspect of the system is the networking interface. The tag-to-tag stack and hardware was developed as part of different work and its evaluation is beyond the scope of the current thesis. Both receiver and transmitter are composed by a TI MSP430FR5969 MCU [5] connected to a ADG904 [54] wideband multiplexer which is used to modulate the ambient RF signals for the transmission protocol. For the evaluation the signals are provided by a BPSG4 [55] signal generator. The receiver hosts the intermittent kernel and the transmitter hosts a transmission application which continuously broadcasts the intended payload data. The three devices are aligned in a straight line with no obstructions. The transmitter was placed 5 cm from the signal generator and the receiver was placed in distances 5–30 cm, with increments of 5 cm. Transmission beyond that range was impossible. The transmitting node was connected to the standard 5 V supply while the

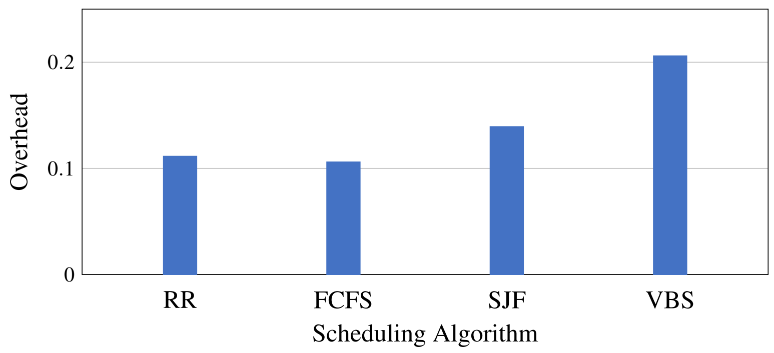

Figure 5.7: Scheduling overhead per evaluated scheduling algorithm.

Table 5.1: Over-the-Air update performance

	Stable Power	**Transient Power**
Bytes/s	14.69	9.13
Reception Time(s)	69.7	112.14
Re-transmissions	3.2	5.21

receiving node was operating intermittently.

To correctly implement a tag-to-tag update service as mentioned in Section 4.2.2 the application layer was developed. This is the part that we are going to quantify based on the constrains imposed by the tag-to-tag network design. Transmission bandwidth, throughput and range hugely vary on near conditions, obstacles and characteristics of the carrier signal.

The amount of time that a successful reception needs depends on a number of unaccounted factors that cannot be controlled. The average time that is needed for a successful 4 bytes reception is 89 ms, although the value ranges from 4 s to 50 ms. Experimentation for intermittent operation was conducted with execution times ranging from 100 ms to 1 s with average execution times of 400 ms. Smaller execution times made it impossible for the transmission to complete. The transmitter on stable power supply needed 21.51 s to transmit a complete task of 1024 bytes.

As mentioned in Chapter 4 the network operates based on the flooding principle, meaning that multiple retransmissions of the complete payload are required for a successful reception. Notably, throughout the experiments both in stable and transient reception ≈90% of the payload was received in the first two transmissions. The average amount of transmissions needed are presented in Table 5.1.

The update operation itself where the received data are installed as a new task, lasts for an average of 15.59 ms. This involves transferring the received task code from the receive buffer to the task-hosting part of memory as well as updating the status of the scheduler to reflect the changes in the composition of the applications.

Chapter 6

Summary

This thesis is surmised by a conclusion in Section 6.1 and in Section 6.2 possibilities for future work are presented.

6.1 Conclusion

In this thesis a software kernel for transiently powered systems is designed, implemented and evaluated. The kernel supports the features mentioned in Chapter 4 that enable an intermittent node to run multiple applications at the same time. The multi-tenant node can schedule the hosted applications using Round Robin, First Come First Serve, Shortest Job First, and Value-based Scheduling. Based on the evaluation section it becomes clear that no optimal scheduling policy exists, out of the ones examined, that can provide a definitive better result. Given the restrictions of intermittent computing and the wide variety of harvesting conditions a viable transient system can only be designed by taking into account the ambient harvesting conditions and the computational demands of the hosted applications.

For the use-case of the glass embedded sensor that was examined in this thesis it is noted that if the system designer requires to optimize for response times predictive value based scheduling provides the best possible results. The response times in this case are $\approx$1.45 times better than the base case. In contrast the other algorithms have larger response times ranging from small differences as in the case of FCFS to substantial as in RR. More details are provided in 6.1

Additionally if the system is not expected to perform in an environment where response time or throughput is important. The RR algorithm provides a low overhead solution that will evenly distribute the available CPU time regardless of other constraints. As it appears in Figure 5.6 the amount of wasted computation in round robin is relatively high in all three applications but remains in the same level when comparing to the other scheduling policies, where the *State Detection* application suffers the most penalty.

Table 6.1: Slowdown of response times in comparison to stable power supply execution

	Sound Detection	**State Detection**	**Environment Sensing**
VBS	1.45	1.48	1.43
FCFS	1.49	1.60	1.58
SJF	1.82	1.89	1.79
RR	10.96	5.34	10.81

Fairness might be important for systems where multiple operators want to host their applications in the same node and want to be ensured that no "greedy" application will hijack their operating time.

If the focus of the system is throughput then Figure 5.3 shows that Value-based Scheduling might fit the requirements of the designer better. Further quantifying the overall throughput of the intermittent node in Figure 5.5, becomes clear that regardless of the increased overhead, predictive scheduling performs better in terms of completions per time unit for the given system.

In terms of networking this thesis proves that transient nodes can be successfully updated remotely, providing ample design space for developers to expand on the usability of transient powered nodes throughout their lifespan. Most importantly this is feasible by a very low energy tag to tag network designed for such constrained processors. Although transmission speed remains low in comparison to modern standards based on the data provided in Table 5.1 it is considered to be satisfying for the nature of the implementation.

As conclusion this thesis proves the feasibility of a low overhead multi-tenant transiently powered node. The node can host multiple applications which can be run by different scheduling algorithms according to the requirements of the system designer. The applications are protected from the effects of intermittency and are able to share the available memory and CPU time with the other tenants. Moreover the fundamental units of the applications, tasks, can be shared by different applications providing an extra level of programming flexibility which has not been introduced for such systems yet. With the added functionality of remote updating, the kernel introduced here, is a highly versatile and modular system that can be used to deploy transient sensing nodes in remote environments. Removing the need for immediate access unattended transient computing comes one step closer to its realization.

6.2 Future Work

This thesis deals with questions of feasibility for multi-tenant, updatable transiently powered systems. Moreover it deals with the performance of common embedded system scheduling techniques for the transient powered domain. Here we summarize the list of further action items and associated research challenges.

Identifying Properties of Deployed Embedded Software: To be able to exploit multi-tenancy at intermittently-powered devices to the fullest, we need to know more about the characteristics of the *real* applications running on intermittently-powered devices. The information of the underlying software will provide input to the design of new schedulers. Points of interest include:

- Which applications are the most widespread among existing deployed nodes?
- What are the minimum operational requirements of such a system?
- Are there any bottlenecks or limitations introduced by multi-tenancy?

Identifying Potential Benefits of Task-Sharing: By identifying common software patterns in sensing nodes, the kernel can introduce a number of predefined tasks that can be shared by applications or be used to remotely configure new applications. As example common filters (FFT, averaging, etc.) and sampling tasks can be used to configure new sensing applications without the need to access and re-flash the node.

Bringing Time to Scheduling: Any scheduling algorithm would benefit from notion of time. This is however difficult in the context of intermittently-powered domain, where energy required for keeping time is unavailable at times of device "death". One idea is to measure time with remanence timekeepers, such as [56]. More specifically, utilizing timekeeping enables scheduling deadlines and periodic tasks which we are currently unable to cover sufficiently in this work as the timekeeping used by the current kernel only keep tracks of successful computation time.

Definition of Realistic Energy Availability Models: A rigid classification of common energy availability environments and harvesting techniques is required to design realistic intermittent systems. This will provide an opportunity of better framing the problem and more adequately providing possible solutions for scheduling and multi-tenancy. Utilizing and improving tools like Ekho [29] for emulating realistic harvesting scenarios and applying possible results to real-life scenarios is fundamental for this task. We note that definitions of realistic energy availability models is a more broad problem applied to any sub-domains of intermittent computing.

Adaptive Scheduling Algorithms: Value-based Scheduling results hint that benefits might come by scheduling algorithms that are able to adapt to the harvesting environment they are deployed. By adapting to energy availability with low overhead cost we might be able to further improve performance of transient nodes.

Tag to Tag Network: Low energy passive radios might be the key to interconnected transient nodes. Network enabled transient devices might hold the key to a truly energy neutral IoT. The network stack utilized by this thesis has ample room for optimization for the intermittent use-case and the results presented here suggest that further research is required towards that direction.

Bibliography

[1] J. A. Paradiso and T. Starner, “Energy scavenging for mobile and wireless electronics,” *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 18–27, Jan.-Mar. 2005.

[2] S. Sudevalayam and P. Kulkarni, “Energy harvesting sensor nodes: Survey and implications,” *IEEE Communications Surveys and Tutorials*, vol. 13, no. 3, pp. 443–461, Third Quarter 2011.

[3] G. V. Merrett, “Invited: Energy harvesting and transient computing: A paradigm shift for embedded systems?” in *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2016.

[4] F. K. Shaikh, S. Zeadally, and E. Exposito, “Enabling technologies for green internet of things,” *IEEE Syst. J.*, vol. 11, no. 2, pp. 983–994, Jun. 2017.

[5] Texas Instruments Inc., “MSP430FR59xx mixed-signal microcontrollers (Rev. F),” http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf, Mar. 2017, last accessed: Jul. 26, 2017.

[6] 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 Trans. Instrum. Meas.*, vol. 57, no. 11, pp. 2608–2615, Nov. 2008.

[7] A. Parks, I. in ’t Veen, S. Naderiparizi, and J. Tan, “WISP 5.0 firmware git,” https://github.com/wisp/wisp5, 2014, last accessed: Jul. 10, 2017.

[8] Y. Zhao, J. R. Smith, and A. Sample, “NFC-WISP: A sensing and computationally enhanced near-field RFID platform,” in *Proc. RFID.* San Diego, CA, USA: IEEE, Apr. 15–17, 2015.

[9] J. Holleman, D. Yeager, R. Prasad, J. R. Smith, and B. Otis, “NeuralWISP: An energy-harvesting wireless neural interface with 1-m range,” in *Proc. IEEE BioCAS.* Baltimore, MD, USA: IEEE, Nov. 20–22, 2008.

[10] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith, "Wispcam: A battery-free rfid camera," in *Proc. IEEE RFID*, San Diego, CA, USA, Apr. 15–17, 2015, pp. 166–173.

[11] Farsens, "Farsense pyros-0373 product description," http://www.farsens.com/en/products/pyros-0373, 2016, last accessed: Jul. 31, 2017.

[12] D. Fearing, "Equine microchips product website," http://www.destronfearing.com/our-products/equine/index.html, 2017, last accessed: Jul. 31, 2017.

[13] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," in *Proc. PLDI.* Portland, OR, US: ACM, Jun. 13–17, 2015.

[14] M. Hicks, "Clank: Architectural support for intermittent computation," in *Proc. ISCA.* Toronto, ON, Canada: ACM, Jun. 24–28, 2017.

[15] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proc. OOPSLA*, vol. 1, no. OOPSLA, pp. 96:1–96:30, Oct. 2017.

[16] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," *SIGPLAN Not.*, vol. 50, no. 6, pp. 575–585, June 2015.

[17] K. S. Yildirim, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester, "Ink: Reactive kernel for tiny batteryless sensors." ACM, 2018, pp. 41–53.

[18] A. Colin and B. Lucia, "Chain: Tasks and channels for reliable intermittent programs," *SIGPLAN Not.*, vol. 51, no. 10, pp. 514–530, Oct 2016.

[19] 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 Syst. Lett.*, vol. 7, no. 1, pp. 15–18, Mar. 2015.

[20] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 12, pp. 1968–1980, Dec. 2016.

[21] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on RFID-scale devices," in *Proc. ASPLOS.* Newport Beach, CA, USA: ACM, Mar. 5–11, 2012, pp. 159–170.

[22] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, “Wireless sensor networks for habitat monitoring,” in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications.* Acm, 2002, pp. 88–97.

[23] S. J. Thomas, R. R. Harrison, A. Leonardo, and M. S. Reynolds, “A battery-free multichannel digital Neural/EMG telemetry system for flying insects,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 6, no. 5, Oct. 2012.

[24] M. Piñuela, P. D. Mitcheson, and S. Lucyszyn, “Ambient RF energy harvesting in urban and semi-urban environments,” *IEEE Trans. Microw. Theory Techn.*, vol. 61, no. 7, pp. 2715–2726, Jul. 2013.

[25] Texas Instruments, Inc., “FRAM faqs,” http://www.ti.com/lit/ml/slat151/slat151.pdf, 2014, last accessed: Jul. 28, 2017.

[26] S. Guruacharya and E. Hossain, “Self-sustainability of energy harvesting systems: concept, analysis, and design,” *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 1, pp. 175–192, 2018.

[27] M. Gorlatova, A. Wallwater, and G. Zussman, “Networking low-power energy harvesting devices: Measurements and algorithms,” *IEEE Trans. Mobile Comput.*, vol. 12, no. 9, pp. 1853–1865, Sep. 2013.

[28] D. Gündüz, K. Stamatiou, N. Michelusi, and M. Zorzi, “Designing intelligent energy harvesting communication systems,” *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 210–216, Jan. 2014.

[29] M. Furlong, J. Hester, K. Storer, and J. Sorber, “Realistic simulation for tiny batteryless sensors,” in *Proc. International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, ser. ENSsys 2016, 2016, pp. 23–26.

[30] B. Ransford, J. Sorber, and K. Fu, “Mementos: System support for long-running computation on rfid-scale devices,” in *Proc. ACM ASPLOS*, New York, NY, USA, Mar. 5–11, 2011, pp. 159–170.

[31] S. Rodriguez, S. Ollmar, M. Waqar, and A. Rusu, “A batteryless sensor ASIC for implantable bio-impedance applications,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 3, pp. 533–544, Jun. 2016.

[32] G. V. Merrett and B. M. Al-Hashimi, “Energy-driven computing: Rethinking the design of energy harvesting systems,” in *Proceedings of the Conference on Design, Automation & Test in Europe.* European Design and Automation Association, 2017, pp. 960–965.

[33] N. Bhatti and L. Mottola, "HarvOS: Efficient code instrumentation for transiently-powered embedded devices," in *Proc. IPSN.* Pittsburgh, PA, USA: ACM/IEEE, Apr. 18–21, 2017.

[34] B. Ransford and B. Lucia, "Nonvolatile memory is a broken time machine," in *Proc. MSPC.* Edinburgh, United Kingdom: ACM, Jun. 14, 2014, pp. 5:1–5:3.

[35] 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 *Proc. International Conference on Embedded Systems Design.* Mumbai, India: IEEE, Jan. 5–9, 2014, pp. 330–335.

[36] J. Van Der Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *Proc. OSDI.* Savannah, GA, USA: ACM, Nov. 2–4, 2016, pp. 17–32.

[37] 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*, W. Weber, J. M. Rabaey, and E. Aarts, Eds. Berlin, Germany: Springer, 2005, pp. 115–148.

[38] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proc. LCN.* Tampa, FL, USA: IEEE, Nov. 16–18, 2004.

[39] L. Gu and J. A. Stankovic, "T-kernel: Providing reliable OS support to wireless sensor networks," in *Proc. SenSys.* Boulder, CO, USA: ACM, Nov. 1–3, 2006.

[40] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *ACM SIGOPS operating systems review*, vol. 34, no. 5, pp. 93–104, 2000.

[41] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms," *Mob. Netw. Appl.*, pp. 563–579, 2005.

[42] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," in *Proc. MobiSys.* Seattle, WA, USA: ACM, Jun. 6–8, 2005, pp. 163–176.

[43] A. Levy, B. Campbell, B. Ghena, D. B. Giffin, P. Pannuto, P. Dutta, and P. Levis, "Multiprogramming a 64kb computer safely and effi-

ciently," in *Proceedings of the 26th Symposium on Operating Systems Principles.* ACM, 2017, pp. 234–251.

[44] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd international conference on Embedded networked sensor systems.* ACM, 2004, pp. 81–94.

[45] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *Proc. ASPLOS.* San Jose, CA, USA: ACM, Oct. 5–9, 2002, pp. 85–95.

[46] J. Tan, P. Pawełczak, A. Parks, and J. R. Smith, "Wisent: Robust downstream communication and storage for computational RFIDs," in *Proc. INFOCOM.* San Francisco, CA, USA: IEEE, Apr. 10–15, 2016.

[47] H. Aantjes, A. Y. Majid, P. Pawełczak, J. Tan, A. Parks, and J. R. Smith, "Fast downstream to many (computational) RFIDs," in *Proc. INFOCOM.* Atlanta, GA, USA: IEEE, May 1–4, 2017.

[48] "Epc radio-frequency identity protocols generation-2 uhf rfid version 2.0.1 2015," 2018. [Online]. Available: http://www.gsl.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf

[49] S. L. Hearndon, "An analysis of bluetooth low energy in the context of intermittently powered devices," Master's thesis, 2016.

[50] K. Finkenzeller, *RFID Handbook.* West Sussex, United Kingdom: John Wiley & Sons, Ltd., 2010.

[51] Y. Chouchang, J. Gummeson, and A. Sample, "Riding the airways: Ultrawideband ambient backscatter via commercial broadcast systems," in *Proc. INFOCOM.* IEEE, 2017, pp. 1–9.

[52] P. Zhang and D. Ganesan, "Enabling bit-by-bit backscatter communication in severe energy harvesting environments," in *Proc. NSDI.* USENIX, 2014.

[53] "The python standard library ¿¿ 9. numeric and mathematical modules ¿¿," 2018. [Online]. Available: https://docs.python.org/2/library/random.html

[54] Aaronia AG, "Portable signal generators bpsg series," https://www.aaronia.com/Datasheets/Generators/Aaronia-Signal-Generators.pdf, 2015, last accessed: Jul. 26, 2018.

[55] A. Devices, "Wideband 2.5 ghz, 37 db isolation at 1 ghz,," https://www.analog.com/media/en/technical-documentation/data-sheets/adg904.pdf, 2018, last accessed: Jun. 26, 2018.

[56] J. Hester, K. Storer, and J. Sorber, “Timely execution on intermittently powered batteryless sensors,” in *Proc. 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys 2017, 2017, pp. 17:1–17:13.