The ubiquitous vision of the Internet-of-Things is greatly hampered by the "battery problem". As reliable power sources like wall power are not always available where IoT devices are deployed, many devices use batteries as their main power source. Batteries, due to their limited cycle count [5, 35], potential long recharge times [32, 47], and hazardous nature [25, 29] have become a less attractive option as a power source for applications that require low maintenance and life-long service. To eliminate these drawbacks, certain ubiquitous applications which previously relied on batteries as their power source, have adopted energy-harvesting power supplies as an alternative. Such applications include building and home automation, smart industrial monitoring, and smart wearable applications. Recent works have even pushed the boundaries of smart sensing by introducing energy-harvesting medical implants [16, 31], wearable activity tracker [28, 42, 46], micro-satellites for space observation [27], and industrial and residential monitoring [1, 9, 14]. Though energy-harvesting systems are making their way into mainstream sensing applications, a vast majority of the commercial off-the-shelf IoT sensors still rely on batteries [10, 20]. Unfortunately, converting a battery-powered application to energy-harvesting is not as straightforward as replacing the battery with a harvester. Harvestable energy is usually very limited, intermittent, and unpredictable which requires special hardware and software support to achieve useful operation [8, 11, 17, 48].

The operating principle of battery-less energy-harvesting applications can be broadly categorized into two approaches: intermittently-powered and energy-neutral. The first category of sensors harvest energy from the environment through solar, RF, thermal, and kinetic sources, store the energy momentarily in a capacitor, operate until the capacitor is depleted, and repeat this cycle continuously, while the latter store energy for future use and regulate the operational frequency of the sensor to ensure that the outgoing energy roughly matches the combined incoming and stored energy. Various designs implement these techniques to realize energy-harvesting systems, including hardware-based [12, 18, 24, 49] and software-based solutions [7, 11, 26, 36]. In both cases, however, energy-harvesting systems typically consist of a single processor along with an energy-harvesting front-end and application peripherals, where the processor is responsible for both energy management tasks (i.e. tracking the amount of energy stored, controlling the wake-up time interval, turning on peripherals at specific voltage levels, etc.), and application-specific tasks (i.e. sampling, computation, and transmitting radio packets). While this monolithic architecture can be simple and efficient for the intended application, adopting these platforms to build new applications can be quite difficult due to tightly-coupled implementations of energy-management code and application code. The intertwined application and energy management requires the developer to be responsible for understanding not only how to manage energy and correctly implement the application, but also how the two halves might interact.
This tight coupling of energy management and application logic imposes a major limiting factor for energy-harvesting system design. In this paper, we propose Altair, a modular architecture for energy-harvesting system design that decouples energy management from application execution. We claim that traditional power supply interfaces (consisting only of one or more voltage rails and possibly a power available flag) must expand to allow energy-harvesting power supplies to encapsulate the complex energy management tasks required of sophisticated energy-harvesting systems to achieve energy-neutral operation. By requiring the power supply itself to handle tasks including energy forecasting, allocation, measurement, and management, the application logic no longer has to integrate these tasks. Application platforms can focus on the IoT task (as they would with a battery-based power supply), and the new “smart” power supply can make intelligent decisions about when the application should wake up, what operating mode it should be in, and how long it should stay active, based on its careful knowledge of the energy state.

To make these decisions, the Altair design incorporates an energy supervisor that runs energy management protocols (for example, reinforcement-based learning algorithms for harvesting prediction and long-term optimizations for energy neutrality) on behalf of the application. Since the algorithms and power supply are tightly coupled, they can be highly optimized, and must only be implemented once. Many application-level platforms can leverage the same power supply. Further, the energy supervisor can handle the uncertainty in energy-harvesting system deployment, relieving each application from needing to consider the range of potential deployment conditions it might face, and instead allowing the power supply to adapt to the local conditions post deployment.

Expanding the role of the power supply also requires fundamentally re-thinking the interface between the application processor and the energy supervisor. Altair includes a much richer interface that supports a range of potential application platforms. Altair supports “harvesting-aware” applications that can instead use the power supply almost as a co-processor to provide hints about the correct operating mode to use to meet the application’s overall operational goals. By supporting a range of use cases, Altair can help many IoT devices embrace the benefits of energy-harvesting operation.

While implementing Altair architecture, we ensure minimal resource conflict on the application hardware by offloading the energy management algorithm to a power-optimized microcontroller. Using a separate core also allows decoupling in the time and power domain and flexibility to be re-used across a variety of devices. To realize Altair design and evaluate its extensibility, we create a prototype implementation of the platform with a functional power supply interface. To demonstrate a potential complex energy management algorithm, we implement a lightweight reinforcement learning (RL)-based duty cycle adaption technique that can run entirely inside the power supply. We provide a bus-based power supply interface, as well as a software library that application platforms can use to interface with the power supply.

In our experiments, we integrate six IoT sensors with the Altair power supply and compare the performance of a variety of energy supervisor control algorithms. By demonstrating the performance of several energy-management techniques on a single hardware platform, integrated with a number of existing devices, we show the generality, flexibility, and robustness of the energy supervisor architecture. Our results show that the event capture rate of sensors when optimized by the RL-based Altair energy supervisor is comparable to using a traditional reliable power source, and the capture rate is 10x higher compared to the intermittently powered ones. Sensors can achieve 1.8-7x higher event detection accuracy with opportunistic duty-cycling. We also find that our system incurs 50% fewer power failures and has 44% more availability than the statically duty-cycled sensors.

To summarize, the main contributions of the paper include:

- We propose, Altair, an energy supervisor architecture for IoT sensing applications that executes energy management decisions separately from the application. We claim that this separation is crucial for better energy optimization and independent application design of energy-harvesting battery-less devices.
- We propose a new power supply-application interface that supports building on top of unreliable power sources and implement a flexible software library to demonstrate the efficiency of the proposed system.
- We implement the proposed architecture as a standalone PCB that can be easily incorporated into new as well as already existing battery-powered devices. The platform is open source.

2 SYSTEM DESIGN CHALLENGES

Energy-harvesting devices must balance an unreliable source of energy with application-level goals. Coupling an application’s task flow to an unreliable source of energy makes energy-harvesting systems difficult to develop and debug, and can result in poor performance. Often, the application’s task i.e., sensing, computing, or transmitting, is carefully mapped to the recent energy state of the energy storage. This tight integration between an application’s task flow and energy availability significantly limits today’s battery-less systems in several ways.

Suboptimal performance. With a high degree of energy-application coupling, an application’s execution becomes highly energy-dependent. With unreliable energy, the application needs to perform complex software checkpointing techniques to ensure forward progress, which is not always guaranteed. Application programs can enter an endless inactive loop [30, 36], producing suboptimal performance. The complexity, uncertainty, and software overhead induced in intermittent computing indicate a need for alternative approaches to design energy-harvesting systems.

Runtime energy optimization. When an application’s task execution is directly mapped to its energy status, this mapping is often performed at design time and is not optimized or re-evaluated during runtime. Decisions made at design time fail to scale post deployment. Since the nature of harvestable energy is time, space, and source dependent, modeling accurate energy states for all possible scenarios a priori is non-trivial. Figure 1 shows how co-located intermittently-powered solar energy-harvesting nodes that both transmit a radio packet each time their capacitor reaches a certain voltage. Though deployed in relatively similar environments, the harvesting rate of the sensors varies quite significantly resulting in different throughput and availability, which is hard to model at design time. Non-linear device parameters are another source of
depicts the high-level overview of the Altair energy supervisor architecture. The design consists of three core components: the energy supervisor, the energy-application interface, and the main application. The energy supervisor monitors the energy states of the storage along with load energy consumption and determines the optimal duty-cycle to achieve energy-neutral operation within the limited energy budget. The supervisor works as a wrapper function that implements power supply functionality and an interface to facilitate calls between the supervisor functions and main application. The main application implements the application specific tasks of an IoT sensor such as sampling, computation, and data communication, and makes call into the energy supervisor using the interface. The energy-application interface handles requests from the main application, defines the function-specific input/output parameters, and ensures reliable data communication. Algorithm 1 outlines how the application and the supervisor can interact. The function MAIN invokes ENERGY_SUPERVISOR specifying application requirements \( (p_1, p_2, \ldots) \) to receive the rate at which a task is performed. Instead of tying an application’s task with the specific energy status of the storage as done in many battery-less applications, the main application offloads the decision to determine an optimal wake-up rate of the sensor to the energy supervisor. This way, the dependence between the energy supervisor and the application is reduced.

3.1 Enabled Properties

Altair enables several desired properties of energy-harvesting system design that traditional implementations often cannot. It introduces a general, reusable, and reliable application-power supply interface for energy-harvesting applications and achieves independent and modular design. Since the energy supervisor and the main application are separate modules of code and the application’s task flow is not directly logically dependent on the outcomes of the supervisor, development can be performed in a parallel fashion. This decoupling also simplifies adding new APIs to the energy-supervisor and new functionality in the application. A standard interface between the energy-harvesting power supply hardware and the IoT sensor enables integrating a variety of sensors with a single power supply without re-designing the harvesting circuitry or energy management logic, enabling reusability and scalability of the platform. Also, since the application does not interact with the underlying energy-harvesting power supply hardware, the IoT application developer does not need to implement power-supply specific drivers in the application code. Moreover, though we propose Altair for energy-harvesting applications, the general architecture can be adopted in battery-powered IoT and mobile applications as well as for advanced power optimization.

Figure 2: Overview of Altair energy supervisor architecture.
Algorithm 1

```c
function ENERGY_SUPERVISOR (p1, p2, ... , pn) return action_rate
function APP_ROUTINE (rate), // application task code return
function MAIN After each t_period {
    rate = ENERGY_SUPERVISOR (p1, p2, ... , pn)
    APP_ROUTINE (rate)
}
```

4 ALTAIR SYSTEM DESIGN

An IoT application interfaces with the energy supervisor of Altair to maximize its energy utilization. In this section, we discuss the core components of the architecture and how they interact. We also investigate the design choices to understand the trade-offs in the design space.

4.1 Design Space Trade-off

We note that the isolation between the energy management and application sub-blocks proposed by Altair can be implemented in both software and hardware. In software, this isolation would be possible by delegating the energy management portion in a separate module with the implementation of appropriate interface functions accessed by the main application. In the hardware version, the energy management functionality could be executed in a separate core or a processor with dedicated hardware resources. We identify some crucial factors when choosing between these various design points. While implementing Altair as a software component would provide the desired logic detanglement and independent code development, we advocate for the hardware version of Altair design to take advantage of several benefits.

4.1.1 Minimal resource conflict. Today’s IoT devices are extremely resource-constrained due to their size and power restrictions, yet, they are expected to perform a diverse range of processing-intensive applications. Such applications include critical real-time processing, multi-radio wireless communications, and even running machine learning inferences. Typically these computation-intensive tasks are handled in real-time by a low-end microcontroller causing significant burden on the shared memory and CPU bandwidth. Adding an online energy management algorithm would exacerbate these concerns. Instead, we leverage an ultra-low power microcontroller with dedicated clock, memory, and I/O bandwidth to execute the energy supervisor in parallel with the application.

4.1.2 Decoupling in the power domain. Using hardware isolation and adding additional hardware components to the system might impose an additional energy cost in an energy-harvesting application. However, we argue that the average energy overhead can actually be reduced by leveraging a lower power core than the main application. As these two cores are decoupled in the power domain and they can be turned on/off independently, one can reduce the overall energy cost. This architecture has been implemented by silicon vendors in many low power dual-core processors [43, 45]. Furthermore, the energy-management core can be further power-optimized with the recent growth of ultra-low power chip technology.

4.1.3 Reusability and generality. A hardware implementation of Altair accelerates the development phase and reduces developer effort by providing modularity and reusability across multiple applications. To promote reusability, we adopt the hardware-accelerated software energy management of Altair and implement the energy supervisor in a lower power microcontroller taking inspiration from the ARM’s big.LITTLE technology [4] that leverages a smaller lower power core to enable power optimization. In the evaluation, we test the performance with a variety of IoT sensors and demonstrate the composability and generality of the platform. This enables future embedded designers to rapidly develop their own applications while adopting energy-harvesting functionality.

4.2 The Energy Supervisor

The energy supervisor of Altair handles the tasks of energy management, prediction, and allocation, and makes decisions independently from the application logic. To accomplish this, the energy supervisor has two key components. First, the supervisor interacts with an energy-harvesting front-end to collect useful information about the harvesting conditions. This information includes the average input power, the charging rate of the storage, and instantaneous and average stored energy. The energy-harvesting front-end typically accommodates an energy-harvester (e.g. solar, RF, thermal, or piezoelectricity), a charge controller, and an energy storage (e.g. capacitor). Second, the supervisor implements the dynamic power management scheme and the interface presented to the main application. For dynamic energy management, the application can specify the parameters (i.e., duty-cycle) to be optimized and an optimization algorithm among the supported ones. The supervisor can also inform the application about which operating mode the application peripherals should be running in, or the recommended order of priorities for multiple applications.

The supervisor makes power management decisions by keeping track of system’s past experience and predicting future expected energy incomes. Learning and adapting the optimization parameters at runtime, as opposed to fixed design time or datasheet parameters, makes the energy supervisor more robust to real-world deployment conditions. The supervisor attempts to support any type of application workload. However, as the underlying hardware can only buffer a finite amount of energy, the average energy consumption of the application must be below the maximum buffered energy.
4.3 Energy-Application Interface

The energy-application interface enables the abstraction layer between the main application and the energy supervisor module. It facilitates communication between the energy supervisor and the main application by implementing a set of useful APIs. This standard interface enables updates and improvements to the energy supervisor and any optimization algorithms without requiring direct changes in the application.

4.3.1 Altair Energy API. Table 1 shows the list of available APIs provided by Altair. The energy supervisor calls get_critical_parameters, get_app_list, and get_power_modes to acquire application or device-specific information. These are fixed configuration parameters of the application that are not expected to change at runtime. get_critical_parameters returns an array of permitted duty-cycles of the running application, according to which the energy supervisor optimizes for long term energy neutrality, and which energy optimization algorithm from the supported ones to use. Currently, the platform implements three duty-cycling mechanisms (described in Section 6.2). To understand how energy is being spent, get_app_list provides the list of energy-atomic operations performed by the application. Energy-atomic operations are categorized into sampling a sensor, computing and analysing the sampled data, transmitting data, or receiving data. Each of these operations is associated with a unique operation ID. The application specifies the required operating power modes using get_power_modes. Altair saves this information into the non-volatile memory of the energy supervisor to eliminate the need to repeat the APIs calls after a power failure.

On the application side, Altair provides another four APIs, namely get_current_energy_status, get_optimal_dutycycle, get_update_period, and get_energy_model. get_optimal_dutycycle returns the calculated optimal duty-cycle which is one of the values specified by get_critical_parameters and the power modes of each operation. The application performs sensor sampling, computation, and communication at this optimal rate and enters sleep mode in between operations. The get_update_period returns at what interval the application should check for the updated duty-cycle. This depends on how variable the incoming energy profile of the device is (defaults to 15 minutes). The get_current_energy_status and get_energy_model offer finer insight into the system’s energy status. By calling these, the application receives the current stored energy on the capacitor and the current numeric input values used by the duty-cycle algorithm to calculate the duty-cycle, respectively.

4.3.2 Hardware Energy Interface. The hardware energy interface consists of the hardware abstraction layer that configures the hardware interface between the supervisor and the application. Each API call is executed by a set of hardware signals and a data communication channel. The interface consists of voltage, control, and data channel as shown in Figure 2. The data channel enables a synchronous communication channel between two processors where the application processor provides the clock signal. When the application processor makes a call into the API functions, it sends an interrupt signal to the energy processor. The energy processor uses the interrupt to configure the communication hardware and initiate data transfer. The energy APIs called in the previous section are translated into data packets. The first byte of energy API packet encapsulates header information specifying the intended API call and a read/write bit, and the next two bytes specify the message length. To invoke the energy supervisor to call an API, the main application sends a write request and an API call from the application is sent as a read request. Both processors avoid sending a new request if there is any previous unresolved or pending request. We also keep a timeout timer to avoid a communication deadlock.

Figure 3 shows a flow diagram between the energy supervisor and the application code using Altair energy API. Upon startup, the main application uses the configure_supervisor to send write requests and prompt the energy supervisor to call the next three functions for configuration. get_current_energy_status and get_energy_model is called at any time application, while, the get_optimal_dutycycle is invoked according to get_update_period.

4.4 The Main Application

The main application is a piece of software that performs the typical workload of an IoT sensor, i.e. sampling, computing, processing, and transmitting.

5 ALTAIR PLATFORM IMPLEMENTATION

We implement the Altair energy-harvesting power supply module in a custom PCB.

5.1 Hardware Components

The Altair hardware consists of two primary modules: a power system module and an external application module. The power system module implements the energy supervisor, low level energy-harvesting hardware, and the hardware interface between the energy supervisor and the main application. The main application is representative of a typical IoT sensing application that is powered through the power supply interface.

Power supply module. The power supply module of Altair hardware accommodates an energy-harvesting front-end and a companion microcontroller that implements the energy supervisor software. Figure 4 shows the power supply module and the block diagram of the core components.

An ultra-low power battery charger IC SPV1050 charges the supercapacitor from a solar or TEG harvester until it reaches 3.1 V. A nano-power boost regulator MAX17222 with > 70% efficiency at 10 µA of input current regulates the supercapacitor voltage after its voltage reaches 2 V. The platform currently uses a monocrystalline IXYS solar cell as the harvester and a 470 mF supercapacitor.
with an ESR value of 25 Ω as electrical storage. We size the capacitor empirically to ensure that it can supply the highest system peak current.

The energy supervisor uses an ultra-low power 32-bit ARM Cortex-M0+ with a 8 KB of SRAM and 64 KB of flash with different low power modes. The power supply consists of a current-sense amplifier MAX9634 to keep track of the load energy consumption. A nano-power power gating IC TPL5110 with reconfigurable time interval allows the MCU to duty cycle the application in hardware with minimal calibration. The MCU leverages a digital potentiometer to dynamically reconfigure the time interval according to the calculated duty cycle.

The interface. The interface of the power supply module provides two voltage rails of 3.3 V and 1.8 V, one duty-cycled voltage rail, capacitor voltage output. We use SPI to exchange information between the two microcontrollers and one GPIO to trigger interrupts. For debugging and evaluation, the interface exposes a UART channel that can be used to log the instantaneous capacitor voltage state and current measurement channel.

Application module. The application module of Altair platform is an externally attached sensor. We implement an air quality and pressure sensor board as a part of the platform.

5.2 Energy Supervisor Implementation

We implement an example energy supervisor to show how the architecture can be leveraged to optimize the duty-cycle of the connected application. With the dedicated hardware resources of the energy supervisor microcontroller, processing-intensive on-device energy optimization can be implemented without imposing significant resource conflict on the application microcontroller. One of the useful properties of the energy supervisor is its capability to learn to behave optimally post deployment without explicitly modeling the harvesting environment pre-deployment. To demonstrate this, we implement an on-device energy supervisor using reinforcement learning. Reinforcement learning has shown promising results as a power management technique since it can enable the sensor node to learn to adjust its duty cycle in a completely unknown environment [3, 21, 41]. The RL-based energy supervisor reacts to changes in available energy to update an application’s operation, in this implementation, the rate of sending packets to report an event. The goal of the algorithm is to maximize the application sensing rate while avoiding critical energy depletion.

At a given time, the energy-harvesting node acts as an agent in different states ($s_t \in S$) corresponding to the available stored energy, incoming energy, and energy consumed by the load. The environment in this scenario consists of the stochastic harvestable energy source and the randomness inherent in the sensor hardware. The node interacts with the environment in time-slotted episodes by selecting a sensing rate ($a_t \in A$), and receives feedback in the form of reward ($R : S \times A \rightarrow R$). Through a series of such interactions with the environment, the agent finds its optimal policy ($\pi^*$) to select future actions.

RL algorithm. We define the state space for the algorithm to capture the energy profile of the system. At a given time-step $t_k$ of an episode, the energy supervisor collects all the following state information,

$$S = \{e_{st}(t_k), e_{in}(t_k), e_{load}(t_k)\}$$  

where $e_{st}(t_k)$, $e_{in}(t_k)$, and $e_{load}(t_k)$ denotes the supercapacitor voltage at $t_k$, average input energy, and the load energy consumption during $t_k$. These parameters are indicative of the system’s overall energy dynamics for which the supervisor finds an optimal action for the sensor. We consider a 24-hour long episode with a time step of 20 minutes.

The action space consists of a set of discrete sensing rate,

$$A = [r_{min}, \ldots, r_{max}]$$  

where $r_{min}$ and $r_{max}$ are the minimum and maximum rate for the application. At each time step $t_k$, the agent selects an action $a(t_k) \in A$ according to the underlying policy. The goal of the reward function is to inspire the agent to choose the actions that maximize the sensing rate of the application and maintains minimum required energy on the energy storage. To model the reward function we adapt the reward function proposed by Aoudia, et al. [3] as follows:

$$R = (e_{st} - e_{min})/(e_{max} - e_{min}) \ast a(t_k)$$  

We assign a negative reward of -400 if capacitor voltage falls below the minimum required voltage level of 2.0 V. We choose this number so that the maximum cumulative reward over an episode
Figure 5: Spectrum of IoT sensors on a scale of hardware and software flexibility. The left-most category has maximum flexibility, whereas to the right-most has fixed hardware and software. We evaluate the Altair platform with different points on this scale to demonstrate generality.

Figure 6: Altair device deployments.

does not exceed the negative reward. Algorithm 2 lists the pseudocode showing how we implement the SARSA reinforcement learning technique [44] to calculate the optimum duty cycle of an application.

Parameter setup. Though states and actions are continuous functions, we discretize those to restrict the size of Q-matrix. The discrete action space is $A = [1,2,3,4,5]$ s, which denotes the time between two consecutive tasks. We set $\lambda = .99$, $\gamma = .8$, $\lambda_{\text{min}} = .1$, $\delta = .001$, $\alpha = .1$ after explicit testing. To enable faster convergence, we ensure that the learned Q-table is saved before a power failure happens by polling the capacitor voltage in the background.

6 EVALUATION

To evaluate the Altair design, we investigate the usability of the energy supervisor architecture and develop a set of different IoT applications. To demonstrate the versatility of the architecture, we run the applications using different energy supervisor algorithms and compare their performance. We tested the platforms across four categories of IoT hardware and evaluated how well these applications perform in terms of event generation frequency for periodic sensing and percentage of accurate detection for event-based applications. We integrated six sensors with the Altair hardware platform. We also explore the performance of the reinforcement learning based energy supervisor to understand how well the system adapts in terms of cumulative active time and reactivity—an inherent feature of the energy supervisor that shows the online adaptability of the system in post deployment situations.

6.1 Methodology

Categorizing existing IoT devices. Altair uses its standard hardware and software interface to enable different applications. To test the usability of the Altair power supply interface, we broadly categorize existing IoT devices into four groups based on the hardware and software interface exposed by the device: 1) sensors that are custom built specifically to use with Altair platform ensuring ideal interfacing, 2) sensors with open source hardware and optimized applications, 3) sensors that have available hardware design with somewhat modifiable software stacks, 4) off-the-shelf sensors with non-modifiable hardware and software. This spectrum is shown in Figure 5. Of these four groups, the first group of sensors is best suited for use with Altair. However, embedded software developers typically use the second and third categories of sensors.

We select six IoT sensors from these four categories to perform our experiments. These sensors are 1) a Pascal sensor board that

<table>
<thead>
<tr>
<th>Test platforms</th>
<th>Processor</th>
<th>Peak current (mA)</th>
<th>Default power supply</th>
<th>Available interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pascal</td>
<td>Cortex-M4 nRF52840</td>
<td>13.6</td>
<td>flexible</td>
<td>power supply, SPI</td>
</tr>
<tr>
<td>BLEES</td>
<td>Cortex-M0 nRF51822</td>
<td>15</td>
<td>Non-rechargable battery</td>
<td>power supply</td>
</tr>
<tr>
<td>Herald</td>
<td>Cortex-M0 nRF51822</td>
<td>14.8</td>
<td>Intermittently powered</td>
<td>power supply</td>
</tr>
<tr>
<td>LPCSB</td>
<td>Cortex-M0 nRF51822</td>
<td>14.6</td>
<td>USB-powered</td>
<td>power supply, I2C</td>
</tr>
<tr>
<td>Nordic Thingy:52</td>
<td>Cortex-M0 nRF52832</td>
<td>10</td>
<td>Rechargeable battery</td>
<td>power supply, I2C, SPI, MOSFET drivers, IO</td>
</tr>
<tr>
<td>SensorBug</td>
<td>BR-LE4.0-S3A</td>
<td>17</td>
<td>Non-rechargable battery</td>
<td>power supply</td>
</tr>
</tbody>
</table>

Table 2: Specifications of test applications.
monitors ambient air quality and pressure (category 1), 2) the BLEES platform [1] that senses temperature, humidity, light, pressure, and movement, (category 2), 3) the LPCSB [40], an ambient light sensor that categorizes natural light from sunlight, (category 2), 4) Herald, an intermittently-powered energy harvesting Bluetooth Low Energy (BLE) beacon [39] (category 2), 5) the Nordic Thingy:52 [33], a multi-sensor prototyping platform (category 3), and 6) the SensorBug [6], a BLE beacon for smart home monitoring with temperature, light, and acceleration sensors (category 4). While BLEES, LPCSB, Herald hardware have limited hardware interfaces, the Pascal and Thingy platform includes a relatively richer interface with ports for communication including I2C and SPI. For the devices that do not have a data channel or open software that we can reprogram, we use the duty-cycled voltage terminal of the power supply interface to turn on/off the sensor according to the calculated duty-cycle. This exhibits the benefit of using the hardware version of the energy supervisor as discussed in Section 4.1.3.

The selected devices are designed to work on different powering options including rechargeable/non-rechargeable batteries, constant power, and intermittent source of energy. Also, these sensors use different application microcontrollers and their energy consumption varies. Table 2 lists the characteristics of these hardware platforms. Altair’s strength lies in its ability to take a battery-powered sensor and convert it to a self-powered energy-harvesting device. We envision that this will pave the way to many future battery-less applications.

**Interfacing with Altair.** To interface with Altair, we simply deactivate the default power supply of the sensor and jumper the power rails and SPI channel to the Altair power supply. In the Thingy:52 board, we connect the voltage rails bypassing the battery monitoring circuitry. The application uses the energy API library at runtime to interface with the energy supervisor. The application developer implements the mapping between the API and their corresponding request id as an initial configuration for both the application and energy supervisor.

**Sensing applications.** We consider periodic and event-based sensing tasks from the above four categories to understand how well the adaptive power management algorithm captures useful events. The sensors use Bluetooth Low Energy (BLE) radio to report events. An always-on BLE receiver scans for advertisement packets and advertisements are sent with short intervals in between, in the range of milliseconds to a few seconds.

**Deployment.** Our deployment scenario consists of four different indoor locations in a building space that are exposed to variable light levels across different times of a day: on three walls, on a desk, a door, and a window. Figure 6 shows some of the deployed devices. A gateway device collects the BLE packets sent by the deployed sensors and logs them for post-processing. We train the energy supervisor reinforcement learning agent before beginning the data collection unless specified otherwise.

### 6.2 Energy Supervisor Performance

**Event frequency.** In this section, we compare the performance of the six test sensors in terms of the captured event frequency with respect to their default power source and different variants of
Figure 8: The percentage data yield of each sensor normalized to their default power supply. The ALTAR energy supervisor produces better data yield than the ALTAR-max variant that always selects the high sampling rate.

Figure 9: Percentage active time comparison across different energy supervisors. Active time denotes the percentage of time within an interval the sensor was continuously transmitting data. ALTAR outperforms the other variants.

Figure 10: Packet distribution with ALTAR. Sensors with ALTAR opportunistically choose between five allowable rates, prioritizing the higher rate.

Figure 11: Packet distribution with the default power source.

outages due to its relatively high peak current (Table 2). ALTAR achieves overall higher captured event frequency than ALTAR-Min, the intermittent power supply, but worse than ALTAR-Max. For the intermittently-powered herald beacons, the time between consecutive samples is directly affected by the availability of harvestable energy and charge time of the storage capacitors resulting in larger delays. ALTAR system however masks the irregularity of energy by storing it in a sufficiently sized capacitor and ensures samples are collected evenly at the desired rate. According to Figure 7, Herald achieves 10× higher captured event frequency with ALTAR than with its intermittent power supply.

ALTAR produces better percentage data yield and active time than both baselines as shown in Figure 8 and Figure 9, as ALTAR optimizes for better sensing rate and fewer power failures. The percentage data yield signifies the amount of produced data normalized with respect to constant power sources and the percentage active time denotes the time in a fixed time interval for how long the sensor was active.

Figure 10 compares the distribution of inter-sample times of the sent packets. ALTAR distributes the sample rate among the allowable rates reactively based on the decision of the energy supervisor. The RL agent chooses more and more actions that sample packets at a high rate when there is an energy surplus and relaxes the rate when the system is likely to see a power outage. The distribution shows that for all the sensors more than 35% of the total samples have a rate of one sample per second. Figure 11 shows the distribution for the default power supply. In the case
of intermittently-powered systems, the samples are more sporadic and the sensor is spending majority of the time in charging the energy storage. Such systems are likely to miss events than Altair that prioritizes higher sample rates when possible.

**Event detection accuracy.** To investigate how well applications can detect external events with Altair, we classify event-based applications into two categories: time critical and non-time critical. For the time critical scenario, detecting an event should be instantaneous (i.e., less than a few seconds) since some external agent might need to react to that event, for example, door sensors and motion-based light switch. For the non-time critical applications, detecting an event in a reasonable time interval is sufficient, for example, temperature sensors for HVAC systems. We deployed one BLEES sensor to detect door events, one to detect motion in two different locations and one Thingy:52 to detect temperature events.

We connected one BLEES board with the Altair power supply and deployed it on a door to detect each time the door has been opened or closed, and two of them in a hallway and on a desk to detect movements for one week. With the default power supply, when the sensor gets an interrupt due to an event, BLEES wakes up to report the event. When connected with Altair, the power supply processor fully controls the turn on/off the BLEES application processor. For the Thingy:52 board, the sensor is configured to go to the sleep mode and wake up when an event happens and report that event only if the capacitor has sufficient voltage. We chose to detect motions in two different locations to emulate two real-life scenarios: spaces that are usually lit most of the time of a day like a hallway, and spaces that have sporadic light exposure and sensing and harvesting is likely to happen simultaneously such as at a desk. To ensure we have enough data for statistical reasoning, we expended the data collection process at the end by manually generating events as capturing organic events takes significant time. We used a constantly powered version of the sensors to collect the ground truth for events. We compared the performance of Altair with two variants: Altair-Min that always chooses minimum duty-cycle and Altair-Max that selects maximum duty-cycle.

**Figure 12** shows the percentage of correctly detected events and compares the result across three power management algorithms in three of the deployment scenarios. We find that sensors with Altair achieves 70% and 80% detection accuracy in the hallway and on the door respectively, higher than the other two variants. This happens since Altair spreads out the system active time by optimally choosing the duty-cycle and is likely to capture events correctly, whereas, Altair-Max sees frequent power failure events and Altair-Min misses events for spending much time in time between wake-ups. However, in the work-desk space Altair-min detects more events than Altair as it aggressively selects higher sampling rate. This signifies that careful decisions should be made for applications where the event of interest can happen before the device can harvest enough energy. In such scenarios, predicting such events beforehand can improve detection accuracy. We plan to investigate such cases for future study.

As a candidate of non-time critical event detection, we deployed one Thingy:52 to monitor the temperature of a home in two different locations: on a window and on an indoor wall. We analyze how many times the sensor can correctly report when the temperature falls below 76°F or exceeds 79°F (selected according to the comfort level of the occupants). Figure 13(a) shows that with Altair the device reports 79% and 83% of the events accurately. To determine the latency between the event has occurred and successfully reported, we show the CDF of detection latency in Figure 13(b). We find that the 95th percentile latency remains within 12 s.

### 6.3 RL Supervisor Robustness

**System active time.** Altair uses a 470 mF supercapacitor as an energy-reservoir of the system. The larger the size of the capacitor, the more time it takes to recharge after a power failure. In this section, we aim to analyze the active time of a sensor connected to Altair. We define the duration of the time a sensor samples continuously before exhausting its energy buffer as the active time.

To evaluate how much time the system spends in recharging the capacitor in a dynamic energy environment, we moved the
6.4 Energy Supervisor Responsiveness

As the energy supervisor processor receives the energy API request from the main processor through a hardware GPIO interrupt, we investigate the number of clock cycles needed to serve the interrupt. We characterize the delay to wake up the energy supervisor from a low power sleep and the delay to respond to an interrupt while performing its routine task. We show the histogram of delays of 100 interrupts in Figure 16(a) and Figure 16(b), respectively. Though the delay in terms of clock cycle varies, the distribution shows the delay can be bounded within a few clock cycles.

6.5 Energy Overhead

Using the Altair platform does come with an energy overhead. However, while implementing the platform, we chose components with low power options. Table 3 lists the active and sleep current of the used components. The average active power draw of the board is 7.8 mA and the quiescent power draw is 94.5 µA. We notice that the significant energy overhead comes from the ADC polling to observe the system energy as ADC reading over one second costs 24.3 µJ. This overhead can be reduced by polling the ADC less frequently.

### Table 3: Power draw overhead of Altair.

<table>
<thead>
<tr>
<th>Component</th>
<th>Active current</th>
<th>Sleep current</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU STM32L010R8</td>
<td>585 µA@16Mhz</td>
<td>4.7 µA</td>
</tr>
<tr>
<td>Charger SPV1050</td>
<td>2.6 µA</td>
<td>1 nA</td>
</tr>
<tr>
<td>Current Sensor Max9634</td>
<td>1 µA</td>
<td>1 nA</td>
</tr>
<tr>
<td>Power Gating TPL5110</td>
<td>35 nA</td>
<td>N/A</td>
</tr>
</tbody>
</table>

7 RELATED WORK

In this section, we review state-of-art designs and architectures in energy harvesting systems and reactive power management schemes using reinforcement learning.

7.1 Energy-Harvesting Device Architectures

Existing works for energy harvesting systems can be broadly categorized into two directions: intermittent systems which perform operations whenever there is enough energy and non-intermittent systems which usually store the harvested energy in larger capacitors for future use.

Intermittent systems are often equipped with small energy buffers and perform simple tasks whenever the stored energy reaches a certain threshold. For example, the Gecko [48] and Monjolo [13] principle performs sense and send type workload whenever the capacitor voltage reaches certain threshold. Monjolo exploits the insight that the rate of energy-harvesting is indicative to the sensed quantity. However such insight fails to scale outside the intended applications since the source of sensing and harvesting are often not co-related. One proposed architecture and toolkit for energy harvesting systems uses a similar principle [9], which masks the inevitable intermittency with a variety of trigger abstractions that...
activate the device for certain conditions. With Altair, we advocate for the power supply to be standard across different types of sensing applications.

UFoP [18] introduces the concept of federated energy that charges a dedicated capacitor for each peripheral which is responsible for specific individual tasks. By discarding the idea of a central storage for the whole system, UFoP provides flexibility for each peripheral and promotes modular application development. Flicker [19] further improves modularity and flexibility required for rapid prototyping of battery-less applications by allowing the peripherals to reconfigure their activation thresholds at runtime. Federating energy across multiple energy stores decreases the dependency between the energy availability on a single energy storage and each peripheral’s task execution. While being motivated by similar goals of modularity, flexibility, and generality, Altair decouples the energy logic altogether from the application, dedicating a sophisticated hardware module and introducing a novel software interface in the battery-less system design. Altair moves away from the intermittent principle of operation to achieve higher uptime and to support long running applications. Capybara [24] introduces a hardware-software approach to match the energy requirement by a task by dynamically resizing for its banked capacitor, which reduces cold start and capacitor recharge time. It provides the flexibility to choose from different energy modes and a combination of capacitors to activate according to the requirements of the application. While Capybara adopts a task-based model to map each task to energy modes and energy modes to the dynamic size of the capacitor, we adopt an interface-based architecture to reduce the direct coupling between the underlying power supply hardware and the application. The Signpost platform [2] is a generalized energy-harvesting platform for city-scale sensing using a shared backplane to interconnect and isolate each module, allowing energy to be used for a particular module. In Signpost, applications virtualize the amount of stored energy and employ a different duty-cycling strategy without affecting each other’s execution. In Altair, we make the power supply itself programmable which continues to learn and adapt post-deployment to learn new strategies to achieve optimal duty-cycling strategy.

7.2 Dynamic Energy Management using RL

Reinforcement learning (RL) has been adopted for dynamic energy management in energy harvesting nodes. Hsu et al. [22] introduce a dynamic power manager for energy harvesting networks using Q-learning algorithm. Another work provides dynamic throughput provisioning according to the battery’s energy level [23]. The RL agent attempts to avoid specific states of the energy storage, which include overcharging, deep-discharge, and depletion. Different from other approaches, Rioual et al. [37] investigate the performance of different reward functions for energy optimization in energy-harvesting IoT nodes. Fraternali et al. [15] introduce a day-by-day learning algorithm using reinforcement learning to maximize the quality of service of the sensing. Another SARSA algorithm proposed by Ortiz et al. [34] attempts to learn a power allocation policy in two-hop communications and maximize the throughput of a communication system. SARSA (\(\lambda\)) was also introduced to develop an adaptive power management algorithm for solar-energy-harvesting nodes [41] using large weather datasets. In this paper, we implemented a SARSA reinforcement learning on a resource-constrained embedded device to maximize the event generation and event detection rates of IoT sensing applications.

8 DISCUSSION

Partial decoupling. Though Altair reduces the logical dependency between energy management and application tasks, both subsystems are required to have a knowledge of the expected information from each other. Since IoT sensors are typically small systems with a handful of running applications, we expect the Altair architecture is sufficient. However, for large scale embedded systems full decoupling may be needed.

Vast heterogeneity of IoT applications. Though we believe that Altair is a stepping stone in the direction of a “general-purpose” energy-harvesting power system suited for IoT sensing applications, the spectrum of sensing is broad in terms of energy cost and time-sensitiveness. Applications that are susceptible to occasional power failures might require back-up source of energy such as rechargeable batteries [24]. In such a case, the RL manager might reduce the negative reward, if the backup energy source is available.

Energy storage size. Though an over-provisioned energy reservoir can mask unstable available energy and eliminate the need for complex software support, bigger capacitors suffer from higher leakage, prolonged cold-start phase, and longer recharge times.

Limited harvester support. Current Altair platform only has support for harvesting energy using solar and TEG harvesters.

Enabling new techniques. We believe that faster testing and development plays an important factor when designing novel energy-harvesting applications and Altair attempts to lower the barrier to entry. We recognize that there is a lack of prototyping platform for energy-harvesting application and this work will attract researchers to build and test new software and hardware techniques for better power management.

9 CONCLUSION

Managing energy is critical for energy-harvesting systems, and this burden has been foisted on the IoT application software with only limited support from the energy-related hardware. We argue that ad-hoc and implementation-specific interfaces between applications and power supplies constrain the development of energy-harvesting devices, and that a new MCU-power supply interface is critical for restoring proper layering to these systems. In this paper, we introduce such a system that isolates the energy management decisions from a sensor’s workload, and provides a simple interface for adding new applications to the system. By strictly separating energy-management from device operation, we believe we can lower the bar for developing energy-harvesting systems, helping to realize a fully batteryless IoT.

10 ACKNOWLEDGEMENTS

We thank the anonymous reviewers and our shepherd for their valuable insights and feedback on improving this paper. This work is supported in part by the National Science Foundation (NSF) under grants CNS-1823325 and CNS-2144940, and the Strategic Investment Fund at the University of Virginia.
REFERENCES


