Scheduling Computational and Energy Harvesting Tasks in Deadline-Aware Intermittent Systems

Bashima Islam and Shahriar Nirjon
Department of Computer Science
UNC Chapel Hill
{bashima, nirjon}@cs.unc.edu

Abstract—The sporadic nature of harvestable energy and the mutually exclusive computing and charging cycles of intermittently powered batteryless systems pose a unique and challenging real-time scheduling problem. Existing literature focus either on the time or the energy constraints but not both at the same time. In this paper, we propose two scheduling algorithms, named Celebi-Offline and Celebi-Online, for intermittent systems that schedule both computational and energy harvesting tasks by harvesting the required minimum amount of energy while maximizing the schedulability of computational jobs. To evaluate Celebi, we conduct simulation as well as trace-based and real-life experiments. Our results show that the proposed Celebi-Offline algorithm has 92% similar performance as an optimal scheduler, and Celebi-Online scheduler schedules 8% – 22% more jobs than the earliest deadline first (EDF), rate monotonic (RM), and as late as possible (ALAP) scheduling algorithms. We deployed solar-powered batteryless systems where four intermittent applications are executed in the TI-MSP430FR5994 microcontroller and demonstrate that the system with Celebi-Online misses 63% less deadline than a non-realtime system and 8% less deadline than the system with a baseline (as late as possible) scheduler.

Index Terms—Intermittent System, Energy Harvesting, Batteryless, Scheduling, DNN

I. INTRODUCTION

Many IoT devices are powered by limited-capacity batteries – which makes them portable, mobile, small, and lightweight. However, the toxic and corrosive materials contained by limited-capacity batteries are an enormous threat to our environment [1]. The U.S. alone produces more than three billion used batteries every year, which are eventually dumped in the environment [2]. Moreover, the short-lifetime of batteries requires periodic maintenance (e.g., replacing and recharging), which is inconvenient and costly, especially at remote or massive-scale deployments. To overcome these shortcomings of the battery-powered devices, batteryless IoT devices have been proposed [3], [4]. These devices are powered by harvesting energy from ambient and renewable power sources such as solar, kinetic, thermal, and radio-frequency (RF), and typically consist of microcontrollers (MCUs), energy-harvesting and management circuitry, capacitors to store energy, sensors and actuators to interact with the environment. Applications of batteryless systems include long-term sensing scenarios, such as wild-life monitoring [5], [6], environment monitoring [7], [8], smart agriculture [9], [10], infrastructure monitoring [11], wearables [12]–[18], and implants [19].

Many IoT applications require timely feedback. For instance, in event monitoring systems, such as – wildlife monitoring [5], [6], building monitoring [11], [20], and car detection [21], events need to be detected and reported on time in order to ensure prompt response, such as – notifying forest rangers of the presence of the poachers, controlling airflow in an HVAC system, evacuation of cities, and avoiding collisions. Though batteryless systems are desirable in such scenarios for their prolonged lifetime, the complexity of on-device computation and unstable power supply from the ambient sources complicates the timely execution of computing tasks on such systems. Complex multitasking workloads, e.g., audio and image processing, multi-tenancy, and ensemble learning, proposed by recent works on intermittent systems [22]–[25] further complicate the timely execution by increasing the CPU utilization.

Existing works on timeliness in intermittent systems can be broadly categorized into three types. The first category focuses on time-keeping [26]–[28], where the goal is to maintain a reliable system clock across power failures. The second category considers the temporal aspect of data across power failure by discarding stale data [29]. The third category includes runtime systems that aim at increasing the number of completed jobs [23], [30], [31] without explicitly considering their deadlines. InK [22] proposes a kernel for intermittent systems, which always executes the next task in the control flow of the highest-priority task thread. However, this work does not consider when to harvest energy or schedule tasks from predefined priority task threads to minimize deadline misses. Several previous works have proposed real-time schedulers to schedule sensing and transmission tasks in batteryless sensor nodes [32], [33], but they only consider sense-and-send operations where only the consumed energy is considered as opposed to considering both the consumed energy and the execution time.

In this paper, we propose Celebi, a pair of scheduling algorithms for scheduling real-time tasks on intermittent systems. Celebi consists of an offline scheduling algorithm, Celebi-Offline, and its online counterpart, Celebi-Online. This is the first paper that schedules not only the computing cycles but also the energy harvesting cycles of an intermittent system by considering the dynamic properties of the environment and the harvestable energy. Through this paper, we make three significant contributions.
• First, we formulate the scheduling problem for intermittent systems considering both the energy and timing demands of real-time tasks as well as the variability of the available energy for harvesting. We deduce necessary conditions for a taskset to be schedulable on an intermittent system.

• Second, we propose an offline scheduling algorithm, namely Celebi-Offline, that schedules both harvesting and computing jobs to maximize the number of jobs that meet the deadline. To achieve this goal, after an initial round of scheduling, we iteratively remove energy harvesting cycles that harvest extra energy and accommodate computing jobs so that they can meet their deadlines.

• Third, we present an online version of Celebi-Offline scheduling algorithm called the Celebi-Online, where the harvestable energy pattern is not known a priori. It is a threshold-based algorithm that avoids situations where no energy is available to harvest, and the harvested energy is not sufficient to run the system. It opportunistically execute computing tasks earlier than it is scheduled, when the harvestable energy is below the threshold.

Celebi complements prior works on intermittent systems such as time-keeping [26], [27] and execution of intermittent tasks [22], [25], [34]–[41]. In contrast to all previous works, our contribution is at the algorithmic level, while its implementation relies upon existing open-source frameworks and APIs [25], [39] that handle the lower-level aspects of an intermittent system.

We implement Celebi in a TI MSP430 microcontroller which is powered by harvesting solar energy. We implement four different complex sensing and computational applications along with system tasks, e.g., maintaining clocks, monitoring energy. These applications include – temperature anomaly detector, DNN based acoustic event classifier, RSA encryption, and bit counter.

We compare our scheduling algorithms with an optimal scheduler and three baseline schedulers (earliest deadline first, rate monotonic and as late as possible) in simulation, trace-based, and real-life experiments. Celebi-Offline scheduler, on average, shows 92% similar performance as the optimal scheduler in controlled experiments. Celebi-Online scheduler schedules 8% – 22% more jobs than baseline schedulers in controlled evaluation. Finally, in real-life evaluation, Celebi-Online performs 63% better than a non-real-time system and 8% better than the baseline scheduler.

In this paper, we propose a framework and scheduling algorithms, named Celebi, that enables deadline-aware execution of processes on intermittently powered systems. A complete implementation of the end-to-end system – from sensing and energy harvesting to real-time execution of wide variety of tasks is also developed and evaluated later in the paper.

Figure 1 shows the five major components of the system: (1) a job generator, (2) an energy monitor, (3) an energy harvester, (4) a microcontroller unit (MCU), and (4) a scheduler.

The system reads data from one or more sensors (e.g., microphone and accelerometer) and process it using one or more preloaded processes. For example, a smart earbud may run two processes – speaker recognition and hotword detection – both using the same microphone. We define the processing pipeline of a sensor stream as a task and an end-to-end processing of a sensor data sample as a job. The Job Generator takes sensor data and the processes to create jobs and adds them to the job list. From the energy monitor, the scheduler gathers the amount of harvested energy in the energy storage (e.g., supercapacitor or capacitor array [42]) and the predicted harvestable energy from the energy source and schedule jobs from the job list for execution. The microcontroller (MCU) draws power from the energy storage and executes the scheduled jobs. The scheduler also schedules when the energy harvester should harvest energy from the source and stores it in the energy storage. In this paper, we focus on the scheduler.

III. ENERGY AND TASK MODELING

A. Energy Modeling

Harvestable Energy. We define the amount of energy that is available to be harvested from energy sources as the harvestable energy. To simplify scheduling and analysis, we quantize the harvestable energy into discrete levels. We divide the total energy for each time slot by a constant (unit energy) and express harvestable energy at each time slot as an integer. The length of the time slots is a constant and depends on the shortest execution time and the lowest energy consumption of a task. For example, in Section X, we use a time quantum of 360ms and an energy quantum of 0.3mJ which corresponds to the minimum time and minimum energy required for reading data from the temperature sensor and partially checking for the anomaly.

Harvested Energy. The energy that is actually harvested by the system and stored in its energy storage (e.g., super-capacitor) is defined as the harvested energy. The MCU consumes this energy to execute jobs. Note that the harvested energy is not just a cumulative sum of the harvestable energy since (1) it changes as energy is consumed by the MCU, and (2) the system may decide not to harvest energy at a time slot even though there is available harvestable energy. We denote harvested energy at time $t$ as $E_t$.

Energy Harvested System Design Choice. The simplest design for an energy harvested system is to connect the harvester’s output directly to the load without using any energy storage in
between. However, this design is not widely used as it wastes energy when the harvestable energy is not equal to the required energy to run the system. To illustrate, when harvestable energy < required energy, the harvestable energy can neither be used nor be stored due to the absence of an energy storage.

Most intermittent computing systems [5], [12], [22], [24], [25], [42]–[44] use energy-storage-based design, where the load is decoupled from the harvester by an energy buffer (e.g., a capacitor), and a hardware or software-based controller controls the charging and discharging of the storage element. In this decoupled design, when harvestable energy < required energy, the capacitor continues to store the harvestable energy until enough energy is accumulated to run the system. Hence, when harvestable energy < required energy, harvesting (charging) and executing tasks (discharging) are mutually exclusive in this design. In this paper, we target systems where charging and discharging are mutually exclusive.

B. Task Modeling

**Computing Task.** Intermittent systems are primarily used in long-term monitoring and surveillance applications [5], [7], [9]–[16], [19], [45], [46] that use cameras, microphones, and motion sensors to collect data at a fixed frame-rate, e.g., one image frame or an audio segment every few hundred ms, and then process the data periodically. Motivated by these applications, we adopt a periodic computing task model.

We consider the processing of a data stream from a sensor on the device as a periodic task [47], \( \tau_i = (T_i, D_i, c_i, e_i) \), where \( T_i \) denotes the period, \( D_i \) denotes the relative deadline, \( c_i \) denotes the computation time, and \( e_i \) is the energy consumption rate (i.e., power). The hyperperiod is the least common multiple of the periods and is denoted by \( T \). We assume an implicit deadline task model where the deadline equals to the period, i.e., \( T_i = D_i \).

An instance of a task \( \tau_i \), aka a job, is defined as \( j_{ik} = (a_{ik}, d_{ik}, c_i, e_i) \), where \( a_{ik} \) is the arrival time, \( d_{ik} \) is the absolute deadline, \( c_i \) is the computation time and \( e_i \) is the energy consumption rate. A job misses its deadline if it fails to execute for \( c_i \) units of time before the deadline \( d_{ik} \).

**Harvesting Task.** Since the rate of energy consumption is much higher than the rate of energy harvesting, during energy harvesting, the system (MCU) enters the sleep mode and blocks execution of any computing job. We define these energy harvesting aka charging cycles as harvesting tasks which are mutually exclusive to computing tasks. A harvesting job \( h_i = (a_i, d_i, c_i, e_i) \). Here, \( a_i \) is the arrival time of the job, \( c_i \) is the execution time or the length of the charging cycle, \( d_i \) denotes the deadline where \( d_i = a_i + c_i \), and \( e_i \) is the rate of harvestable energy that is harvested by the harvesting job \( h_i \).

**NOP Task.** When the harvested energy is not sufficient to execute a computing job and the harvestable energy is zero, no computing or harvesting job can take place. We consider such cases as NOP tasks.

Fig. 2: An example of the energy and task model over 5 time units. The computing task \( \tau_1 = (5, 5, 1, 3) \).

C. Example

Figure 2 shows an example of the energy and task models for an intermittent system. Figure 2(a) shows the harvestable energy at different time slots, and Figure 2(b) shows the harvested energy (i.e., energy stored in the capacitor) at each time slot. Figure 2(c) shows a computing job \( j_{11} = (0, 5, 1, 3) \), of the computing task \( \tau_1 = (5, 5, 1, 3) \), four harvesting jobs – \( h_1 = (0, 1, 1, 1) \), \( h_2 = (2, 3, 1, 1) \), \( h_3 = (3, 4, 1, 1) \), and \( h_4 = (4, 5, 2, 1) \), and a NOP job.

At \( t = 0 \), harvesting job \( h_1 \) harvests 1 unit of harvestable energy and the harvested energy becomes \( E_0 = 1 \). At \( t = 1 \), the harvestable energy is 0 and there is not sufficient harvested energy to run the MCU. Thus, a NOP task takes place and harvested energy \( E_1 \) remains the same as \( E_0 \). During \( t = 2 \) and \( t = 3 \), the harvesting jobs \( h_2 \) and \( h_3 \) takes place and the harvested energy increases to \( E_2 = 2 \) and \( E_3 = 3 \). The MCU consumes 3 units of energy at \( t = 4 \) by executing the computing job \( j_{11} \), which reduces the harvested energy to \( E_4 = 0 \). As the system cannot harvest energy and execute the MCU at the same time, it cannot harvest 2 units of harvestable energy at \( t = 4 \).

IV. FORMULATION OF SCHEDULING PROBLEM FOR INTERMITTENT SYSTEMS

In this section, we describe the scheduling problem along with the assumptions and an example schedule.

A. Assumptions

- A1: The energy consumption rate is higher than the energy generation rate. Energy harvesters that power intermittent systems harvest energy at the rate of \( \mu \text{W} \) to \( \text{mW} \) (e.g., solar cell [48], [49], RF signal [50]–[55], and piezoelectric [56], [57]) harvesters harvest 2.5mW–1mW, 0.1mW–1mW, and 0.2mW–2.1mW, respectively. The active-mode power consumption of an MCU on the other hand is \( \approx 6\text{mW} \) [58]. Hence, for most intermittent systems [41], [59], [60], the energy consumption rate is higher than the energy generation rate.

- A2: Harvesting and computing jobs are mutually exclusive. Due to the hardware design choices, such systems exist. For example, intermittent computing systems with a single capacitor [34], [39] have mutually exclusive harvesting and computing tasks. In storage-based models (Section III-A – Energy Harvested System Design Choice), where the energy
consumption rate is higher than the energy generation rate, the mutual exclusion between harvesting and computing tasks is a fundamental characteristic. Other systems where harvesting and computing may happen simultaneously are not the target of this paper.

- **A3:** The capacitor has a fixed charging rate. A capacitor’s charging rate is not linear but it decreases as the voltage across it increases. However, to simplify the scheduling and analysis, we consider a fixed charging rate. We also assume that the storage is sufficiently large.

- **A4:** For the offline scheduling algorithm, the harvestable energy pattern is assumed to be known a priori. Estimating the energy harvesting pattern is an unsolved problem. Many [61]–[67] have achieved up to $\approx 90\%$ accuracy in estimating the energy generation pattern. For analysis purpose, in the offline scheduling algorithm, we consider that the harvestable energy pattern is known in known. Later in this paper, we provide an online scheduling algorithm where we lift this assumption.

### B. Problem Formulation

We formulate an optimization problem that maximizes the number of computing jobs that meet the deadline given a set of computing (J) and harvesting (H) jobs.

The decision variables are defined as follows:

- $x_{jt} \in \{0, 1\}$ indicates whether job $j \in J \cup H$ execute ($x_{jt} = 1$) or not ($x_{jt} = 0$) at time $t$.
- $R_j \in \{0, 1\}$ indicates whether job $j \in J \cup H$ executed fully. $R_j = 1$ when $\sum_t x_{jt} = c_j$, and $R_j = 0$ otherwise.
- $z_j \in \{-1, +1\}$, where $z_j = -1$ when $j \in J$ and $z_j = +1$ when $j \in H$.

The optimization problem is expressed as follows:

$$\max \sum_j R_j$$ \hspace{1cm} (1)

s.t. $\sum_{t \in T} x_{jt} \leq 1$, \hspace{1cm} $\forall t \in T$ \hspace{1cm} (2)

$$\sum_{t \in T} x_{jt} \in \{0, c_j\}, \hspace{1cm} \forall j \in J \cup H$$ \hspace{1cm} (3)

$$\sum_{n=0}^{t} \sum_{j \in J \cup H} z_j e_j x_{jn} \geq 0, \hspace{1cm} \forall t \in T$$ \hspace{1cm} (4)

$$(x_{jt} = 1) \implies a_j \leq t \leq d_j, \hspace{1cm} \forall t \in T, \forall j \in J \cup H$$ \hspace{1cm} (5)

- **Objective Function.** The objective function is expressed by Equation (1), which maximizes the number of computing jobs that get completed.
- **Task Constraint.** Equation (2) ensures that only one task can execute at any time slot.
- **Execution Time Constraint.** Equation (3) ensures that a job either executes fully or not at all.
- **Energy Constraint.** Equation (4) ensures that the harvested energy is always non-negative.
- **Deadline Constraint.** Equation (5) ensures that no job is scheduled before its arrival or after its deadline.

To solve this optimization problem, we use a linear programming solver [68] which uses simplex algorithm. The worst-case computational complexity of the simplex algorithm is exponential, although it can solve most problems in cubic time [69]. Such a computational cost is not feasible for larger jobsets and larger hyperperiods.

### C. Example

In Figure 3, we show a task set having two tasks $\tau_1 = (10, 1, 6)$ and $\tau_2 = (20, 20, 3, 3)$. The hyperperiod $T = 20$ and there are three jobs: $j_{11} = (0, 10, 1, 6)$, $j_{12} = (10, 20, 1, 6)$ and $j_{21} = (0, 20, 3, 3)$. Figure 3(a) shows the schedule and harvested energy when the jobs are scheduled using EDF. Here, $j_{12}$ misses the deadline due to the scarcity of energy. In Figure 3(b), $j_{12}$ misses the deadline when scheduled by a lazy scheduling algorithm that schedules a job as late as possible before the deadline. In Figure 3(c), $j_{21}$ misses the deadline due insufficient energy. Finally, Figure 3(d) shows an optimal schedule which is obtained by solving Equations (1) – (5).

### V. Observations

In this section, we describe some key observations, which are later utilized to design the scheduling algorithms.

**Theorem 1:** If the task, execution time, energy, and deadline constraints are satisfied, for all optimal schedules, a computing job is scheduled at time $t$ when harvestable energy is zero.

**Proof of Theorem 1.** We prove this by contradiction. We assume that a computing job $j$ is schedulable at time $t_m$ or $t_n$, where harvestable energy at $t_m$ and $t_n$ are $k$ and $0$, respectively. Let us assume that scheduling $j$ at $t_m$ is optimal. There are two cases.
In the first case, \( t_m \) occurs before \( t_n \). The harvested energy at \( (t_m - 1) \) is \( E \). Thus, the harvested energy at \( t_n \), \( E_n = E - e_j \), and a computing job \( j' \) where \( e'_j = E - e_j + k \) cannot be scheduled. On the other hand, by scheduling \( j \) at \( t_n \), the harvested energy at \( t_n \) becomes \( E - e_j + k \), which is sufficient to execute \( j' \). This contradicts our assumption.

In the second case \( t_n \) occurs before \( t_m \) and the harvested energy at \( t_n - 1 \) is \( E \). Like the previous case, the harvested energy at \( t_m \) is \( E - e_j \), which is not sufficient for executing \( j' \). On the contrary, scheduling at \( t_n \) provides sufficient harvested energy \( E - e_j + k \) to execute job \( j' \) at \( t_m + 1 \), which contradicts our assumption.

**Theorem 2**: If a jobset is schedulable when harvesting and computing jobs are mutually exclusive, it is also schedulable when harvesting and computing tasks can occur concurrently.

*Proof of Theorem 2.* We prove this theorem by contradiction. Let us assume that a computing jobset \( J \) is unschedulable when computing and harvesting jobs are not mutually exclusive and schedulable when they are mutually exclusive. Let, \( j \) be the first job that misses deadline, and \( j - 1 \) be the previous job that meets the deadline. A deadline miss occurs if the processor is not available for job \( j \) between its arrival \( a_j \) and deadline \( d_j \) for \( t_j \) where \( t_j < c_j \), or the available energy during that period is \( e'_j \) where \( e'_j < e_j \). Let us consider \( \Delta t \) and \( \Delta e \) be the time and energy difference when harvesting and computing jobs are mutually exclusive. Thus, the available computation time and energy for mutually exclusive harvesting and computing jobs are \( t_j + \Delta t \) and \( e'_j + \Delta e \), respectively. When harvesting and computing jobs execute in parallel, the execution time is reduced and the harvested energy is increased. Thus, both \( \Delta t \) and \( \Delta e \) are non-positive numbers. Therefore, job \( j \) is not scheduled with mutually exclusive computing and harvesting jobs, which contradicts our assumption.

**Theorem 3**: For a computing jobset to be schedulable, it is necessary that the total energy consumed by computing jobs must be less than equal to total harvested energy by the harvesting jobs in that hyperperiod. Thus, a necessary condition for a computing jobset \( J \) to be schedulable is:

\[
\sum_{j \in J} e_j \leq \sum_{h \in H} e_h \quad \text{where } H \text{ is the harvesting jobset.}
\]

*Proof of Theorem 3.* We prove it by contrapositive [70]. Instead of proving the statement above, we prove that if \( \sum_{j \in J} e_j > \sum_{h \in H} e_h \), then \( J \) is not schedulable.

Let us assume that \( \sum_{j \in J} e_j > \sum_{h \in H} e_h \). Then, there exists a \( k \) such that \( \sum_{j \in J} e_j + k > \sum_{h \in H} e_h \). Thus, there exists a job \( j' \) that fails to compute for \( \frac{e_j}{i_j} \times k \) time unit in that hyperperiod. Therefore, \( j' \) misses the deadline and \( J \) is not schedulable.

**Theorem 4**: For a set \( \tau \) of preemptive periodic computing tasks with implicit deadline to be schedulable, the necessary condition is:

\[
\sum_{t \in \tau} k_i e_i + \sum_{t \in \tau} k_i e_i \sum_{h \in H^e} e_l \leq T
\]

Here, \( T \) is the hyperperiod, \( k_i \) is the coefficient that denotes the number of jobs of that task, and \( H^e \subseteq H \), where \( H \) is harvesting taskset.

*Proof of Theorem 4.* The total computation time of the computing jobs in the hyperperiod \( T \) is \( \sum_{t \in \tau} k_i e_i \), and the execution time of required harvesting jobs is \( (\sum_{t \in \tau} k_i e_i) / (\sum_{t \in \tau} e_i) \times c_i \). For simplicity, let us denote these by \( m \) and \( n \), respectively. Thus, the necessary condition becomes \( m + n \leq T \).

We prove this by contradiction. Let us assume that \( m + n > T \). We also assume that \( J \) is schedulable. \( m + n > T \) can be expressed as \( m + n = T + k_1 + k_2 \), where \( k_1, k_2 \in \mathbb{R} \) or, \( (m - k_1) + (n - k_2) = T \). Now, there are two cases:

Case 1: When \( k_1 \geq 0 \), there exists at least one job \( j \), for which, the available execution time is less than its computation time. Let us assume that only one job \( j \) gets execution time \( c_j - k_1 \). Hence, \( j \) is not schedulable. This contradicts our assumption.

Case 2: If \( k_2 > 0 \), a sufficient number of harvesting tasks can not be executed. Let, \( h \) be the harvesting job with execution time \( k_2 \) that fails to execute, and \( e_h \) is the energy harvested by \( h \). Thus, a job \( j \) fails to execute for \( c_j - ((e_j / e_h) \times k_2) \) time units and misses the deadline. This contradicts our assumption.

**Theorem 5**: For a set of \( n \) preemptive periodic computing tasks with implicit deadline scheduled by a static/fixed priority scheduling algorithm where harvestable energy rate \( e_i \) is fixed, and harvesting and computing tasks are mutually exclusive, the worst case response time \( R_i \) of a task \( \tau_i \) is \( c_i + \left( \sum_{k=1}^{n-1} \left[ \frac{R_i}{h_k} \right] \right) \left( c_k + \frac{e_k}{e_i} \right) \) and the utilization bound is \( \sum_{i=1}^{n} \frac{c_i + (e_i / e_h)}{T_i} \leq n(2^{n/2} - 1) \).

*Proof of Theorem 5.* We prove this by construction. Let us assume that tasks are ordered by their decreasing priority, \( P(\tau_i) > P(\tau_j) \) when \( i < j \). Here, \( P(.) \) denotes the priority of the task. The worst-case response time, \( R_i \) of a task, \( \tau_i \) depends on:

- The execution time of \( \tau_i \) which is \( c_i \).
- Execution time of higher priority tasks that can preempt \( \tau_i \) and increase its response time. In fixed priority scheduling without energy constraints, this is \( \sum_{k=1}^{n-1} \left[ \frac{R_i}{h_k} \right] \). However, for intermittent systems, the execution time is a combination of computation time and the time to harvest sufficient energy to execute the job. This can be written as \( \sum_{k=1}^{n-1} \left[ \frac{R_i}{h_k} \right] \left( c_k + \frac{e_k}{e_i} \right) \).
- The required time to harvest sufficient energy \( (e_i) \). When harvestable energy rate \( e_i \) is fixed, required time is \( c_i / e_i \). Thus, \( R_i = c_i + \left( \sum_{k=1}^{n-1} \left[ \frac{R_i}{h_k} \right] \right) \left( c_k + \frac{e_k}{e_i} \right) + \frac{c_i}{e_i} \).

The utilization bound of a rate monotonic scheduling algorithm for implicit deadline periodic task model is: \( U_n = \sum_{i=1}^{n} \frac{c_i + (e_i / e_h)}{T_i} \leq n(2^{n/2} - 1) \). At the computation time of each job includes the computation time of the harvesting jobs required to harvest sufficient energy, the utilization bound for intermittent systems, \( U_n = \sum_{i=1}^{n} \frac{c_i + (e_i / e_h)}{T_i} \leq n(2^{n/2} - 1) \).

**Lemma 1**: Given that the harvestable energy rate is variable, and harvesting and computing tasks are mutually
exclusive, a necessary condition for a preemptive periodic task, $\tau_i$, with implicit deadline to be schedulable by a fixed priority scheduling algorithm is: $c_i \geq \frac{e_i + \sum_{k=1}^{T_i} \frac{D_i}{T_i}}{D_i - c_i - \sum_{k=1}^{T_i}\frac{c_i}{T_i}}$

Proof of Lemma 1. For a task $\tau_i$ to be schedulable with fixed priority scheduling, $R_i \leq D_i$ must be true. Using the value of $R_i$ from Theorem 4, we can derive this necessary condition, where $c_i$ is the average harvestable energy rate.

VI. Celebi-Offline Scheduling Algorithms

In this section, we describe the Celebi-Offline scheduling algorithm for intermittent computing systems that exploits the observations from Section V. It is an offline scheduling algorithms where the pattern of harvestable energy is assumed given. We lift this requirement in the next section where an online version of it is described. Celebi-Offline is applicable in scenarios where energy sources are controllable [72], e.g., in offices and warehouses where the lighting and the position and transmission power of RF readers are controllable by using timers or by presetting trajectories.

![Fig. 4: Step-by-step execution of Celebi-Offline algorithm.](image)

A. Scheduling Algorithm

Using the example in Figure 4, the four steps of Celebi-Offline is described as follows –

- **Initialization.** First, we initialize the time slots and harvested energy list. The example in Figure 4 has 14 harvesting jobs and three computing jobs: $j_{11} = (0, 10, 1, 6), j_{12} = (10, 20, 1, 6)$ and $j_{21} = (0, 20, 3, 3)$. The hyperperiod $T$ is 20.

- **Step 1: Scheduling Harvesting Jobs.** In this step, first, we schedule harvesting jobs at all time-slots where the harvestable energy is present. Then, we update the harvesting energy list by calculating the cumulative sums of the harvestable energy. Figure 4(a) shows the schedule and the updated harvested energy list after this step.

- **Step 2: Scheduling NOP Jobs.** We generate and schedule the NOP jobs from the jobset by checking the empty time slots where harvested energy is smaller that the minimum energy consumption rate of the computing jobs. Scheduling NOP jobs does not update the harvested energy list as no energy is being harvested or consumed at that time slot. Figure 4(b) shows the updated schedule with NOP jobs.

- **Step 3: Scheduling Computing Jobs in Empty Time Slots.** According to Theorem 1, the remaining time slots after scheduling the harvesting and NOP jobs are optimal for scheduling the computing jobs. Using EDF scheduling algorithm and Theorem 4, we determine the computing jobset that is schedulable in the remaining time slots. Note that fixed priority scheduling algorithms such as rate monotonic scheduling algorithm, deadline monotonic scheduling algorithm can also be used instead of EDF having Lemma 1 as a necessary condition. After getting the schedulable jobset we schedule the jobs using EDF and update the harvested energy list by deducting consumed energy from the harvested energy. Figure 4(c) shows the resultant schedule after this step.

- **Step 4: Iterative Removing of Harvesting Jobs.** This is the crucial step of the Celebi-Offline algorithm. In this step, for each unscheduled job, $j$ (starting from largest deadline), we check the presence of harvesting jobs between the arrival and deadline of job $j$. If there is no harvesting job, the computing job cannot be scheduled, and this job is added to an unschedulable list. Otherwise, for each harvesting job (starting from latest arrival time), we check if the replacement results in energy scarcity in any of the scheduled jobs. If the job becomes schedulable by replacing the harvesting jobs without resulting in any energy scarcity for already scheduled jobs, we replace the chosen harvesting jobs with the computing job and add it to the scheduled job list. In Figure 4(d), the harvesting job at $t = 18$ gets replaced by $j_{12}$ and all jobs are scheduled.

The computational complexity of Celebi-Offline is $O(nD)$, where $n$ is the number of computing jobs and $D$ is the maximum relative deadline of the computing jobs.

B. Schedulability Analysis

For simplicity, we assume that there are no NOP jobs. We consider two cases: (1) harvesting tasks are periodic, and (2) harvesting tasks are aperiodic.

- **Case 1: Periodic Harvesting Tasks.** For a task to be schedulable in an intermittent system, it has to satisfy two constraints – the timing constraint and the energy constraints.

  In the first and the third steps of Celebi-Offline, we schedule the harvesting and computing tasks using EDF. When both tasksets are periodic, the combined periodic taskset is
schedulable if they satisfy the timing constraint, i.e., \( \Delta(t) \leq t; \forall t > 0 \) [73]. Here, \( \Delta(t) \) is the processor demand function that calculates the maximum execution time requirement of all jobs which have both their arrival times and their deadlines in a contiguous interval of length \( t \). In an intermittent system, all tasks include both the harvesting taskset, \( H \) and the computing taskset, \( \tau \). \( \Delta(t) \) for an intermittent system is as follows–

\[
\Delta(t) = \sum_{i \in \tau \cup H} \max \left\{ 0, 1 + \left[ \frac{t - D_i}{T_i} \right] \right\} c_i
\]

Similarly, the energy constraint is: \( \delta(t) \geq 0; \forall t > 0 \). Here, \( \delta(t) \) is the energy demand function that denotes the difference between the maximum energy requirement of all computing jobs and the maximum energy generation of all harvesting jobs which have both their arrival times and their deadlines in a contiguous interval of length \( t \). \( \delta(t) \) for an intermittent system is given by–

\[
\delta(t) = \sum_{i \in \tau \cup H} \max \left\{ 0, 1 + \left[ \frac{t - D_i}{T_i} \right] \right\} (c_i e_i z_i)
\]

Here, \( \tau \) and \( H \) are the set of computing tasks and harvesting tasks, and \( z_i = -1 \) when \( i \in \tau \) and \( z_i = 1 \) when \( i \in H \). If \( \delta(t) < 0 \), then the energy demand by the computing tasks are greater than the energy harvested by the harvesting tasks, and thus, the taskset is not schedulable.

Let us assume that we removed \( K \) harvesting tasks during step 4 of Celebi-Offline. Thus, the time constraint is: \( \Delta(t) \leq t + t_K \), where \( t_K \) is the total execution time of the removed harvesting jobs. The energy constraints is: \( \delta(t) \geq e_K \), where \( e_K \) is the harvestable energy during \( K \) harvesting jobs.

- **Case 2: Aperiodic Harvesting Tasks.** When harvesting tasks are aperiodic, each job have different harvesting tasks between its arrival and the deadline. Thus, we determine whether each job \( j_{ik} \) is schedulable on arrival. A job \( j_{ik} \) is schedulable only if

\[
d_{ik} - a_{ik} - \sum_{m \in H_1} c_m - \sum_{n \in J_1} (c_n - f_n) \geq c_{ik}
\]

and

\[
E_{a_{ik}} + \sum_{m \in H_1} (c_m \times c_m) - \sum_{n \in J_1} e_n \times (c_n - f_n) \geq (e_{ik} \times c_{ik})
\]

Here, \( H_1 \) and \( J_1 \) are the harvesting and the computing jobsets which have higher priorities than job \( j_{ik} \), and are scheduled between \( a_{ik} \) and \( d_{ik} \). \( f_n \) is the scheduled execution time of job \( j_n \) before \( a_{ik} \). Equation 8 is the timing constraints which states that the remaining time after the execution of the high priority jobs between the deadline and the arrival time is greater than or equal to the execution time of \( j_{ik} \). Similarly, Equation 9 is the energy constraints which denotes that the available energy after the execution of the high priority jobs between the deadline and the arrival time is greater than or equal to the energy demand of job \( j_{ik} \).

**VII. Celebi-Online Scheduling Algorithm**

Many IoT tasks demand for an online scheduling approach where the decision needs to be made on the go. In such algorithms, the harvestable energy is not known a priori. We propose an online, threshold-based scheduling algorithm for intermittent systems, named Celebi-Online scheduling algorithm. When the harvestable energy is below a threshold this algorithm executes computing jobs early. In this algorithm, the harvestable energy at the beginning of each time-slot is either predicted or measured using a sensor, and is assumed to remain unchanged during that time slot. In section X, we measure the harvestable energy with a sensor by measuring the voltage of the solar panel and the capacitor.

**A. Scheduling Algorithm**

The Celebi-Online scheduling algorithm has three steps. Using the example in Figure 5, we describe the steps of Celebi-Online as follows –

**Step 1: Initializing and Pre-Scheduling.** First, we schedule the computing jobs using As Late As Possible (ALAP) scheduling algorithm [74] based on the deadline and the execution time. As late as possible scheduling algorithm starts the execution of a job at the latest time as long as it meets the deadline. This presents the intermittent system with the opportunity to harvest as much energy as possible before executing the tasks. All the unscheduled jobs after applying ALAP algorithm are added to an unscheduled list for consideration at a later step. Figure 5(a) shows the schedule after this step.

**Step 2: Execution of Computing Jobs.** If a computing job is scheduled at the current time by the previous step and the harvested energy is sufficient, we execute the scheduled computing job. If not, we add this job to the unscheduled list for consideration. We then check the unscheduled list to see if jobs can be executed at current time, where the remaining time in the schedule is enough to meet its deadline. If so, then we execute that job; otherwise, we execute a harvesting job (if harvestable energy is present) or a NOP job (if harvestable energy is zero).
If no jobs are scheduled at the current time and the harvestable energy is greater than a threshold, we harvest energy. If the predicted harvestable energy is smaller than or equal to the threshold, we check if any of the scheduled jobs has the opportunity to be executed early. If so, we execute it; otherwise, we harvest energy. In Figure 5(b), we execute $j_{21}$ earlier than scheduled by ALAP.

- **Step 3: Adapting Threshold.** The threshold $\rho$ is updated after each hyperperiod if either of the two conditions are true: (1) the remaining harvested energy after the hyperperiod is greater than the summation of (a) the maximum energy consumed by a computing job that misses the deadline, and (b) the minimum harvestable energy, and (2) the total execution time of NOP jobs are greater than the summation of the maximum execution time a computing job that misses the deadline and the time required by a harvesting job with lowest energy generation rate to harvest sufficient energy for executing that computing job. Condition (1) implies that we have wasted time to harvest more energy than required. Condition (2) refers that we are not harvesting enough energy and creating energy scarcity. The updated threshold equals to the minimum harvestable energy during the previous hyperperiod because a lower threshold might result in condition (1), whereas a higher threshold might result in energy scarcity.

### B. Computational Complexity

The computational complexity of ALAP is $O(1)$ as it can be prescheduled with known periodic tasks. The most time consuming computation for Celebi-Online is to determine the schedulable job when the harvestable energy is below the threshold, given all $n$ jobs are available. Thus, the computational complexity of Celebi-Online is $O(n)$.

### C. Schedulability Analysis

For a job $j$ to be schedulable with Celebi-Online, following is a necessary condition:

$$c_j + c_{H(j)} + \sum_{k \in H^P(j)} (c_k + c_{H(k)}) \leq d_j$$

(10)

Here, $H^P(j)$ are the higher priority jobs than $j$ in the jobset $J$. $c_{H(j)}$ is the total execution time of the harvesting tasks that harvest at least $(c_j \times e_j)$ units of energy. In Celebi-Online, the condition for a job $k$ to be of higher priority than job $j$ is:

$$a_j \leq (d_k - e_k) \leq d_j$$

### VIII. Simulation-based Evaluation on Synthetic Dataset

In this section, we compare the performance of Celebi scheduling algorithms against baseline algorithms using synthetic taskset and harvestable energy pattern.

#### A. Baseline Algorithms and Performance Metric.

We evaluate Celebi by comparing them with an optimal scheduler and three online baseline scheduling algorithms – earliest deadline first (EDF) [71], rate monotonic (RM) [71] and as late as possible (ALAP) [74]. Table I shows the worst case computational complexity of these scheduling algorithms, where $n$ is the number of computing jobs.

<table>
<thead>
<tr>
<th>Optimal</th>
<th>Celebi-Offline</th>
<th>Celebi-Online</th>
<th>EDF</th>
<th>RM</th>
<th>ALAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(c^m)$</td>
<td>$O(nD)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

TABLE I: Worst case computational complexity.

We use the ratio of number of jobs scheduled by the target scheduling algorithm and the number of jobs scheduled by the optimal scheduling algorithm as the performance metric.

#### B. Synthetic Dataset

The synthetic dataset contains 1,000 randomly generated computing tasks. We provide the maximum allowed period, the minimum number of tasks, and the CPU utilization, i.e., the summation of the ratio of the execution time and the period of all computing tasks, as the inputs to the random task generator, and it generates tasks with random execution time, energy consumption rate, and periods. The periods are chosen randomly from a predefined range of 1s to 60s, following existing literature on intermittent computing systems [13], [15], [18], [24], [25], [75]–[78]. We choose the execution time randomly between 1s and the period. The period is considered as the upper bound as the execution time can not be greater than the period in an implicit deadline system. The random selection of execution time depends on the number of tasks (2 to 10) and the CPU utilization (in multiples of 10). To select the energy consumption rate, we randomly choose one of the three levels of energy consumption rates. The first two levels correspond to the two levels of power consumption of an MSP430FR5994 microcontroller in its active mode. Additional sensors consume more energy to operate; hence, we add the third level to support the activation of these sensors. For each evaluation, we generate 10 iterations of 1,000 tasksets and report the average performance over these iterations. To generate synthetic energy traces, we randomly generate four levels of harvestable energy. We use these synthetic tasksets and harvestable energy traces for evaluation in Section VIII-C, VIII-D, and VIII-E.

#### C. Effect of CPU Utilization

In Figure 6a, we show the performance of the schedulers for various CPU utilization. The CPU utilization is the summation of the ratio of the execution time and the period of all computing tasks, $\sum_{i=1}^{N} (c_i / T_i)$, for $\tau_1, \tau_2, ..., \tau_N$. To demonstrate the effect of CPU utilization on different tasksets, we vary the number of tasks to find combinations of periods and execution times that have the same CPU utilization. Though the performance of Celebi-Offline is unaffected by the variation of CPU utilization, online algorithms suffer when utilization is high. The inability to rectify greedy/ suboptimal decisions in online algorithms contributes to this by executing jobs which later fails to meet the deadline due to lack of energy. Celebi-Online schedules 70% of the jobs scheduled by the optimal scheduler for 80% CPU utilization, whereas EDF, RM, and ALAP schedule 33%, 54% and 50% jobs, respectively. With further experiment, we observe that at very
low CPU utilization (< 10%) all schedulers behave close to the optimal scheduler. EDF schedules jobs with higher time and energy demands more frequently which results in energy scarcity for the remaining jobs and decreases performance. In summary, Celebi scheduling algorithms perform better than the baseline algorithms even with high CPU utilization.

D. Effect of Taskset Size

Figure 6b shows the performance of the schedulers over different number of tasks where CPU utilization is 50%. We randomly choose different periods and calculate the required execution time within the range to generate task-sets having a fixed number of tasks and a fixed CPU utilization. The performance of the schedulers drop with increasing number of tasks. Though Celebi-Offline experiences 2% performance drop, the online schedulers incur 14% – 34% performance drop. Higher number of jobs results in more choices during selection of jobs. Among the online schedulers, Celebi-Online shows higher resistance to increasing number of tasks with a performance drop of 14% because it provides more charging time than the RM and the EDF and has lesser NOP tasks than the ALAP scheduling algorithm.

E. Effect of Different Periods

In Figure 6c, we show the behaviour of the schedulers for different task periods. We consider tasksets with three different variance levels among task periods. At low variance, a taskset has tasks with same periods. At high variance, periods on all the tasks are significantly different. At medium variance, a taskset has tasks with same periods as well as tasks with significantly different periods. At high variance, Celebi-Online, EDF, RM and ALAP incur 7.5%, 8%, 9% and 13% performance drop, respectively. In this scenario, RM and EDF achieve relatively better performance by choosing jobs with short periods over the longer periods. This results in higher number of scheduled jobs but jobs with the longer period never get scheduled. When the variance is low, Celebi-Online struggles in step 1 to choose between jobs with the same deadline, which decreases the performance. The same reason also contributes to the lower performance of ALAP, RM and EDF scheduling algorithms. Therefore, in systems where different tasks have different periods, e.g., small period for timer and large period for transmission, Celebi-Online performs relatively better than the baseline online schedulers.

Even though our algorithms are intended for periodic task models, they can schedule non-periodic (sporadic and aperiodic) tasks, if the arrival times of the jobs are known beforehand, in addition to the execution time, relative deadline, and energy consumption rate of these non-periodic tasks. Figure 7 compares the same scheduling algorithms as in Figure 6c, except for RM which is impractical for aperiodic tasks. Figure 7 shows that Celebi-Offline and Celebi-Online successfully schedules ≈39% and ≈23% more jobs, respectively, compared to EDF and ALAP.

IX. SIMULATION-BASED EVALUATION ON TRACE-BASED HARVESTABLE ENERGY

In this section, we evaluate the performance of Celebi with two types of energy sources (i.e., solar and RF) in two types of scenarios: dynamic and static. We use the synthetic taskset described in Section VIII-B along with real-world energy harvesting traces for this evaluation.

A. Energy-Trace Collection

Solar Energy Trace. We collect solar energy trace in two scenarios – static and dynamic. In the static scenario, the harvestable energy is nearly constant. We collect solar energy traces during cloud-free sunny days to represent the static scenarios (Figure 9a). In the dynamic scenario, the harvestable energy varies over time. We collect solar energy trace from the side-walk of a busy street to represent the dynamic scenarios (Figure 9b) where pedestrians and passing vehicles momentarily overshadow the sunlight. To collect the energy trace, we use a Raspberry Pi that measures the voltage across the solar panel connected to a load resistor. As the Raspberry Pi is not equipped with an ADC, we use an Arduino Uno to collect the voltage and send it to the Raspberry Pi using UART. Figure 8a shows the energy trace collection setup.

RF Energy Trace. To collect the RF energy trace, we use a 915 MHz harvester-transmitter pair [79], [80] (Figure 8b). We measure the analog voltage level corresponding to the harvested power that is provided by pin $D_{out}$ of the harvester at different transmitter-to-harvester distances using an Arduino.
RF
EDF
Solar
Solar
Celebi-Offline
Celebi-Online
RF
RM
RM
(b) NLoS (wood)
RF
Celebi-Online
EDF
Static
RF
RF
Celebi-Offline
ALAP
On sidewalk
(b) RF
EDF
Solar
Solar
Celebi-Offline
Celebi-Online
RF
RM
RM
Fig. 8: Energy trace collection setup.

(a) Beside a window
(b) On sidewalk

Fig. 9: Solar energy trace

(a) LoS
(b) NLoS (wood)
(c) NLoS (human)

Fig. 10: Analog voltage level corresponding to the harvested power at different transmitter to RF harvester distance in line of sight (LoS) and non line of sight (NLoS).

Uno and Raspberry Pi. Figure 10 shows the analog voltage level for different distances and scenarios, i.e., line-of-sight and non-line-of-sight. For the static scenario, the harvester and the transmitter are in the line-of-sight at 1m distance.

To simulate a real-life dynamic scenario, we collect location trajectory of a mobile robot from [81]. For each position of the robot, we estimate the RF energy it would have harvested if it carried a RF harvester. The estimation process maps the distance to RF energy, which we measure in our lab by varying the transmitter-to-receiver distance of the RF harvester.

B. Effect of Different Energy Sources

In the dynamic scenario of Figure 11a, both Celebi-Offline and Celebi-Online perform better than the rest due to their capability of handling the variation in the harvestable energy. Despite having less harvestable energy than the solar, the RF harvester in the dynamic scenario is better for the scheduler as the transmitter-to-receiver distance changes linearly. In the static scenario of Figure 11a, Celebi-Online performs slightly better than RM by executing jobs that have larger periods but smaller execution time or smaller energy consumption rate. Such jobs get interrupted by jobs with smaller periods in RM and thus, they miss their deadline. In ALAP, more jobs misses deadline as unlike Celebi-Online, it does not reconsider the unschedulable jobs.

In Figure 11b, we evaluate the performance of the scheduling algorithms on the trace-based harvestable energy and synthetic datasets. In this experiment, all tasks have the same execution time and energy consumption rate to understand the effect of the energy sources without the influence of the taskset. We choose the average execution time and the average energy consumption of the tasks in the synthetic dataset for these taskset generation. In the dynamic scenarios of Figure 11b, Celebi-Online and ALAP performs similarly due to the lack of opportunity to execute a scheduled job early. Both RM and EDF suffers due to executing tasks too early and choosing non-optimal harvesting tasks. In the static scenario of Figure 11b, Celebi-Offline, Celebi-Online and ALAP perform similar to the optimal scheduler as both the demand of the tasks and the harvestable energy are static, and therefore, executing any task is optimal.

X. REAL SYSTEM EVALUATION

In this section, we demonstrate the performance of Celebi in uncontrolled real-life scenarios. Unlike Section VIII and Section IX, this evaluation is performed using real tasksets executing on an MSP430 microcontroller that is deployed in the wild.

A. Hardware Implementation

To implement a real system, we use TI MSP430FR5994 [58] MCU (in Figure 12 and 13) having 256KB FRAM, 8KB SRAM, direct memory access (DMA), and an operating voltage range of 1.8V to 3.6V at 8MHz CPU clock speed. We use a solar panel with polycrystalline solar cells [82] which outputs at most 5V at 40mA. As the operating voltage of the MCU is below 3.6V, we use a step-up regulator [83] that ensures that the output voltage is always at 3.3V. As the energy storage, we use a 680mF super capacitor. To monitor the harvested and harvestable energy, we utilize the analog to digital converter (ADC) in MSP430 and a 1MΩ capacitor. We use this high capacitance to reduce energy flow in the measurement circuit which draws energy from the capacitor.
For sensing, we use an electret microphone [84] and the on-board temperature sensor of the MSP430FR5994 launchpad. We read the audio sensor at 8KHz using the ADC, perform FFT, and write the data to the FRAM using the low energy accelerator (LEA) and direct memory access (DMA) without involving the CPU. Like [22], [29], we use a real-time clock (DS3132 [85]) connected via I2C for timekeeping. We use this clock only during the power up to sync and maintain the internal clocks of the MCU. This clock is replaceable with an SRAM or capacitor-based timekeeping system during power outages [26], [27]. As the capacitor can charge by draining energy from the battery of the real-time clock, we implement rectifiers using an N-channel MOSFET and a P-channel MOSFET to isolate the clock signal (SCA) and data signal (SDA) when the real-time clock is not being used. Note that the worst-case energy consumption rate of a task can be estimated [86], [87] or measured. We use TI Energytrace++ [88] to estimate energy requirements.

The second sensing task is an acoustic event detector using a scaled-down deep neural network (DNN) that runs on an MCU. This event detector reads audio signal from the microphone and executes a 5-layer DNN having 3 convolution layers and 2 fully connected layers. We use max pool layers and rectified linear unit (RELU) activation function. Due to the high computational demand of a DNN, this is the highest energy and time consuming task in our taskset. We implement this using an open-source framework for executing DNN in intermittent systems, named SONIC [25]. SONIC is a special framework for DNN built on top of ALPACA.

The RSA encrypts a fixed, in-memory input string of an arbitrary size using a fixed encryption key. We use a 6,000 character string and 64-bit key in our experiment. The bit counter uses seven different algorithms to count the set bits in a random string and compares their results to ensure correctness [89]. We repeat each operation 10,000 times. We use the open-source intermittent execution framework ALPACA [39] to implement these tasks.

We implement Celebi-Online scheduling algorithm and a baseline online ALAP scheduling algorithm to schedule these four tasks. The execution time overhead of Celebi-Online and ALAP are 12ms and 1ms over a hyper-period of 120s, respectively. ALAP incurs less overhead as it does not update at runtime and requires only a queue lookup operation. Considering the task execution time, the overhead of Celebi-Online is 30x-810x smaller. Similar to previous work [22] where multiple tasks execute in an intermittent uniprocessor, these tasks are preemptive at their atomic boundaries. This means that a task is only preempted at the end of its atomic portions. To implement this, we do not execute an atom if the remaining time-slot is insufficient. Atoms being more than 20 times smaller than the time-slots, there is effectively no utilization loss.

We implement a taskset consisting of four tasks which are described in Table II. The temperature anomaly detector reads data from the on-board temperature sensor and calculates the local outlier factor (LOF). If the LOF >> 1, the data sample is an outlier. This task is constructed with five atomic portions implemented using an open-source task-based intermittent computing framework, ALPACA [39].

<table>
<thead>
<tr>
<th>Task Type</th>
<th>Task Name</th>
<th>Execution Time</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sense and Compute</td>
<td>Temperature Anomaly Detector</td>
<td>0.36s</td>
<td>60s</td>
</tr>
<tr>
<td></td>
<td>Acoustic Event Classifier (DNN)</td>
<td>9.72s</td>
<td>60s</td>
</tr>
<tr>
<td>Compute</td>
<td>RSA Encryptor</td>
<td>4.68s</td>
<td>40s</td>
</tr>
<tr>
<td></td>
<td>Bit Counter</td>
<td>2.16s</td>
<td>30s</td>
</tr>
</tbody>
</table>

TABLE II: Description of the taskset.

B. Software Implementation

We implement a taskset consisting of four tasks which are described in Table II. The temperature anomaly detector reads data from the on-board temperature sensor and calculates the local outlier factor (LOF). If the LOF >> 1, the data sample is an outlier. This task is constructed with five atomic portions implemented using an open-source task-based intermittent computing framework, ALPACA [39].
even more, resulting in a deadline miss ratio 28%. The adaptability of Celebi-Online to the randomness of harvestable energy contributes to this additional performance boost.

XI. RELATED WORK

A. Intermittent Computing

Intermittently-powered systems experience frequent power failure that resets the software execution and results in repeated execution of the same code and inconsistency in non-volatile memory. Previous works address the progress and memory consistency [22], [25], [35]–[41], [90]–[96], time keeping [26]–[28] and energy management [42], [97]–[99] on intermittent systems. In our work we use an atomic task based model [34], [39], [100] to maintain computational progress. None of the above-mentioned works consider scheduling tasks with timing constraints, but by implementing Celebi scheduling algorithms, they can enable deadline-aware execution of tasks on an intermittent system.

B. Scheduling in Batteryless Systems

Prior works on intermittent computing propose runtime systems to increase the likelihood of task completion by finding optimum voltage [30], adapting execution rate [101], [102], and discarding stale data [29]. However, none of these considered real-time deadline-aware execution of tasks. [23] proposes execution of multiple application in intermittent system. Though the system kernel consists of a scheduler it mostly focuses on getting tasks done rather than completing task within a deadline. As previous works [22], [29] show that processing stale data in waste, a real time scheduler is necessary. InK [22] proposes a reactive kernel that enables energy-aware dynamic execution of multiple threads. However, it does not focus on scheduling the tasks to increase number of jobs meeting deadlines. Moreover, the benefit of choosing optimal harvesting is considered in this paper. Our real-time schedulers are complementary to these works and can be integrated with these kernels to make them real-time kernel for intermittent systems. Some works in wireless sensors [31]–[33] have addressed scheduling, but none of them consider the higher computation load and only focuses on sense-send operations. Moreover, most of them consider either the energy demand or the time demand of jobs for scheduling. In this paper, we focus on both type of demands of computing jobs.

XII. DISCUSSION

In this section, we discuss the limitations of Celebi along with possible solutions to address them.

The Case of Harvesters Directly Powering the Load. The proposed scheduling algorithms are not designed for intermittent systems that connect the harvester directly to the load without using any energy storage in between. These systems behaves like a persistently-powered system as long as the harvestable energy is abundant. Hence, we recommend using existing real-time scheduling algorithms for scheduling tasks on them. To incorporate intermittence into the scheduling framework, these systems can model the power-down phases as high-priority tasks prior to applying the scheduling algorithms.

The Case of Non-Periodic Tasks. In Section VIII-E, we demonstrated that the proposed algorithms are applicable to non-periodic tasksets with known job arrival times. When the arrival times are not known a priori, these algorithms are not generally applicable to non-periodic tasks. However, Celebi-Online, can be extended to support a sporadic taskset. The scheduler, in this case, will schedule anticipated sporadic jobs based on the minimum period between consecutive sporadic jobs, and will delay computing jobs by adding more harvesting jobs or NOP jobs until the sporadic job actually arrives. It will also have to discard a scheduled sporadic job if the sporadic job eventually does not arrive before the release time of the next anticipated sporadic job.

The Case of Abundant Harvestable Energy. An energy harvesting system that has harvestable energy (supply) $\geq$ required energy (demand) behaves like a persistently-powered system because there is no intermittence in power supply. These systems are out of scope of this paper.

The Case of Highly Varying Harvestable Energy. There may exist energy harvesting systems where the relationship between the harvestable energy (supply) and required energy (demand) is unknown and may change at runtime, i.e., sometimes the supply $\geq$ demand, and sometimes supply $< demand. To schedule real-time tasks on such systems, the runtime system should isolate these two cases and apply the proposed scheduling algorithms (Celebi-Offline or Celebi-Online) only when supply $< demand, and use an existing real-time scheduling algorithm (e.g., ALAP) when supply $\geq demand. This is because, although Celebi would still be able to execute the real-time tasks correctly, we acknowledge that there is a loss of opportunity to harvest energy when energy is abundant (supply $> demand) but our algorithm schedules a computing job due to the mutual exclusion of harvesting and computing jobs. The loss, however, is limited by the size of the capacitor. For instance, the potential loss of harvestable energy due to the mutual exclusion of harvesting and computing jobs is 12.6mW–17mW for the systems presented in Section X which is the difference between the consumption rate and the maximum rate of harvestable energy.

XIII. CONCLUSION

In this paper, we study the real-time scheduling problem for intermittent systems that takes into account the time and energy demands of the tasks as well as the harvestable energy in the environment. We propose Celebi, an offline and an online scheduling algorithm, that schedule both harvesting and computing jobs to increase the number of jobs that meet the deadline. Celebi-Offline performs 92% similar to an optimal scheduler and Celebi-Online schedules 8%-22% more jobs than traditional scheduling algorithms. In real system evaluation, Celebi-Online scheduling algorithm schedules 63% more tasks than a non-real-time system and 8% more jobs than a baseline scheduling algorithm.
ACKNOWLEDGEMENT

This paper was supported, in part, by NSF grants CNS-1816213 and CNS-1704469 and NIH grant 1R01LM013329-01. We thank our shepherd for guidance and the anonymous RTAS reviewers for their comments.

REFERENCES


