Hypnos: a Hardware and Software Toolkit for Energy-Aware Sensing in Low-Cost IoT Nodes

Alberto González-Pérez, Sven Casteleyn

***Abstract*—Through the Internet of Things, autonomous sensing devices can be deployed to regularly capture environmental and other sensor measurements for a variety of usage scenarios. However, for the market segment of stand-alone, self-sustaining small IoT nodes, long term deployment remains problematic due to the energy-constrained nature of these devices, requiring frequent maintenance. This article introduces Hypnos, an open hardware and software toolkit that aims to balance energy intake and usage through adaptive sensing rate for low-cost Internet-connected IoT nodes. We describe the hardware architecture of the IoT node, an open hardware board based on the Arduino Uno form-factor packing the energy measurement circuitry, and the associated open source software library, that interfaces with the sensing node's microcontroller and provides access to the low-level energy measurements. Hypnos comes equipped with a built-in, configurable, modified sigmoid function to regulate duty cycle frequency based on energy intake and usage, yet developers may also plug in their custom duty/sleep balancing function. An experiment was set up, whereby two identical boards ran for two months: one with the Hypnos software framework and built-in energy balancing function to regulate sensing rate and the other with fixed sensing rate. The experiment showed that Hypnos is able to successfully balance energy usage and sensing frequency within configurable energy ranges. Hereby, it increases reliability by avoiding complete shutdown, while at the same time optimizing performance in terms of average amount of sensor measurements.**

***Index Terms*—Constrained Devices, Energy Efficient Devices, Energy Harvesting, Low Cost Sensors and Devices, In Situ Processing**

I. Introduction

SINCE the original vision of trackable, connected objects in the context of supply chain management in 1999 [1], the Internet of Things (IoT) has evolved to a large heterogeneous network of so-called IoT nodes, ranging from simple identifiable objects capable of communicating limited metadata, to sophisticated computerized devices, comprising application logic and capable of sensing environmental parameters, communicating with other IoT devices and (cloud) servers, and actuating [2], [3]. The potential of such automated systems to collect data from connected nodes – real-world objects in daily life – process it, derive actionable knowledge and act upon it, is enormous. It should thus not come as a surprise that the Internet of Things is rapidly becoming a reality. Gartner forecasted the amount of IoT nodes to grow to 5.81 billion by the end of 2020, a yearly increase of 21% since 2018 [4]. Indeed, a plethora of next-generation applications in various domains sprung into life [5], ranging from smart cities [6], smart homes [7], health care [8] and agriculture [9], to smart factories and generally, Industry 4.0 [10].

From an architectural point of view, the IoT can be subdivided in 4 layers (bottom-up) [11]: the perception layer, which describes the IoT nodes' sensing and actuation capacities; the transmission layer, which describes how IoT nodes can communicate and transmit information; the computation layer, which describes how hardware and software technologies can receive and process data to make decisions; and the application layer, which describes practical applications in various domains. Across these layers, the IoT space is highly heterogeneous, with a variety of standards, hardware and (software) platforms available, and major commercial players each implementing and pushing their respective solutions [11]. Nevertheless, open IoT platforms are gaining importance, promoting the use of open standards, open APIs, open source and open layers [12]. This article focuses on the latter market segment, where IoT nodes consist of small, lightweight, self-sustaining and often low-cost devices, with limited memory, computational capabilities and battery power. Their deployment ranges from (hobby) projects in the maker movement [13], where technically skilled individuals assemble, program and deploy their own IoT nodes, to citizen science projects, where non-scientists deploy and use IoT nodes for scientific purposes, such as crowd-sourced environmental monitoring [14], hydrologic monitoring [15] or water quality monitoring [16] in water management, and small-scale industrial applications, such as parcel monitoring in agriculture [17] or remote water quality monitoring [18].

Energy efficiency is hereby – and in the IoT in general – one of the main challenges to address [19]–[21]. Proposed solutions range from energy efficient hardware (e.g., microcontrollers, chips, sensors) and software (e.g., lightweight communication protocols, batch cloud communication) solutions, to energy harvesting through renewable energy sources (e.g., mainly solar energy) to prolong operation time. In this article, we contribute to the reliability and performance of such small, energy-restrained open IoT nodes, by presenting a hardware and software solution to manage and balance available energy – obtained through inherently unpredictable energy harvesting – and computations (usually, sensing). Our system, Hypnos, resides in the IoT computation layer, and is agnostic of sensing and actuation (perception layer), communication (transmission layer) or practical applications (application layer). It consists

A. González-Pérez and S. Casteleyn are with the GEOTEC research group, Institute of New Imaging Technology, Universitat Jaume I, Castellón, 12071

E-mails: {alberto.gonzalez, sven.casteleyn}@uji.es

Manuscript received Xxxx XX, XXXX; revised Xxxx XX, XXXX.

of a hardware and software component. The first is an open-hardware board, pluggable on any device with the widely used Arduino pinout, a de facto standard, which measures energy intake and the overall device energy consumption. The second component is a C++ software library, which provides programmatic access to the hardware board capabilities and manages the device's duty-cycle by controlling sleep times. The software comes equipped with a modified, parameterized sigmoid duty/sleep balancing function, which increases/decreases the duty frequency according to energy intake and consumption, but developers can deploy their own function to better suit their particular use case.

The main contributions of this work are (i) a fully modular, integrated and pluggable open-source IoT platform – Hypnos – based on standards and targeted at small, energy-constrained single-threaded IoT nodes, consisting of a hardware board to keep track of the system's energy availability and a software library for energy management, tailorable depending on the use case (ii) a ready-to-use, built-in non-linear battery-centric reactive energy management algorithm, based on a customizable modified sigmoid function, suitable for variable sensing rate applications (iii) an evaluation of Hypnos' deployment and built-in energy balancing function in real-world conditions, showing good performance in terms of average amount of sensor measurements, stable energy consumption and uptime over a longer time period according to configured parameters.

In the remainder of this article, we discuss related work (Section II), detail Hypnos' hardware and software component (Section III), describe a 2+ months experiment with two different single-board microcontrollers running in identical conditions, one equipped with and one without Hypnos (Section IV), discuss results and position Hypnos w.r.t other energy management solutions (Section V).

II. Related work

One of the main challenges for standalone IoT nodes is energy efficiency [19]–[21]. In scientific literature, various approaches aim to save energy and extend operation time in IoT nodes across all four IoT layers. In the perception layer, energy optimizations focus on reducing the sensors' energy footprint, by improving the sensor units' designs (e.g., [22]) or operation time (e.g., [23]). In the transmission layer, a plethora of network technologies and protocols was specifically created for energy efficiency in IoT solutions – see [24], [25] for an overview. In the computation layer, we distinguish hardware and software-based solutions. In the former, IoT node manufacturers apply several optimization techniques to improve energy efficiency, such as increased miniaturization, integrating functionality in the microcontroller's on-board chips (i.e., Systems on a Chip) [26], decoupling processing by using independent yet cooperating processing units [27] or task-specific processor cores [28]. Software-based solutions mainly consist of power and resource management solutions – given Hypnos resides at this level, we'll discuss these solutions in detail along the remainder of this section. Finally, in the application layer, IoT applications reside, which may implement domain- or application-specific optimizations, generally aimed at reducing network traffic or sensing rate. For example, in the medical domain, [29] reduces network traffic by delaying transmission based on medical relevance, and for environmental sensing, [30] reduces data transmission and adapts sensing rate based on data variability. Examples of application-specific energy optimizations include adaptive sensing rate for car parking applications based on occupancy [31] or energy-efficient IoT nodes for fall detection [32].

We now shift our attention to software-based solutions, more specifically energy management algorithms, in the layer where Hypnos resides – the computation layer. Existing solutions aim to reduce energy consumption by efficiently managing the IoT node's power and resource usage through sleep time regulation. Hereby, the operational time (high power consumption) is decreased in favor of sleep time (low power consumption) according to the IoT node's requirements. Power management algorithms can be classified in two categories [33]: reactive and predictive, with some algorithms utilizing techniques from both.

Reactive algorithms take into account the current state of the system to decide how much energy can be consumed during an operation cycle and/or when the next operation cycle should happen. In an early work related to event detection, Vigorito *et al.* [34] considered the trade-off between operation time and (spatial) coverage as a constrained optimization problem, where sleep time is maximized w.r.t. minimal detection delay. In [35], Le *et al.* proposed a similar idea, but instead using a proportional integral derivative to adjust the sleep time of a node to its stored energy. In [36], sensing and transmission are decoupled, and the authors compare approaches to determine optimal sensing rate and data throughput, while allowing some data loss. More recently, the work from Kulau *et al.* [30] takes the volatility of the measured sensor values into account when calculating sleep time. Trilles *et al.* [17] use a simple, two-value variable sensing rate as a disaster prevention mechanism, whereby the IoT node's battery state of charge (SoC) is monitored to increase sleep time when the battery level is low. Hypnos' built-in modified sigmoid function-based algorithm falls within the category of predictive algorithms, and aims to dynamically and continuously balance energy availability (i.e., SoC) with sampling rate.

Predictive algorithms, on the other side, use historical data and scientific models to predict future energy harvesting, which is used to decide energy consumption. Often, predictive models are complemented with reactive techniques, to compensate for inaccurate predictions. Pioneering work in this area was done by Kansal *et al.* [37], who explored a prediction model based on an exponentially weighted moving average to approximate the daily solar cycle, while recognizing the importance of momentary measurements for corrective adjustment in sleep time. In other works, authors attempted to improve on-device predictions using online available solar maps [38] and wind maps [39]. Other innovative approaches consider the use of machine learning, more concretely reinforcement learning, to compute and adjust the energy availability predictions online [40]. The newer trend in predictive algorithms is to take into account the utility of a task when considering its execution. For example, PreAct [33] takes advantage of prior knowledge on temporal utility of

sensing data to decide on future energy usage, combined with corrective actions based on measured SoC. A comparable work is EmRep [41], which additionally considers low and high energy-intake periods to decrease/increased workload or reduce/prolong the duty cycle.

Next to energy management algorithms running on the edge, as Hypnos and the previously described related work, we also highlight fog- and cloud-based solutions. In [42], Wang *et al.* follow a quality of service (QoS) approach to guarantee that a service is provided by a minimal subset of networked nodes, whereby a single node governs the nodes' wake/sleep time to achieve this goal. Taneja *et al.* [43] propose a framework to reduce nodes' uptime in non-trivial network topologies where gateways (GW) and servers can aggregate messages and predict future requests' outcomes to reduce node and GW wakeups. Vo [44] proposes a reactive algorithm which runs on an IoT gateway to decide the sleep time of nodes directly dependent on it. This approach takes into account information quality and history, along with remaining battery capacity to decide on sleep time. As Hypnos runs on the edge and is agnostic in the way it interacts with other layers, it may integrate with such solutions. We cover this in the discussion (Section V-D).

With respect to the hardware, as circuitry that keeps track of the system's SoC, Hypnos uses a widely used interrupt-based coulomb counter – Linear Technology's LTC4150 [45] – to report inbound and outbound charges. Next to coulomb counters, other types of circuits are possible. For example, SPOT [46], iCount [47] and Nemo [48] are current-based energy meter modules that allow to poll energy consumption measurements. However, such solutions don't differentiate inbound from outbound current, so two meter modules would be required to distinguish power consumption and recharge in energy harvesting nodes. In this article, we do not further investigate the accuracy or energy efficiency of different circuitry.

With respect to Hypnos as an integrated hardware and software solution, to the best of our knowledge, there is only one recent comparable solution: ECO [49], an integrated energy consumption solution for more sophisticated IoT nodes, with more powerful processing capabilities, multi-threading and thread-based energy measurement support. The authors present a modular hardware design which can be integrated in existing systems running RIOT OS, using the I^2C bus to communicate with the main MCU, along with a software module to facilitate the polling of energy usage originating from voltage and current probes. The Hypnos toolkit is aligned with this proposal, yet it targets the widespread small, low cost resource-constrained IoT nodes, without multi-threading capabilities. Hereby, it offloads the responsibility of continuously tracking the SoC to a co-processor, allowing the more power-hungry MCU to sleep more (whereas in ECO, the MCU is required for multi-threading and energy measurements). Moreover, the Hypnos software integrates the use of a power management algorithm, on top of low level energy polling.

After presenting Hypnos (Section III) and empirically demonstrating its validity in a real setting (Section IV), in the discussion (Section V) we compare Hypnos with the commensurable solutions presented here at hardware, software and algorithmic level, and comment the integration possibilities – see Section V-C and V-D respectively.

III. Hypnos Toolkit

Designing and deploying reliable small, self-sustaining open IoT nodes is a non-trivial task. In scenarios where such nodes are deployed, hardware engineers have to properly size energy storage (e.g. batteries) and energy harvesting (e.g. solar panels) components, and software engineers face the tedious task of utilizing their full potential according to the usage scenario. Exceptional conditions, such as unexpected low energy intake (e.g. environmental factors causing poorly performing energy harvesting components) or other irregularities (e.g., energy loss due to isolated hardware malfunctions), further complicate the issue.

Until now, addressing those scenarios required *ad hoc* solutions. On the hardware side, this implies modifying existing hardware with a component – if not yet present – to monitor energy: current energy levels, energy recharge and consumption rates. This, in turn, allows software developers to continuously monitor the raw energy status, analyse the results (over time), and devise an efficient energy usage policy based on them. Hereby, the responsiveness of an IoT node is in principle only limited by the clock speed of its micro processor: current energy levels can be obtained at the beginning of each iteration of its execution loop, and its behaviour can be adjusted accordingly. In consequence, this allows to increase the IoT node's performance and output in real time, by boosting activity cycles as the IoT node approaches full energy availability, while improving reliability at lower energy levels, by gradually reducing activity cycles as power depletes.

This is exactly the goal of the Hypnos toolkit: to offer an out-of-the-box hardware and software solution for fine-grained energy management in small, self-sustainable open IoT nodes, and hereby bring increased performance and reliability of such IoT nodes within reach of the assemblers and firmware developers. Hypnos is made up of two components: a hardware module (board), pluggable on any microcontroller board supporting the de facto Arduino pinout standard, and an accompanying software library, which provides API access to the energy measurements provided through the hardware, along with a configurable energy balancing function to provide real-time energy management.

A. Hypnos hardware board

The function of the Hypnos hardware component is to accurately measure the available amount of energy at the beginning of each execution cycle iteration.

Contrary to their industrial counterparts, small open IoT nodes are mainly powered by consumer microcontroller boards from specialized manufacturing companies, i.e. Arduino, Adafruit, SeeedStudio or Particle, to mention a few. Those boards are usually not equipped with a mechanism to measure the system's available energy – as a general rule, its battery charge status. For example, Arduino boards are power source

agnostic. This means that they do not include a mechanism to adequately recharge a battery connected to it (e.g. a battery charger chip), nor a way to determine its charge status (e.g. a fuel gauge chip).

If a built-in component is present, it often lacks sufficient granularity to obtain the energy charge at every internal iteration of the main execution loop. Examples of boards that include mechanisms to obtain charge information, but lack sufficient accuracy are MediaTek's LinkIt One or Adafruit's Feather HUZZAH32 - ESP32 board. For example, MediaTek's board can only report 4 different charge levels[1] (0%, 33%, 66% and 100%), and Adafruit's board provides access to three charging thresholds[2], without direct access to the battery charge level through the SDK's API.

Only exceptionally, a microcontroller board has a built-in battery charging and fuel gauge chip. Particle's Electron board provides an on-board Texas Intrument's TI's BQ24195 charging chip[3] and the Max Integrated's MAX17043[4] fuel gauge chip, joint with a way to access the system's battery charge level through its SDK[5].

The Hypnos hardware component is designed for those systems lacking fine-grained access to energy levels, i.e. both at quantitative (i.e., Hypnos provides milliampere hour (mAh) level charge differences in $\sim 1/5$ mAh steps) and temporal level (i.e., Hypnos is only restricted by the clock speed of the micro co-processor responsible for energy measurement).

1) Requirements: During its design the following requirements were taken into consideration:

- **R1:** energy availability needs to be reported with a high level of granularity, as to allow detailed energy management algorithms based on it.
- **R2:** a data communication protocol supported by the vast majority of the boards available on the market has to be used.
- **R3:** compatibility with existing hardware, to ensure a seamless integration in new and existing systems, is required.

To meet the first requirement (**R1**), from a hardware point of view, a coulomb counter, which measures the accumulated energy added or discharged from a battery, fulfills the basic need. Moreover, given the battery's capacity, the accumulated energy count (over time) allows to accurately calculate the remaining amount of milliampere hour (mAh) available in the battery. Consequently, the coulomb counter is able to continuously provide an accurate battery level, only limited by the underlying clock speed of the microcontroller board. Based on this, fine-grained energy management (software) strategies can be built.

To meet the second requirement (**R2**) the Hypnos hardware component employed I^2C, a widely spread serial protocol implemented in (nearly) every consumer board to connect peripherals, to enable the communication between the coulumb counter and the main microcontroller.

Finally, regarding the design of the hardware board, the small amount of required components gave us ample freedom in choosing an adequate form-factor. To ensure a plug & play nature in as much microcontroller boards as possible, we adapted the board's pin-out and PCB format to the one of Arduino UNO, one of the most widely adopted microcontroller boards. Moreover, many different manufacturers have created their products adhering to it (e.g Sparkfun[6], SeeedStudio[7]) or offer adapter boards to map their own pin-outs to the one of Arduino (e.g. SeeedStudio[8], Particle[9]). By doing this we successfully fulfilled **R3**.

The resulting PCB design, following the Arduino UNO shield form factor, can be seen in Fig. 1b. The PCB encloses all the components of the Hypnos hardware board, to be detailed next.

2) Hypnos hardware component design: Prior to define the PCB design, we created the Hypnos hardware schematics to depict how the different board circuit components work together. We used a Linear Technology's LTC4150 coulomb counter, which senses the current passing through an external sense resistor and raises an interruption (INT) when a certain amount of charge has passed through the probe in either direction: charging or discharging. There is a polarity pin (POL) to differentiate both.

To accommodate both fine-grained energy readings and accumulated charge counts over time, we paired the coulomb counter with a separate microcontroller. Since this additional microcontroller is powered up all the time, it should use few energy. We therefore chose the ATtiny85 microcontroller, configured to be powered at 3.3V and run at 1 MHz, which consumes between 0.1 mA (while idle) and 1 mA (while active)[10].

To pack everything together, we started from the LTC4150 board design made by M. Grusing for Sparkfun [50]. The original design has been modified as follows:

- The LTC4150's interruption (INT) pin has been soldered to the ATtiny85's analog 2 (A2) pin, which has support for hardware interrupts.
- The LTC4150's polarity (POL) pin has been soldered to the ATtiny85's analog 3 (A3) pin, in order to recognise if the detected charge – the one that raises an interruption through the INT pin – corresponds to an inbound or an outbound charge, to or from the battery, respectively.
- The INT and the clear (CLR) pins from the LTC4150 chip have been soldered together and to the ground (GND). This enables to automatically clear each interruption. The result is a faster charge count detection with no drawbacks, as the ATtiny never enters into sleep mode.

[1]http://labs.mediatek.com/api/linkit-one/LBatteryClass__level.html

[2]https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/power-management#measuring-battery-2385442-8

[3]https://docs.particle.io/datasheets/electron/electron-datasheet/#pmic-power-management-integrated-circuit-

[4]https://docs.particle.io/datasheets/electron/electron-datasheet/#pmic-power-management-integrated-circuit-

[5]https://docs.particle.io/reference/device-os/firmware/electron/#batterystate-

[6]https://www.sparkfun.com/products/15123

[7]https://www.seeedstudio.com/LinkIt-ONE-p-2017.html

[8]https://wiki.seeedstudio.com/Arduino_Breakout_for_LinkIt_Smart_7688_Duo/

[9]https://docs.particle.io/datasheets/accessories/legacy-accessories/#shield-shield

[10]http://www.farnell.com/datasheets/1698186.pdf

- ATtiny's pins 6 (PWM) and 7 (A1) have been soldered to the Voltage Common Collector (VCC) with two intermediate 4.7k Ω resistors and configured as the SDA and the SDL terminals of the I^2C protocol respectively.
- All resistor values have been adjusted for the board to work at 3.3V only – instead of 3.3v / 5v – in order to reduce the board's energy consumption to a minimum.

The resulting design can be seen in the form of an EAGLE schematic in Fig. 1a.

The IoT node's battery is connected to the connector tagged as IN on the schematic, the battery charger and the rest of the system components, i.e. the microcontroller board, the attached sensors and actuators, is attached to either of the connectors tagged as OUT. I^2C headers from the ATtiny85 are in turn connected to the I^2C bus of the IoT node's main microcontroller. Finally, the board has to be powered with 3.3V by the IoT node through its VCC pin.

Each time 0.1707 mAh are consumed or recharged, the LTC4150 chip interrupts the ATtiny85, which in turn increments or decrements (respectively) its internal in-memory charge counter (an integer). The main microcontroller can obtain the total charge count at anytime by requesting it through the I^2C bus at the 0x4 address. At the main microcontroller's end, the Hypnos library multiplies that value by 0,1707 to obtain the total amount of charge that has been used/charged and subtracts/adds it from/to the total battery capacity in order to calculate the remaining battery charge. It is also possible to divide the obtained value by the total capacity of the battery to obtain the remaining battery percentage, which the Hypnos library offers to the developer. The resolution of that value is astonishingly high. For example, for a battery of 1000 mAh, we obtain that the Hypnos board can report battery charge changes of 0.0001707%, or 585823 different values (power stages) in a range 0 to 1.

The schematics of the Hypnos board circuitry and the firmware of the ATtiny85 microcontroller have been made available in a public repository [51].

B. Hypnos software library

The modular nature of the Hypnos hardware board provides an accurate mechanism to obtain the system's battery charge for those microcontroller boards that provide too course-grained values, or lack this feature altogether. Instinctively, the Hypnos software library provides programmatic access to these values and makes it possible to dynamically adjust the duty-cycle frequency in accordance to the available energy.

In essence, the Hypnos software library provides access to the fine-grained, real-time energy level provided by Hypnos hardware component plugged on the IoT node, and it provides the means to control the wake/sleep time of the IoT node – a widely spread way to save energy in IoT nodes [2]. The more the IoT node sleeps the more its battery life is enlarged, yet at the cost of decreased productivity (e.g. a reduced duty – often sensing – rate).

As the scenarios in which IoT nodes are deployed widely vary, so do the requirements for the adopted energy management strategy. Some require a minimum or maximum amount of measurements per time unit, others favor a reliable functioning over a longer period of time.

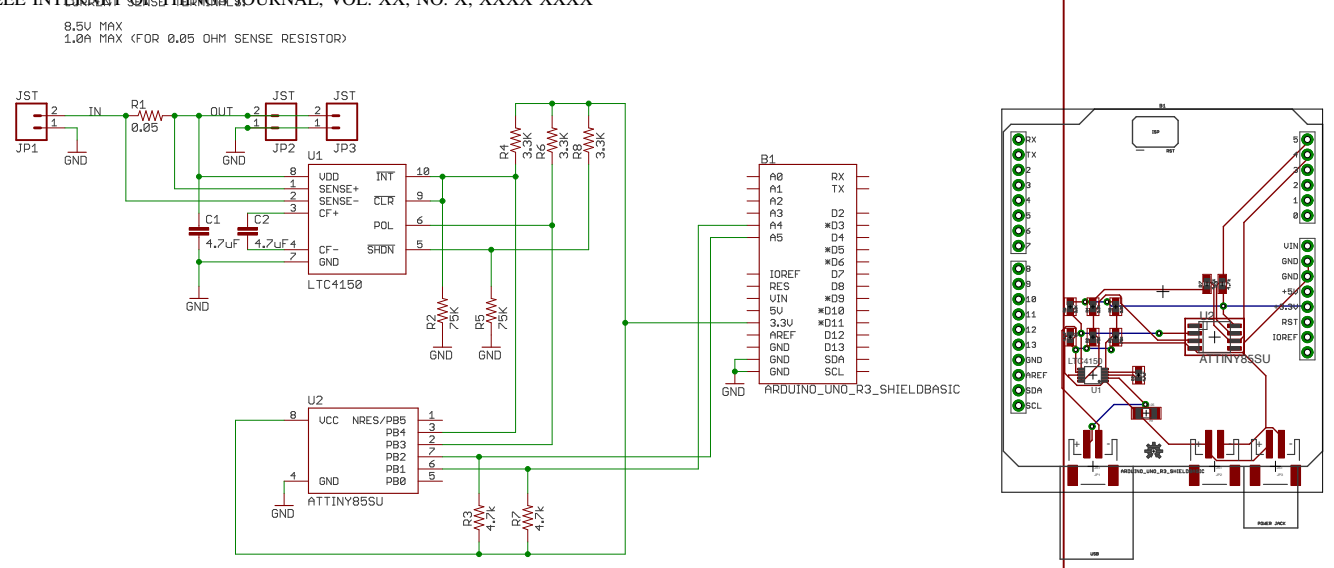
1) Requirements: In order for the Hypnos library to empower the firmware development with the necessary software tools to define fine-grained energy management strategies, the following requirements to develop the software library were taken into account:

- **R1:** It has to offer a way to read raw energy values, as well as energy availability as a percentage in terms of remaining charge (mAh). This allows fine-grained energy readings and, at the same time, a convenient abstraction to handle available energy levels.
- **R2:** It has to allow to put the system in low energy mode (sleep) for an arbitrary amount of time, whereby the amount of sleep time is calculated in real time, based on remaining energy level. This allows to regulate the duty cycle frequency in function of energy availability, or in other words, it provides an energy balancing mechanism.
- **R3:** It has to offer a way to customize the internal energy balancing mechanism, in order to adapt it to the specific requirements of each deployment.
- **R4:** It has to offer a preview on how long the IoT node will sleep before going into low energy mode. This ensures predictability and allows internal householding (e.g., perform critical operations in case of a long foreseen sleep time).

In the next subsections, we indicate for each requirement how it is fulfilled by the Hypnos software library.

2) Balancing energy availability and workload - the Sigmoid function: As explained in Section II, existing energy management strategies for small, low cost single-threaded IoT nodes may consider various parameters as trade-off for energy saving: sensing coverage, data transmission, data loss or sensing rate. In the latter category, in lack of hardware support for energy measuring and/or software support for more sophisticated energy management strategies, existing systems often resort to discontinuous sleep-regulation functions, reducing battery stages to a few concrete states (i.e., disaster management strategies), or fixed sensing rate strategies, both of which are vulnerable to energy depletion under averse conditions. Nevertheless, with quasi continuous energy readings available through the Hypnos hardware component (i.e. at the rate of the microcontroller's clock speed), the duty-cycling frequency of an IoT node can be regulated at a much finer level of detail, using a continuous function.

The Hypnos software library provides a built-in (modified) sigmoid function for this purpose, which can be customised, or replaced with the developers' custom function fitting their particular scenario. Hereby, we partially fulfill requirements R2 and R3. The modified sigmoid function was chosen for several reasons: 1/ the typical non-linear S-shape of the function suits our needs perfectly, resulting in a smooth approximation to minimal and maximum sleep time at both ends, and a smooth transition regulating in-between states; 2/ the function lends itself very well to produce values between 0 and 1, which can be straightforwardly used as a percentage of the maximum allowed sleep time; 3/ the function can be used as-is (with default parameters), yet is easily customizable, modifying



(a) Hypnos board scheme

(b) Hypnos board PCB layout

Fig. 1: Hypnos board EAGLE schemes (following Arduino Shield form factor).

it's slope inclination and displacement. These respectively represent the rate of increase in sleep time and the delay before sleep time is gradually increased. This allows to operate steadily at fixed time intervals, until the battery level falls below a user-defined threshold (displacement) and smoothly transition to lower operational frequency according to a certain rate (slope inclination). Depending on these customizations, a conservative or aggressive energy management strategy can be implemented, as required by the scenario at hand.

The resulting function (1), whose input and output variables and parameters are detailed in Table I, is an inverted sigmoid function, parameterizable in terms of slope inclination and slope start displacement, with centering and scaling of the remaining battery value. The inversion ensures mapping of lower battery values to higher sleep time intervals. Next, the domain of the sigmoid function at which it fully covers the 0-1 range with 2 decimal point precision (i.e., by default approximately -6 to 6), is projected to a -0.5 to 0.5 interval. Then, we decentre the function (i.e. the output value) from the -0.5 to 0.5 domain (i.e. by default around the Y axis) to the 0 to 1 domain, to map it to the percentage of remaining energy (i.e. battery level). Finally, the resulting value has been scaled up by 10 for convenience when adjusting the slope inclination parameter.

$$w = \frac{1}{1 + d \cdot e^{s \cdot (rb - 0.5) \cdot 10}} \tag{1}$$

The modified sigmoid function, used as built-in energy management function in the Hypnos software library.

The visual representation of the modified sigmoid function using the default configuration values (i.e., displacement 20; slope inclination 1.5 – see Table I) can be seen in Fig. 2 (red function).

The default slope displacement (20) and slope inclination (1.5) parameters are rather conservative, which lead to start steadily (i.e. due to slope inclination) increasing sleep times once the system's battery goes bellow approx 70% (i.e. due to displacement) - red function in Fig. 2. As previously stated, this can be configured by the developer to adhere to a wide variety of specific use cases, and the built-in function can be completely replaced by a custom one if required.

As a matter of example, a displacement value of 2 (instead of 20) would start increasing sleep times sooner, i.e. starting

TABLE I: Hypnos' custom sigmoid function parameters

Symbol	Meaning	Type	Description
rb	Remaining battery	Input	Represents the remaining battery percentage of the IoT node. It ranges from 0 (empty) to 1 (full).
d	Displacement	Parameter	A fixed value that allows to displace the start of the sigmoid slope in the x-axis. The higher the value, the further the slope is away from the 100% battery mark, and thus the longer the IoT node will work at the highest frequency (i.e., lowest sleep time) at maximum energy usage. To keep the properties of the original function, this value should always be a positive number ($[0 < d \leq \infty]$). In Hypnos, the displacement value is by default set to 20, but it can be overridden via the developer's API.
s	Slope inclination	Parameter	It allows to increase or decrease the inclination of the sigmoid's slope. The inclination ratio must never go bellow 1 ($[1 \leq s \leq \infty]$) and must be re-adjusted each time the displacement is changed, to keep the properties of the sigmoid function. The higher the value, the steeper the slope inclination, and the more rapidly sleep time will increase once entering the slope. In Hypnos, the slope inclination is by default set to 1.5, but it can be overridden via the developer's API.
w	Weight	Output	Is the resulting adjustment – from 0 (full energy available) to 1 (energy fully depleted) – with respect to the specified minimum and maximum sleep time values. It represents the sleep time, expressed as a percentage of the permitted sleep time range.

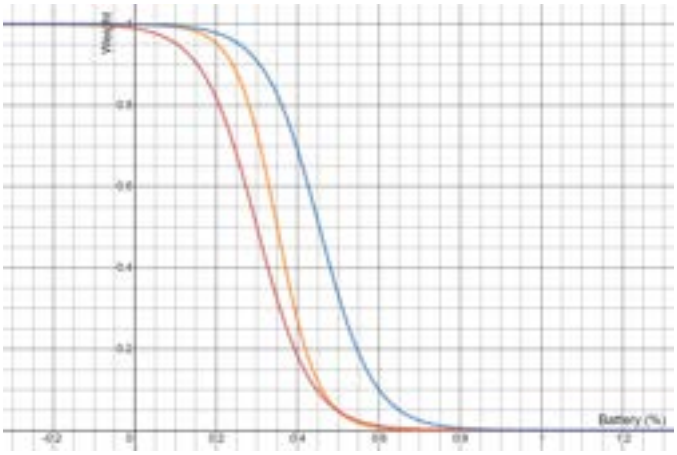


Fig. 2: A comparison of different parameter configurations of the built-in sigmoid function. In red, Hypnos' default sigmoid function (displacement d = 20; slope inclination s = 1.5). In blue, Hypnos' sigmoid function with smaller displacement value (displacement d = 2). In orange, Hypnos' sigmoid function with higher slope inclination (displacement d = 20; slope inclination s = 2).

below approx. 95% (blue function in Fig. 2), whereas a value of 400 would delay increasing sleep times, i.e. starting from below 50% (not shown in Fig. 2). On the other hand, a slope inclination value of 2 (instead of 1.5) would increase the rate at which sleep time is increased (i.e. make the slope steeper) - (orange function in Fig. 2).

Note that slope displacement and inclination parameters are related: significantly increasing the displacement requires increasing the slope as well in order to properly fit within the 0-1 domain.

3) Implementation of the Hypnos software library: As a practical implementation, the Hypnos software library was written in C++, compatible with the node's main microcontroller firmware. It uses the Wire library – available in a wide variety of platforms, e.g. Arduino[11], Mediatek Labs' SDK (LinkIt One)[12] and Particle's Device OS[13] – to communicate with the Hypnos board through the I^2C protocol in order to obtain the current charge counter of the system's battery. Fig. 3 schematically depicts the connection between the Hypnos software and hardware component.

In order to fulfill the requirements for the software library (see Section III-B1), the Hypnos library presents a succinct yet powerful API, enclosed in Hypnos' main C++ class. Table II describes its constructor and all its methods.

To illustrate the simplicity of usage of the Hypnos library, Listing 1 includes a sample code snippet where the basic usage of the Hypnos library is demonstrated.

```
#include <Hypnos.h>
```

[11]https://www.arduino.cc/en/reference/wire
[12]http://labs.mediatek.com/api/linkit-one/Wire.html
[13]https://docs.particle.io/reference/device-os/firmware/electron/#wire-i2c-
[14]https://www.arduino.cc/en/Reference/LowPowerSleep
[15]https://docs.particle.io/reference/device-os/firmware/photon/#sleep-sleep-
[16]https://www.arduino.cc/reference/en/language/functions/time/delay/

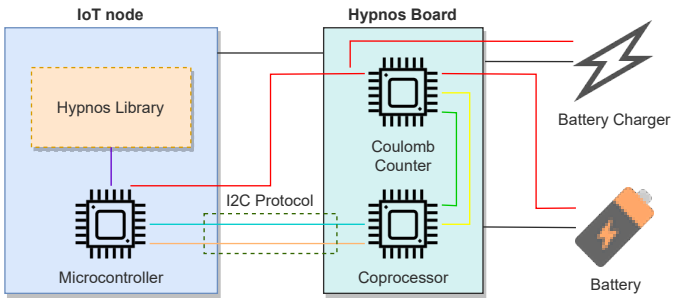


Fig. 3: Conceptual diagram of a Hypnos-powered IoT node

```
#define BATTERY_MAH 1050 // Must be changed in accordance to battery specifications
Hypnos hypnos(BATTERY_MAH);

void setup() {
  Serial.begin(9600); // set data rate per second for serial data transmission
  hypnos.setMinDelayMillis(5*60*1000); // 1 minute by default
  hypnos.setMaxDelayMillis(5*3600*1000); // 12 hours by default
  hypnos.setDelayFunction(&delay); // Using Arduino's delay function as an example
  hypnos.init(); // Library must be initialized after configuration
}

void loop() {
  Serial.print("Remaining mAh: ");
  Serial.print(hypnos.getRemainingCapacity());
  Serial.print(", Remaining %: ");
  Serial.print(hypnos.getRemainingPercentage() * 100.0); // From 0 to 1
  Serial.print(", Sleep time: ");
  Serial.println(hypnos.previewSleepTime()); // In milliseconds
  hypnos.sleep(); // Puts the IoT node to sleep according to sigmoid function calculation
}
```

Listing 1: Hypnos library basic usage sample in an Arduino-like system

For high projected sleep values, the Hypnos library implements an optimization so it can react to sudden energy availability during long sleep times. For those occasions, a threshold sleep value is calculated, which lies at 10% of the range between the minimum and maximum sleep values. If the Hypnos board subsequently goes to sleep for a value higher than two times this value, the threshold value is iteratively used until arriving to the originally projected sleep value (the last iteration is shortened to achieve the original sleep value). At waking up after each of these intermediate iterations, the IoT node does one of two things (by calling the *init* method), depending on the battery level: 1/ in case of further battery discharge (i.e., the expected situation), immediately put the system back to sleep for a next iteration; 2/ in case of sufficient battery recharge, perform a full iteration of the duty-cycle, and re-calculate the sleep time.

In the best scenario, i.e. when some extra energy gets acquired, the node will sleep for just a small fraction of the calculated (long) sleep time, whereas in the worst scenario, i.e. environmental conditions prevent the node to harvest

TABLE II: The Hypnos library API

Method name	Parameters (Type)	Return type	Description
Hypnos	batteryMAh (*integer)*, *sleepData (*SleepData*, optional)	*Hypnos*	The Hypnos library instance constructor. Creates a Hypnos library object with the total battery capacity passed as a parameter. Enables access to the rest of the API methods and allows read-only access to the battery capacity. Optionally, a reference to a persistent sleep data object can be passed, in case of recovering from a crash.
init	None	*void*	Meant to be called once, at the microcontroller's setup / initialization step. Establishes the connection with the Hypnos board via the I^2C protocol and puts the device to sleep immediately if it has been woken up in a long sleep period while the device's battery continues to discharge.
sleep	None	*void*	The method to be called at the end of every duty-cycle iteration. It uses the included *previewSleepTime* method to obtain the duration the system is going to sleep and invokes the sleep function passing the obtained duration as a parameter. Its usage and signature are similar to other methods with a similar purpose, found in many microcontroller's SDKs (see [14] and [15]), thus reducing the learning curve of the library and partially fulfilling requirement **R2**. The other half is fulfilled by the *previewSleepTime* and the *setDelayFunction* methods.
setDelayFunction	*delayFunction (*integer* $\rightarrow$ *void*)	*void*	Must be called before calling the *sleep* method. Allows the developer to specify the underlying (microcontroller board specific) function to be called by the *sleep* method, in order to put the device to sleep. Usually, the function to be passed by parameter is a hardware specific microcontroller SDK function, such as Arduino's *LowPower.sleep()*[14], Particle's *System.sleep()*[15] or the simpler Arduino's *delay()* function[16], or any custom function. The function receives the amount of milliseconds to sleep as a parameter each time the *sleep* method is called. By including this method the library's API aids to fulfill requirement **R2**.
setMinDelayMillis	minDelayMillis (*integer*)	*void*	Allows to override the default minimum sleep time value (60000 ms, i.e. 1 minute). By including this method, the API partially fulfills requirement **R3**. The other half is fulfilled by the *setMaxDelayMillis*, *setSlope* and *setDisplacement* methods.
setMaxDelayMillis	maxDelayMillis (*integer*)	*void*	Allows to override the default maximum sleep time value (43200000 ms, i.e. 12 hours), thus partially fulfilling requirement **R3**.
setSlope	slope (*float*)	*void*	Allows to modify the steepness ratio of the built-in modified sigmoid function used by the *previewSleepTime* method – and in turn, by the *sleep* method. See Section III-B2 for a detailed explanation. The default ratio is 1.5. Allowing to customize the slope inclination of the modified sigmoid partially fulfills requirement **R3**.
setDisplacement	displacement (*float*)	*void*	Allows to displace the start of the built-in sigmoid function slope used by the *previewSleepTime* method – and in turn, by the *sleep* method. The default displacement is 20 units, applied on the X axis. See Section III-B2 for a detailed explanation. This method, jointly with *setMinDelayMillis*, *setMaxDelayMillis* and *setSlope* methods, successfully fulfill requirement **R3**.
previewSleepTime	None	*integer*	Allows to query the amount of time that the system is going to sleep, by evaluating the built-in custom sigmoid function, using the current remaining battery percentage – reported by the *getRemainingPercentage* method. This method, in conjunction with the *sleep* and *setDelayFunction* methods, successfully fulfill the **R2**. In addition, the present method by itself fulfills the **R4**.
getRemainingCapacity	None	*float*	Allows to read the remaining system's battery capacity (in mAh), on demand. It does so by asking the Hypnos board for the accumulated value of LTC4150's charge ticks that have been consumed, then multiplying them by 0.1707 (i.e. the amount of mAh that each tick represents) and deducting the resulting amount from the total battery capacity. This API's method partially fulfills requirement **R1**.
getRemainingPercentage	None	*float*	Allows to read the remaining system's battery percentage on demand, in a continuous float value ranging from 0 to 1. It does so by calling the *getRemainingCapacity* method to obtain the remaining battery charge, then dividing the returned value by the total capacity of the IoT node battery. This method in conjunction with the *getRemainingCapacity* method, successfully fulfill requirement **R1**.

more energy, the node will marginally loose energy (due to performing the lightweight intermediate iterations) while sleeping as foreseen and perform the planned operation cycle.

The Hypnos' library code has been open sourced and granted with a permissive license [52]. This allows firmware developers to use and adapt it to their needs. The code has been structured to seamlessly allow its modification to support new microcontroller boards, replace the connection with the Hypnos board or, in cases it does not fit the specific use case's needs, replace the built-in sigmoid function with a custom function.

C. Integration scenarios

To integrate the Hypnos toolkit in existing microcontroller systems, we envision three different integration scenarios:

- **The main microcontroller board is directly supported.** While designing and developing the Hypnos toolkit, the first prototype was tested in three different platforms: Arduino UNO R3, MediaTek's LinkIt One and Particle's Photon. For the latter, a Shield-Shield adapter is required in order to map the Feather-like headers to the Arduino pin-out. If the main microcontroller uses one of those platforms, Hypnos works out-of-the-box.
- **The main microcontroller board is compatible.** In case a microcontroller board has a compatible pinout (or permits its remapping) and SDK library, the Hypnos toolkit works out-of-the-box as well. This is not an uncommon situation, as certain board manufacturers create their hardware using a similar PCB to the popular and widely used Arduino UNO (e.g. Sparkfun or Alorium[17]). Others offer adapters to map their custom pinout to an Arduino UNO compatible one. Moreover, the development APIs of some of them follow the same interface[18]. As long as those microcontrollers have a 3.3v output pin and the I^2C interface accessible from the two first analog pins (A0 and A1), the Hypnos toolkit will be compatible.
- **The main microcontroller board is not compatible.** In case the microcontroller board has a different pin-out that cannot be remapped and/or its SDK uses a different API, some additional work is needed to still integrate the Hypnos toolkit. In case of an incompatible pin-out, an adapter needs to be designed and assembled, taking into account that the Hypnos board requires a 3.3V power source and a I^2C header to enable data communication between the boards. Hypnos' board original design can also be altered to adjust it to the target system's needs. In case the integration friction originates from an incompatible software API, it needs to be mapped on the Hypnos software library. For example, if the SDK lacks the Wire library, the source code of the software library needs to be modified in order to integrate a different I^2C communication interface. In any case, given the simplicity of interaction of the Hypnos board with the main microcontroller's board, an integration solution is likely to be possible.

[17]https://www.aloriumtech.com/arduino-compatible/

[18]https://www.sparkfun.com/products/13975

IV. Hypnos evaluation

The main goal the Hypnos toolkit is to offer a seamless energy management solution, which allows to improve performance and reliability of small, low-cost, self-sustainable open IoT nodes. In order to evaluate Hypnos, we set up an experiment, whereby two identical IoT nodes ran over a large time period in a real-world setting, one running Hypnos for energy management and the other without it. Theoretically, we expect to see the node running Hypnos to show a more stable battery consumption, ideally reaching a balance between energy recharge and consumption, and (ideally) never running out of battery, regardless of environmental conditions.

In summary, the main goals of the experiment are as follows:

- **O1:** To detect any reliability differences, in terms of continuous up-time and more stable energy consumption, between the two systems in a real-world setting.
- **O2:** To compare the performance, in terms of number of completed duty cycles, of the two systems.

These goals need to be studied in conjunction, as indeed we are seeking a balance between energy consumption and performance, while increasing reliability.

A. Setup

To perform the evaluation, two identical IoT nodes were deployed in identical conditions. Both devices use an assembled Hypnos board (see Fig. 4) to collect the remaining battery level. In order to exclude external factors to influence results, the nodes' complexity was reduced to a minimum, and only implemented a single task to perform: collect atmospheric temperature and send it to a central server. To do so, both IoT nodes used a standard temperature sensor plugged onto the microcontroller board and an on-board WiFi module connected to the Internet through a nearby WiFi Access Point ($\sim$ 3 meters away, with a double partition wall in between). For diagnostics purposes, along with the atmospheric temperature, battery level and sleep value were sent over WiFi to a university server during each duty cycle. Both IoT nodes were powered by a 3.7V Lithium Ion battery, which was charged by a 3W solar panel. A fully assembled node can be seen in Fig. 5, and the detailed bill of materials is shown in Table III.

Software-wise, only one of the two IoT nodes employed the Hypnos library to dynamically regulate sleep time. The other node employed a fixed sleep duration of 5 minutes after each duty-cycle iteration, a common and straightforward implementation in real-world applications in lack of more sophisticated out-of-the-box solutions.

The IoT node running the Hypnos sleep functionality was configured with a minimum sleep delay of 20 seconds, a maximum delay of 12 hours, a slope inclination of 1.5 and a displacement of 5.5. We hereby deliberately reduced the minimum sleep time (from the default 60 to 20 seconds) to stress the battery, and we also reduced the displacement (from 20 to 5.5) to allow the Hypnos energy balancing to kick in sooner (i.e. more conservative strategy). Fig. 6 shows how the balancing function is expected to behave during the experiment based on the previously detailed configuration. Given the

TABLE III: Bill of materials for the IoT sensing node assembled for the experiment

Component	Description
Particle Photon	An ARM Cortex M3 powered microcontroller manufactured by Particle served as the IoT node's microcontroller. It possesses an on-board WiFi chip which allows direct connection to Internet. The IoT nodes run the DeviceOS v0.6.0 system firmware.
Particle Shield-Shield adapter	Was used to map Photon's Feather pin-out to the the Arduino UNO pin-out.
SeeedStudio's Solar Charger Shield v2.2	An energy harvester enclosed in an Arduino Shield form factor, directly placed on top of the Shield-Shield adapter and the Photon. Features a built-in maximum power point tracker (MPPT) algorithm to maximize the efficiency of the solar panel. Outputs up to 600mAh@5V to power the system and recharge a Lithium battery.
3W Solar Panel	The sole energy entry point of the system. Its constrained dimensions (138x160 mm) allow to keep the IoT node reduced in size. It has an energy transformation efficiency of 17%.
3.7V Lithium Ion Battery	The main power storage of the system. It has a capacity of 2200 mAh, enough to power the system for several consecutive cloudy days.
Hypnos Board Prototype	Features the Arduino Shield form-factor and is placed on top of the energy harvester board, wired in the middle of the battery and the rest of the system, including the battery charger.
SeeedStudio's Base Shield V2	Is stacked on top of the Hypnos board and offers 16 Grove ports through the Arduino pin-out. Allows to seamlessly attach a plethora of sensors and actuators.
Grove DHT22 Temperature & Humidity Sensor	A high accuracy temperature and humidity sensor. Presents a Grove interface enclosing a I^2C data communication header.
200x200x100mm Plastic Case	A transparent case to protect the electronic components from water and dust, while at the same time allowing sunlight to pass through.

Fig. 4: A Hypnos board prototype assembled for the experiment

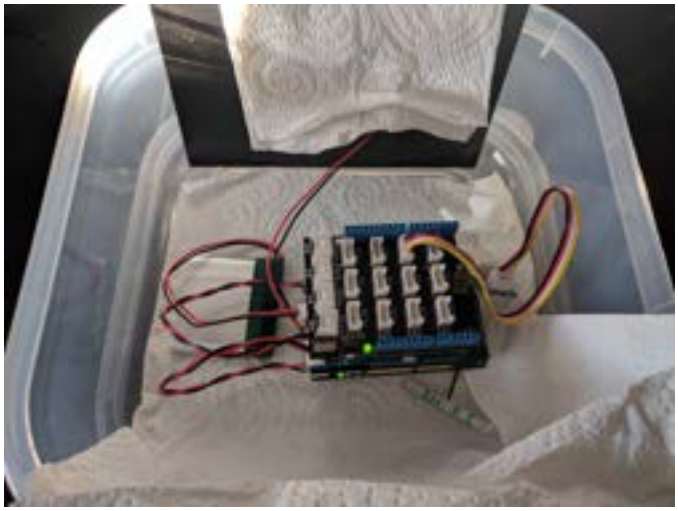


Fig. 5: A fully assembled experiment's IoT node

displacement value of 5.5, the sigmoid function starts to slowly rise below 80% available battery level, increasing sleep time to reach 30 minutes when battery level reaches 60% (see green box in Fig. 6). From 60% downwards, according to the slope inclination of 1.5, the sleep time rises rapidly to two hours (50%) (see orange box in Fig. 6) and above (red box in Fig. 6). This implements a conservative scenario, in which we consider a frequency of over 30 minutes undesirable, yet building in robustness by setting 30 minutes sleep time at 60% battery level. Our aim is thus to start energy balancing around 80%, and allow 60% of battery to compensate for unfavorable conditions (while accepting rapidly increasing penalties between 60% and 20%). The chosen built-in function used to put both devices to sleep was Particle's DeviceOS *System.sleep()* method in deep sleep mode, using retained variables to keep the Hypnos state while sleeping, i.e. the SleepData struct.

At the server side, the server was running a RabbitMQ broker (v3.7.3) with the MQTT plugin, used to define an entry point for the nodes to deliver their readings. The server was also running one instance of Apache Flume (v1.7.0), configured to read from an AMQP source – the RabbitMQ topic receiving the data from the IoT nodes – and write the readings to a MongoDB (v3.6) collection hosted on the same server.

The experiment started on the 30th of April and ended on the 11th of July, running for 71 days. Both nodes ran under identical climatic conditions: they were located outdoors – placed next to each other – in the city of Castellón (Spain), with their solar panels east oriented and tilted at $\sim 30°$. Throughout the duration of the experiment, both nodes received an average of 7 hours of sunlight, with direct incidence during 4 consecutive hours, from approx. 6h30 am to 10h30 am (due to partial coverage by a roof and complete occlusion by a west-placed wall). Cloud conditions further determined the variance in daily solar irradiance. The weather was mostly clear, with notable episodes of cloudy days and sporadic heavy rains (2 or more days of consecutive rains). No exceptional environmental or climatic events occurred during

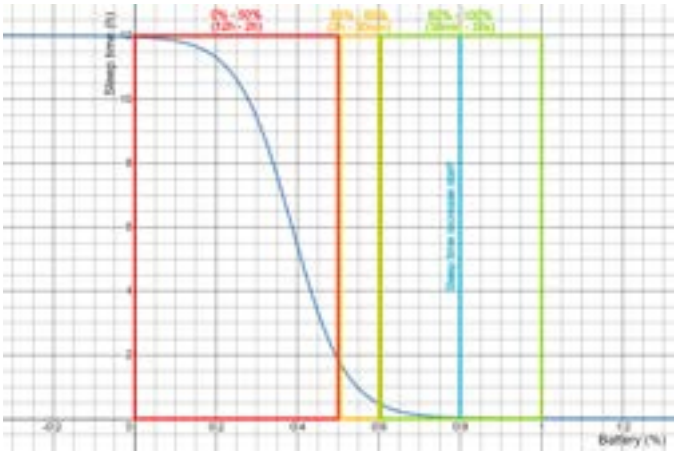


Fig. 6: Experiment's customized balancing function. Represents the amount of hours (h) that the node sleeps depending on its remaining battery level. The three boxes (green, orange, red) show high, medium and low duty cycling frequencies respectively.

the experiment which could interfere with the solar panel's intake (e.g., solar eclipse, volcanic eruption), nor was there any human interference with the devices. Figures 7a and 7b show how both IoT nodes were deployed.

The gathered battery, sleep and temperature data collected from the two IoT nodes were extracted from the MongoDB database via a Python script and converted to a more manageable CSV data format. Solar irradiance data, taken from NASA's POWER dataset [53], for the time span and location of the experiment deployment, were used to complement the IoT measurements in our experiment. Particularly, we used the "All Sky Surface Longwave Downward Irradiance" dataset, which reports solar irradiance (in watts per square meter, W/m^2) taking into account environmental conditions such as cloudiness. The irradiance values for the whole period have been normalized to represent them as percentage of irrandiance (min: 0 W/m^2; max: 980.88 W/m^2).

B. Results

Before processing the results, a data tidying procedure was performed on the obtained dataset. In particular, for the IoT node not running Hypnos, misaligned battery power values were reported each time the device came back online after an unexpectedly shut down due to lack of power. In those cases, the previous battery power value was not updated as the duty cycle did not complete (note that they were correctly updated in the next cycle). In the tidying procedure, these values were set to 0%. This was not a problem in the Hypnos library and furthermore did not influence the experiment. Experiment data and its subsequent analysis have been made available in an online repository [54].

The resulting dataset, summarized using common descriptive statistic values in Table IV, was stored as a CSV file for analysis.

From the device with Hypnos, 34182 battery values were collected over 71 days, after which it was manually shut down. The mean of the battery power values was 81.67%, with a standard deviation of 7.32%. The minimum recorded battery value was 45%, whereas the maximum was 100%. The 50% percentile for battery power was 83%, whereas the Q1 and Q3 values were 77% and 87% respectively.

On the other hand, from the IoT node running without Hypnos, we first need to state that the device depleted its battery after 33 days (after a stretch of low-sun days), and after various failed attempts to boot up again over time, became unresponsive. During those 33 days, 7751 battery values were collected. The mean of the battery power values was 54.05%, significantly lower. The standard deviation shows a noticeable difference too, with a value of 34.79%. The minimum recorded value was of -5%, meaning that the battery (which had slightly more capacity than announced by the manufacturer) was depleted, whereas the maximum value remained identical at 100%. The Q1, Q2 and Q3 values fell in the 16%, 59%, 87% respectively, thus showing a wider dispersion in the recorded values in comparison with the data recorded from the IoT node running Hypnos.

In order to analyze the reliability differences between the two IoT nodes in more detail (**O1**), the extracted battery values are represented as a scatter plot (see Fig. 8), with a different series for each device, i.e. in blue for the device running with Hypnos sleep mechanism, and in red for the device running without it. A trend line has been added to each series to clearly see the tendency of the two series. The remaining battery values of the device running without Hypnos show a clear trend towards 0 (% of remaining battery), whereas the same trend in the values of the device running with Hypnos follow a practically straight line around the 83% value. In addition, a series with the average daily percentage of solar irradiance, represented as a bar chart, was added. We observe that the battery of the device without Hypnos is unable to recover from the first consecutive series of days with low sun irradiance (i.e. cloudy/rainy conditions on 10 - 12 May), and subsequently continues to deteriorate to finally stop on the 3rd of June, during two consecutive low sun irradiance days. On the contrary, the Hypnos board's battery value closely follows the sun irradiance pattern, due to Hypnos' energy management solution.

This latter aspect can be more clearly observed in the distribution plot shown in the Fig. 9. Here, the battery power values recorded from the device running without Hypnos do not follow a clear trend, only showing two probability peaks on the lower and higher battery values, along with a testimonial presence in the middle values. However, the values recorded from the device running with Hypnos follow a clear trend around the 83% of remaining battery, with values ranging from the 60% to the 100% and a higher prevalence in the 80% to 90% range. Note that these concrete values correspond with the displacement configuration parameter set for the experiment (see Section IV-A); a higher displacement value, denoting a more risky strategy, would aim to balance battery level around a lower percentage. We can hereby conclude that, with respect to up-time and energy balancing, the Hypnos IoT node is stable and reliable, more so than the node without Hypnos.

(a) (b)

Fig. 7: The two experiment's nodes deployed on a real-world setting

TABLE IV: Descriptive statistics of the readings acquired during the experiment

Device	Reading type	Reading value							
		Count	Mean	S.D.	Min	Q1	Q2	Q3	Max
Device w\Hypnos	**Battery**	34182	81.67%	7.32%	45%	77%	83%	87%	100%
	Sleep	34182	2 min & 31s	3 min & 56s	25s	50s	1 min & 20s	2 min & 40s	195 min & 16s
	Temperature	34182	30.20 °C	4.75 °C	15.13 °C	27 °C	30.76 °C	34.53 °C	37.49 °C
Device w\o Hypnos	**Battery**	7751	54.05%	34.79%	-5%	16%	59%	87%	100%
	Sleep	7751	5 min	0 min	5 min	5 min	5 min	5 min	5 min
	Temperature	7751	25.8 °C	5.83 °C	14.22 °C	20.83 °C	25.17 °C	29.8 °C	37.3 °C

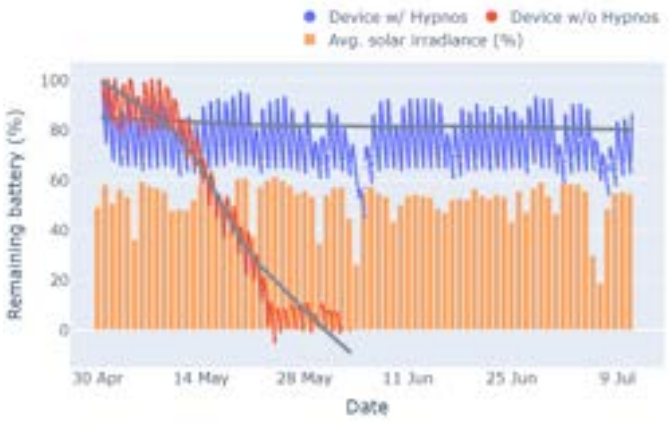


Fig. 8: The recorded battery power values of the two sensing nodes, along with the average daily solar irradiance percentage

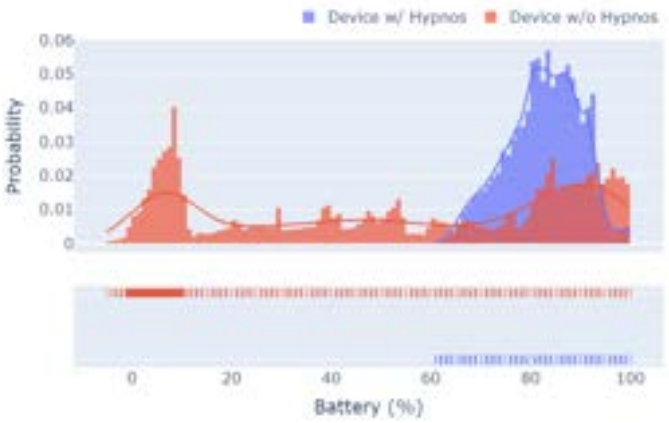


Fig. 9: The battery distribution of the recorded battery values of the two IoT nodes throughout the experiment

An important aspect to offset against the up-time and energy consumption is the time spent sleeping, or in other words, the duty-cycle rate. Fig. 10 shows the collected sleep values as a line chart. Additionally, the average daily percentage of solar irradiance is added as a separate series (bar chart).

Evidently, the sleep value of the device running without Hypnos remains stable around the 5-minute mark. In contrast, the sleep values of the device running with Hypnos show a higher variability, with values ranging from 24 seconds to 3 hours and 15 minutes. Large sleep time spikes visibly correspond with lower solar irradiance (i.e. heavy rain periods and consequent lack of sunlight), whereby the first such spike was responsible for completely depleting the battery from the node without Hypnos due to the lack of sunlight (day 33).

Despite the spikes in sleep time for the IoT node with Hypnos, where the Hypnos algorithm gradually compensated

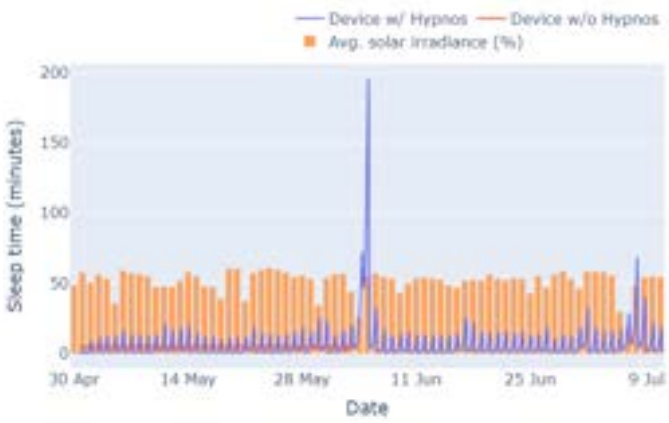


Fig. 10: The sleep times of the two IoT nodes, along with the average daily solar irradiance percentage

for periods of insufficient energy harvesting, the overall frequency of duty cycles is higher. The device running with Hypnos slept 2 minutes and 30 seconds on average, with a standard deviation of 3 minutes and 56 seconds, and Q1, Q2 and Q3 values falling in 0.83, 1.33 and 2.66 minutes respectively (see Table 8), which means that the device running with Hypnos stayed more active, on average, than the device running without it.

This leads us to compare the two systems in terms of performance, i.e. performed duty cycles (**O2**). To achieve this goal, the number of collected temperature readings (one per duty cycle) were aggregated by experiment day, and plotted in a bar chart, see Fig. 11. As a separate series, we also add the average raw solar irradiance value in W/m^2.

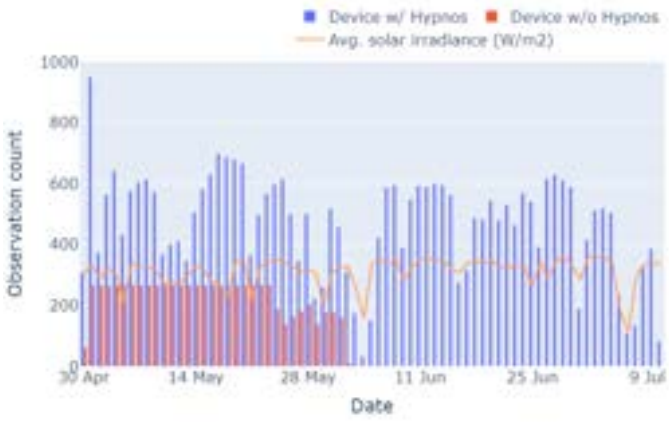


Fig. 11: The number of daily collected observations by the two devices, along with the average daily solar irradiance

Further analysis of this data shows that the device running without Hypnos reported a daily mean of 221 observations, with a standard deviation of 75, a minimum value of 1 and a maximum value of 268. Q1, Q2 and Q3 values fall in 181, 267 and 267 values respectively, meaning that the node has performed an operation every 5 minutes and 23 seconds at least 50% of the days, more or less as expected. Note that at day 33, the node stopped functioning with a depleted battery.

In contrast, for the device running with Hypnos, we visibly notice a clear correlation between the amount of daily collected data and the solar irradiance. As a result, the Hypnos node reported more than two times the daily mean of records, with 468 records, a standard deviation of 169, a minimum value of 33 and a maximum value of 950 reported measurements during the first full day. Q1, Q2 and Q3 values fall in 368, 505 and 590 respectively, meaning that a least 75% of the days the device running with Hypnos has reported more values than the maximum amount reported by the device running without Hypnos. At least 50% of the days, the node running with Hypnos reported an observation every 2 minutes and 51 seconds. This clearly indicates an overall better performance for the IoT node with Hypnos.

Finally, we analyse the distribution of the performed duty cycles (collected amount of temperature measurements) per hour of the day. In order to do so, the series of captured observations from the Hypnos board was first reduced to the same time interval as the non-Hypnos board (which prematurely failed) for a more fair comparison.

The results were plotted as a comparative bar chart, which can be seen in Fig. 12. In addition, the hourly average battery level of the two IoT nodes is depicted in two separate series in order to disclose the relation between performed duty cycles and remaining battery. Furthermore, the average hourly percentage of solar irradiance is also included as a separate series.

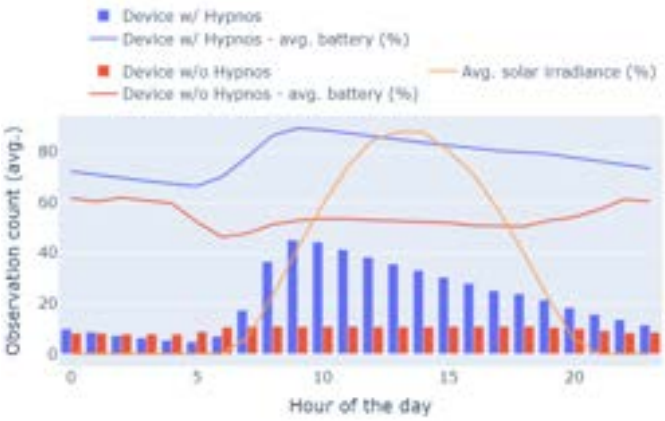


Fig. 12: The hourly distribution of the measurements collected by each device during the experiment, along with their average battery level and solar irradiance percentage

From Fig. 12, we observe that for the node running with Hypnos, (i) the amount of collected values is closely correlated with the average solar irradiance (direct sunlight starting at approx 6h30; shadow starting from approx 10h30 - see IV-A), (ii) the amount of collected values is directly correlated with the battery level. Indeed, the IoT nodes' batteries start recharging at sunrise, which is reflected by an increase in data collection frequency in the Hypnos node, and slowly discharges once the direct sunlight window passed (approx. 10h30). The device running without it does not follow a clear trend. Regarding the distribution of average duty cycles per hour, we observe that during the day and some hours

beyond daylight, the Hypnos IoT node produces vastly more observations. Only from 1 to 7 in nighttime, the IoT node without Hypnos slightly outperformed its counterpart in amount of duty cycles. We hereby need to take into consideration that the IoT node without Hypnos stopped reporting after 33 days, and the aforementioned average observations were calculated over those 33 days. We furthermore note that the node without Hypnos underperforms (w.r.t. the configured 5 minutes frequency) in nightly hours, as it suffers downtimes due to lack of battery.

V. Discussion and Summary

A. Experimental results

The results of the experiment show that the IoT node running Hypnos is reliable and efficient, more so than the IoT node not running Hypnos. The experiment implemented a typical simple strategy deployed in practice (i.e. a fixed sleep time – and sensing rate – of 5 minutes) versus a relatively conservative version of Hypnos (i.e. aiming at maximizing performance at an average $\sim 20\%$ battery usage, i.e., $\sim 80\%$ remaining battery power).

The IoT node running Hypnos performed 2.25 times more operations on average than the one without it, employing the same hardware. Whilst the device running without Hypnos depleted its battery shortly after a month of operation (33 days), the device running with Hypnos predominantly kept its battery in a range of 60% to 100% (45% with bad weather) during the whole experiment, as configured through the modified sigmoid function. Nevertheless, during the night hours (1am - 7am), when no energy intake from the solar panels was available, the Hypnos node's duty cycle rate slowed down (due to the configured sigmoid slope displacement), slightly below the IoT node without Hypnos, fixed to a 5-minute rate. However, this came at the cost of depleting battery and decommissioning of the node without Hypnos due to overspending and insufficient energy intake under adverse power harvesting conditions (i.e. a long series of cloudy days jointly with a few consecutive raining days), while the Hypnos IoT node showed itself resilient and was manually stopped at the end of the experiment.

A more general observation is that both IoT nodes are highly dependent on their configuration and conditions in the real-life deployment. Each particular use case will determine how to balance competing concerns, i.e., performance and reliability. Some use cases require high frequency measurements generally aligned with energy harvesting (e.g. in city air quality monitoring, with higher variability during the day), while others require adapted strategies (e.g. meteorological sensors that require a minimum amount of measurements even under adverse conditions).

B. Toolkit features summary

Considering the Hypnos toolkit as a whole, we summarise the features it offers. On the hardware side, the Hypnos board is a reliable way to continuously obtain energy availability information for the IoT node to operate. The use of well-known standards, such as the I^2C data communication protocol and the Arduino UNO pin-out and its PCB form factor, enables seamless hardware integration in a wide variety of small IoT nodes. The library has been used seamlessly with Arduino, Particle and LinkIt One microcontroller boards, yet due to its open hardware and software nature, it's possible for anyone to customize it, adapt it to other platforms and assemble it [51].

On the software side, the Hypnos library provides a flexible, more fine-grained alternative to existing energy management protocols. Its familiar API, which uses common patterns applied in many widely used microcontroller board's SDKs, smooths the learning curve for firmware developers to use it. It comes with a built-in modified sigmoid energy-balancing function which is in charge of regulating the duty-cycling frequency of the IoT node in order to improve its up-time, duty cycles and overall reliability, always having in mind the system's energy availability. As such, the software library alleviates the burden for the developer to (manually) handle energy management (i.e., wake and sleep times). The sigmoid function can be customized to meet different usage scenarios' needs, allowing for more conservative or aggressive strategies, or be replaced by a custom function in case the offered degree of customization is not sufficient for a specific use case [52].

C. Related work comparison

In Table V, we explicitly compare the features of commensurable works introduced in Section II with Hypnos. We exclude non general-purpose solutions and solutions which cannot be executed on the edge.

From Table V, we observe that Hypnos is the only solution which integrates hardware to measure energy availability, software as unified API to access energy measurements and govern duty cycling, and a built-in algorithmic solution for energy management in small, low cost IoT nodes. Along with ECO [33], it is the only solution whose code and schematics are open-sourced and readily available for use, alter or contribute. ECO, however, is aimed at more sophisticated multi-threaded IoT nodes (and thus performs thread-based energy management), doesn't deploy a dedicated hardware board microprocessor for energy measurements (preventing its more power hungry MCU longer sleep intervals) and doesn't provide an out-of-the-box energy management algorithm. With respect to the algorithmic solution, Hypnos implements a reactive algorithm based on a continuous function (in contrast with Joseph *et al.* [36]), allowing for fine-grained energy management. It's hardware design based on energy measurements at the clock-speed of the independent co-processor doesn't suffer from coarse-grained duty cycling (as in [35]). Hypnos' energy management solution is furthermore not dependent on other (scenario-specific) parameters (such as spatial coverage in [34] or data variability in [30]); it only relies on energy availability. Even though – given its open nature – such parameters could be integrated in Hypnos' energy management function, out-of-the-box, Hypnos here trades generality for case-specific optimizations.

[19]System's SoC is automatically obtained by the Hypnos' software without needing the developer to explicitly obtain it.

[20]Default values are provided.

TABLE V: Comparison of Hypnos with existing solutions

Solution	Components	Code availability	Node can sleep	Sleep value nature	Input requirements	Config options	Drawback
Vigorito *et al.* [34]	Reactive Algorithm	✗	✓	Continuous	Min. and max. duty frequency, battery level	Variance	Unpredictable duty cycling
Joseph *et al.* [36]	Reactive Algorithm	✗	✓	Discontinuous	Harvested energy, queued packets, delivered packets, packets' utility statistics	Energy, data and sleep buffer sizes, buffer slot timespan	Data loss
Le *et al.* [35]	Reactive Algorithm	✗	✓	Continuous	Max. and current capacitor capacity, capacitor voltage, max. sleep time	Proportional integral derivative parameters	Coarse duty cycling
Kulau *et al.* [30]	Reactive Algorithm	✗	✓	Continuous	Current sensed value, observed sensed value history, max. wait time	Wait time exponent	Battery drain under prolonged data variability
Kansal *et al.* [37]	Predictive Algorithm	✗	✓	Discontinuous	Min., max. and initial duty cycling frequency, current available energy, energy availability history	Harvested energy and sleep buffer sizes, buffer slot timespan	Non-optimal duty cycling
Buchli *et al.* [38]	Predictive Algorithm	✗	✗	N/A	Solar panel location, orientation and inclination, average system energy consumption, battery capacity, voltage and max. current draw, harvested and consumed energy	Average meteorological conditions	Underperforming in more favorable conditions
Pro-Energy-VLT [39]	Predictive Algorithm	✗	✗	N/A	Harvested energy history	Number of time slots per day, number of days stored, observations considered for predictions	Underperforming in more favorable conditions
SARSA [40]	Predictive Algorithm	✗	✗	N/A	Battery level, distance from neutral battery level, harvested energy, weather forecast	Reinforcement model convergence rate	Underperforming in more favorable conditions
PreAct [33]	Predictive Algorithm	✗	✗	N/A	Solar panel location and dimensions, defined utility at specific times of day, harvested energy, system energy consumption, state of charge, ideal state of charge, target state of charge, ideal energy consumption, battery capacity	Number of time slots per day	Unpredictable duty cycling
EmRep [41]	Predictive Algorithm	✗	✓	Continuous	Solar panel location and dimensions, defined utility at specific times of day, harvested energy, solar panel efficiency, capacitor state and specs (voltage, size, leakage current)	Number of time slots per day	Unpredictable duty cycling
ECO [49]	Hardware/ Software	✓	✗	N/A	–	I^2C bus mode	Continuous uptime
SPOT [46]	Hardware/ Software	✗	✓	N/A	–	–	N/A
iCount [47]	Hardware/ Software	✗	✓	N/A	–	–	N/A
Nemo [48]	Hardware/ Software	✗	✓	N/A	–	–	N/A
Hypnos	Hardware/ Software/ Reactive Algorithm	✓	✓	Continuous	Battery capacity, node's sleep function reference [19]	Min. and max. delay, sigmoid function's displacement and slope parameters[20]	Unpredictable duty cycling

With respect to predictive algorithms, Hypnos shares the same trade offs with other reactive algorithms: easier setup and configuration, less processing-intensive (e.g., statistical analysis over historical data, machine learning and/or predictive models) or dependent on external sources (e.g., solar or wind maps/predictions), versus better adjustment to "normal" (predicted) scenarios or under utility-variant conditions (i.e. high sensing necessity independent of energy availability). Indeed, under highly predicable (which sustainable energy harvesting solutions typically are not) or utility-specific conditions, predictive models may be better suited than purely reactive systems.

Regarding Hypnos' ease of setup, it only requires the capacity of the system's battery and the sleep function reference of the underlying MCU. Hereby note that even though other solutions do not require the latter, as they do not provide an out-of-the-box energy management solution, instead they require the sleep management system to be implemented. As other systems that rely on dynamic duty cycling – and by extension, reactive algorithms – one drawback is an unpredictable wake time of the IoT node. While this is acceptable in many scenarios, in some it is not (e.g., time-specific measurements, utility-based measurements). In such cases, Hypnos's extensibility allows to integrate such scenario-specific constraints.

D. Integrability

Next we explore how Hypnos could integrate or be integrated in existing solutions at algorithmic, software and hardware level. Given the Hypnos' built-in sleep function is interchangeable, any sleep regulation function that relies on energy availability readings could replace Hypnos' built-in sleep regulation function. From the identified related work, the algorithms proposed by [33]–[35], [40], [41] could thus be implemented and used. For algorithms which do not rely on energy intake/usage, such as [30] which only considers the volatility of sensed data, integrating Hypnos' sigmoid function could add the additional dimension of achieving a good balance between data quality and battery depletion prevention.

At a software level, Hypnos could furthermore be integrated with cloud and fog-based solutions, by using Hypnos at IoT node (edge) level in harmony with a global, network-based solution. For example, Hypnos could be used as a backup in case the central (fog) actor calculating the sleep time of each node becomes unavailable, as in [42], [44], or to calculate and report sleep time locally at IoT network node level to report it to the (closest) gateway as in [43].

At the hardware level, we provide a pluggable solution, yet in case a coulomb counting chip is available in the microcontroller board (see Section III-C), it can replace the Hypnos hardware board, only requiring to replace the reference to the energy measurement function in the Hypnos software library.

E. Limitations

The presented work has some limitations, mainly in the extent of the evaluation. The experiment only tested one specific setup, which aimed to demonstrate Hypnos' functioning and features, set off against a typical fixed rate alternative. However, many configurations, both for the Hypnos and non-Hypnos board, are possible and could be tested and compared. Even though we only tested Hypnos with solar panels, alternative energy harvesting technologies (e.g., based on hydraulic or eolic power) are possible, and we expect that – by design – the Hypnos energy management strategy would be unaffected. Finally, while we explicitly reported on the relationship between climatic conditions (cloud cover) and the experiment results, and set up the experiment to exclude external factors by using an identical setup, we acknowledge that the working of the microcontroller boards may still be influenced by external factors specific to each experiment (e.g., dust accumulation on solar panels). A wider range of experiments, testing different configurations according to different use cases (i.e., conservative versus aggressive) comparing with other compatible algorithms (see next subsection), under various conditions (i.e., climatically-different environments or crowded areas) and using different energy harvesting technologies, could provide further insights in the robustness, efficiency and reliability of Hypnos.

VI. Conclusion

In the Internet of Things' ample subdomain of small, stand-alone, self-sustaining single-threaded IoT nodes, energy optimization strategies are essential to ensure their prolonged, reliable functioning. In this article, we presented Hypnos, an integrated hardware and software solution to balance energy availability and consumption for such IoT nodes. Hardware-wise, Hypnos offers a hardware board, pluggable on any microcontroller board supporting the de facto Arduino pinout standard, which delivers reliable, continuous energy availability readings. Software-wise, Hypnos offers a software library, which provides API access to the energy measurements provided by the hardware board, along with a built-in, configurable sigmoid-based energy balancing function which regulates the IoT node's duty cycle (sensing) rate, with zero (default parameters) or low configuration (customizing the Sigmoid function's slope inclination and displacements – respectively representing the rate of increase in sleep time and the delay before sleep time is gradually increased). Hereby, Hypnos ensures low effort, reliable, fine-grained energy management. Hypnos resides at the IoT computation layer, where it operates at the level of an independent IoT node and is agnostic of data transmission or application-specific parameters. Nevertheless, Hypnos' default sigmoid-based balancing function can be customised, or replaced altogether, to include support for domain- and application-specific scenarios. Being agnostic from other IoT layers, Hypnos can thus integrate or be integrated with other energy management solutions, either by replacing its built-in sigmoid function with existing energy balancing functions, or cooperate with fog/cloud-based solutions at node level. Overall, Hypnos provides an integrated, open hardware and software solution to continuously collect and react upon energy availability, hereby improving reliability, in terms of more stable energy consumption and uptime, and performance, in terms of average amount of sensor measurements.

Hypnos was validated in a two month experiment in real-word conditions. The experiment showed that the Hypnos board was able to cope with unstable energy harvesting, including adverse conditions (i.e., several cloudy/raining days, preventing sufficient energy intake via solar panels), and overall provides a reliable, efficient solution through energy-aware variable rate sensing. Hypnos's energy balancing function is furthermore customizable, allowing more aggressive (trading increased sensing rate for a reduced buffer against prolonged low energy availability) or conservative (trading a faster decrease of sensing rate for an increased energy buffer to promote prolonged reliability) energy-usage strategies, depending on the sensing needs of the particular application.

Reproducibility and availability

Data collected during the experiment, data tidying procedures, analyses scripts and resulting figures are available online in a public repository [54].

The Hypnos software library code and the schematics of its hardware counterpart are available online with a permissive license, thus allowing their reuse and modification [51], [52].

Acknowledgment

Alberto González-Pérez is funded by the Spanish Ministry of Education, Culture and Sports (grant reference FPU17/03832). Sven Casteleyn was partly funded by the Ramon y Cajal Programme of the Spanish government (grant reference RYC-2014-16606).

References

[1] K. Ashton, "That 'internet of things' thing," *RFID Journal*, 1999.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, 2014.

[4] "Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020," https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io, (Accessed on 07/24/2020).

[5] P. Sethi and S. R. Sarangi, "Internet of things: Architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, pp. 1–25, 2017.

[6] V. Scuotto, A. Ferraris, and S. Bresciani, "Internet of things: applications and challenges in smart cities: A case study of ibm smart city projects," *Business Process Management Journal*, vol. 22, no. 2, pp. 357–367, 2016.

[7] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454–1464, 2017.

[8] M. S. Hossain and G. Muhammad, "Cloud-assisted industrial internet of things (iiot) - enabled framework for health monitoring," *Computer Networks*, vol. 101, pp. 192–202, 2016.

[9] A. Tzounis, N. Katsoulas, T. Bartzanas, and C. Kittas, "Internet of things in agriculture, recent advances and future challenges," *Biosystems Engineering*, vol. 164, pp. 31–48, 2017.

[10] F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm," in *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, 2014, pp. 697–701.

[11] A. J. Trappey, C. V. Trappey, U. H. Govindarajan, A. C. Chuang, and J. J. Sun, "A review of essential standards and patent landscapes for the internet of things: A key enabler for industry 4.0," *Advanced Engineering Informatics*, vol. 33, pp. 208–229, 2017.

[12] B. Vogel, Y. Dong, B. Emruli, P. Davidsson, and R. Spalazzese, "What is an open iot platform? insights from a systematic mapping study," *Future Internet*, vol. 12, no. 4, p. 73, 2020.

[13] K. J. Singh and D. S. Kapoor, "Create your own internet of things: A survey of iot platforms." *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 57–68, 2017.

[14] F. Montori, L. Bedogni, and L. Bononi, "A collaborative internet of things architecture for smart cities and environmental monitoring," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 592–605, 2018.

[15] W. Buytaert, A. Dewulf, B. de Bièvre, J. Clark, and D. Hannah, "Citizen science for water resources management: Toward polycentric monitoring and governance?" *Journal of Water Resources Planning and Management*, vol. 142, no. 4, p. 1816002, 2016.

[16] J. Laut, E. Henry, O. Nov, and M. Porfiri, "Development of a mechatronics-based citizen science platform for aquatic environmental monitoring," *IEEE-ASME Transactions on Mechatronics*, vol. 19, no. 5, pp. 1541–1551, 2014.

[17] S. Trilles, A. González-Pérez, and J. Huerta, "A comprehensive iot node proposal using open hardware. a smart farming use case to monitor vineyards," *Electronics*, vol. 7, no. 12, p. 419, 2018.

[18] Y. Chen and D. Han, "Water quality monitoring in a smart city: a pilot project," *Automation in Construction*, vol. 89, pp. 307–316, 2018.

[19] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani, "Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 10–16, 2017.

[20] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[21] P. Asghari, A. M. Rahmani, and H. H. S. Javadi, "Internet of things applications: A systematic review," *Computer Networks*, vol. 148, pp. 241–261, 2019.

[22] H. Khodr, N. Kouzayha, M. Abdallah, J. Costantine, and Z. Dawy, "Energy efficient iot sensor with rf wake-up and addressing capability," *IEEE sensors letters*, vol. 1, no. 6, pp. 1–4, 2017.

[23] M. M. Macías, C. J. G. Orellana, H. M. G. Velasco, A. G. Manso, J. E. A. Garzón, and H. S. Santamaría, "Gas sensor measurements during the initial action period of duty-cycling for power saving," *Sensors and Actuators B: Chemical*, vol. 239, pp. 1003–1009, 2017.

[24] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (iot) communication protocols," in *2017 8th International conference on information technology (ICIT)*. IEEE, 2017, pp. 685–690.

[25] A. Triantafyllou, P. Sarigiannidis, and T. D. Lagkas, "Network protocols, schemes, and mechanisms for internet of things (iot): Features, open challenges, and trends," *Wireless communications and mobile computing*, vol. 2018, 2018.

[26] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Gürkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, S. Mangard, and L. Benini, "An iot endpoint system-on-chip for secure and energy-efficient near-sensor analytics," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2481–2494, 2017.

[27] V. S. Patil, Y. B. Mane, and S. Deshpande, "Fpga based power saving technique for sensor node in wireless sensor network (wsn)," in *Computational Intelligence in Sensor Networks*. Springer, 2019, pp. 385–404.

[28] R. Arshad, S. Zahoor, M. A. Shah, A. Wahid, and H. Yu, "Green iot: An investigation on energy saving practices for 2020 and beyond," *IEEE Access*, vol. 5, pp. 15 667–15 681, 2017.

[29] M. S. Kiran, P. Rajalakshmi, K. Bharadwaj, and A. Acharyya, "Adaptive rule engine based iot enabled remote health care data acquisition and smart transmission system," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, 2014, pp. 253–258.

[30] U. Kulau, J. van Balen, S. Schildt, F. Büsching, and L. Wolf, "Dynamic sample rate adaptation for long-term iot sensing applications," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, 2016, pp. 271–276.

[31] M. Coulibaly, A. Errami, and E. Sabir, "Occupancy-aware power saving for smart parking iot sensors," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 2020, pp. 1–4.

[32] T. N. Gia, V. K. Sarker, I. Tcarenko, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Energy efficient wearable sensor node for iot-based fall detection systems," *Microprocessors and Microsystems*, vol. 56, pp. 34–46, 2018.

[33] K. Geissdoerfer, R. Jurdak, B. Kusy, and M. Zimmerling, "Getting more out of energy-harvesting systems: Energy management under time-varying utility with preact," in *Proceedings of the 18th International*

Conference on Information Processing in Sensor Networks, 2019, pp. 109–120.

[34] C. M. Vigorito, D. Ganesan, and A. G. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *2007 4th Annual IEEE communications society conference on sensor, mesh and ad hoc communications and networks*. IEEE, 2007, pp. 21–30.

[35] T. N. Le, O. Sentieys, O. Berder, A. Pegatoquet, and C. Belleudy, "Power manager with pid controller in energy harvesting wireless sensor networks," in *2012 IEEE International Conference on Green Computing and Communications*. IEEE, 2012, pp. 668–670.

[36] V. Joseph, V. Sharma, and U. Mukherji, "Optimal sleep-wake policies for an energy harvesting sensor node," in *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–6.

[37] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 4, pp. 32–es, 2007.

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

[39] A. Cammarano, C. Petrioli, and D. Spenza, "Online energy harvesting prediction in environmentally powered wireless sensor networks," *IEEE Sensors Journal*, vol. 16, no. 17, pp. 6793–6804, 2016.

[40] S. Shresthamali, M. Kondo, and H. Nakamura, "Adaptive power management in solar energy harvesting sensor node using reinforcement learning," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–21, 2017.

[41] L. Hanschke and C. Renner, "Emrep: Energy management relying on state-of-charge extrema prediction," *IET Computers & Digital Techniques*, 2021.

[42] J. Wang, D. Li, G. Xing, and H. Du, "Cross-layer sleep scheduling design in service-oriented wireless sensor networks," *IEEE transactions on mobile computing*, vol. 9, no. 11, pp. 1622–1633, 2010.

[43] M. Taneja, "A framework for power saving in iot networks," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2014, pp. 369–375.

[44] H. Vo, "Implementing energy saving techniques for sensor nodes in iot applications," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 5, no. 17, 2018.

[45] L. Technology, "Ltc4150 - coulomb counter/battery gas gauge - analog devices," 2003. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/4150fc.pdf

[46] X. Jiang, P. Dutta, D. Culler, and I. Stoica, "Micro power meter for energy monitoring of wireless sensor networks at scale," in *2007 6th International Symposium on Information Processing in Sensor Networks*. IEEE, 2007, pp. 186–195.

[47] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler, "Energy metering for free: Augmenting switching regulators for real-time monitoring," in *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*. IEEE, 2008, pp. 283–294.

[48] R. Zhou and G. Xing, "Nemo: A high-fidelity noninvasive power meter system for wireless sensor networks," in *2013 ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2013, pp. 141–152.

[49] M. Rottleuthner, T. C. Schmidt, and M. Wählisch, "Sense your power: The eco approach to energy awareness for iot devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 3, pp. 1–25, 2021.

[50] M. Grusin, "LTC4150 Breakout Board," https://cdn.sparkfun.com/datasheets/BreakoutBoards/LTC4150_BOB_v10.pdf, 2014, [Online; accessed 30-July-2019].

[51] A. González-Pérez, "Hypnos Board - Energy-Aware Sensing in Low-Cost IoT Nodes," Jan. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4457994

[52] ——, "Hypnos Library - Energy-Aware Sensing in Low-Cost IoT Nodes," Jan. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4457980

[53] A. H. Sparks, "nasapower: A nasa power global meteorology, surface solar energy and climatology data client for r," *The Journal of Open Source Software*, vol. 3, no. 30, p. 1035, oct 2018.

[54] A. González-Pérez, "Reproducibility Package for "Hypnos: a Hardware and Software Toolkit forEnergy-Aware Sensing in Low-Cost IoT Nodes"," Dec. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5790070

Alberto González-Pérez is currently working towards the Ph.D. degree at the Geospatial Technologies Research Group, part of the Institute of New Imaging Technologies at the Universidad Jaime I, Spain. His research focuses on mobile computing, cloud computing and IoT. Contact him at alberto.gonzalez@uji.es.

Sven Casteleyn is an associate professor at the Geospatial Technologies Research Group, part of the Institute of New Imaging Technologies at the Universidad Jaime I, Spain. His research focuses on Web and mobile systems, geographical information science & technology and their application fields, and (mental) health informatics. Contact him at sven.casteleyn@uji.es.