Abu Bakar, Alexander G. Ross *Northwestern University, Evanston, IL, USA*
Kasim Sinan Yildirim *University of Trento, Trento, TN, Italy*
Josiah Hester *Northwestern University, Evanston, IL, USA*

Editors: Nicholas D. Lane and Xia Zhou

HEURISTIC ADAPTATION FOR INTERMITTENTLY POWERED BATTERYLESS SENSORS

Excerpted from "REHASH: A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing," from *Proceedings of the ACM on IMWUT* with permission. https://dl.acm.org/doi/abs/10.1145/3478077 ©ACM 2021

This research is based upon work supported by the National Science Foundation under award numbers CNS1850496, CNS-2032408, CNS-2038853, CNS-2030251, and CNS-2107400. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Battery-free sensing devices harvest energy from their surrounding environment to perform sensing, computation, and communication. A core challenge for these devices is maintaining usefulness despite erratic, random, or irregular energy availability, which causes inconsistent execution, loss of service, and power failures. Adapting execution (degrading or upgrading) based on available or predicted power/energy seems promising to stave off power failures, meet deadlines, or increase throughput. However, due to constrained resources and limited local information, deciding *what* and *when* exactly to adapt is challenging. This article explores the fundamentals of energy-aware adaptation for intermittently powered computers and proposes heuristic adaptation mechanisms to dynamically modulate the program complexity at run-time to enable higher sensor coverage and throughput. While we target battery-free, intermittently powered, resource-constrained sensors, we see a general application to all energy harvesting devices.

Photo, istockphoto.com

Realizing the vision of smart dust [2], TerraSwarm, and ubiquitous computing requires sensors to harvest ambient energy to ensure long lifetimes and scalable deployments with low maintenance costs. By leaving the batteries behind, the environmental impact of disposal is reduced, and sensor lifetimes are further increased since all batteries, even rechargeables, must eventually be replaced. Instead of batteries, these devices collect harvested energy into a capacitor and run programs only when the capacitor charge is sufficient to keep the device operating. When the stored energy is consumed, the device switches off until the next charging cycle is complete, resulting in very short active periods (e.g., a few milliseconds) and usually extended power outages (e.g., hours). When the energy is fully drained from the capacitor during computation, the device dies, resetting the volatile state (registers, stack, program counter, peripherals, time, etc.), preventing forward progress of computation and potentially corrupting memory.

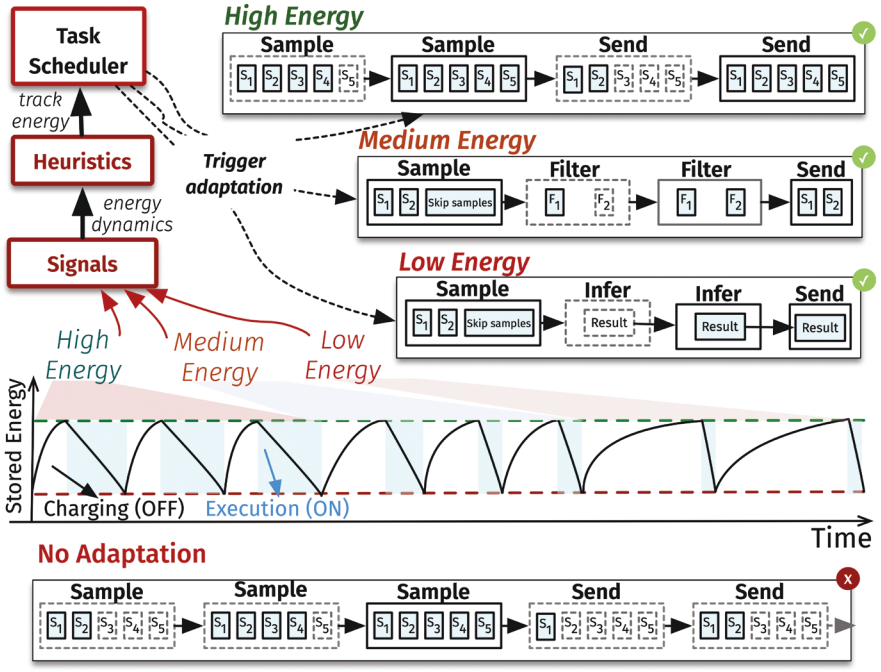

FIGURE 1. A simple sense and send application reads the temperature sensor five times and sends data to a base station. Energy harvesting causes power outages. Without adaptation (bottom), the application fails to report full information. Adaptive execution (top) quickly adapts to changing environmental conditions and finishes off quickly – using harvested energy efficiently by gracefully modifying execution.

CURRENT RESEARCH IN INTERMITTENT COMPUTING

Software systems for intermittent computing preserve forward progress and ensure memory consistency by inserting checkpoints throughout program code [3]. At each checkpoint, or when a power failure approaches, the volatile content is pushed to non-volatile storage (usually a FRAM). Other approaches employ task-based programming models [4, 5], where developers are expected to manually define tasks that are guaranteed to execute atomically. Tasks have all-or-nothing semantics, i.e., they either run to completion by committing their output to the non-volatile memory or have no effect on the program state if their execution is interrupted by a power failure. The design goal of all these systems is to ensure forward progress while maintaining memory consistency, but when power failures and the length of power outages are higher than usual, these systems fail to produce useful outputs within a reasonable time and waste vital energy on saving the current program state. What is needed is a mechanism for adaptation so that computation complexity can be degraded or upgraded depending on the available energy to ensure that something useful happens.

ADAPTATION FOR INTERMITTENT COMPUTING

Adaptation is not a new concept in mobile computer systems; it is a core factor of numerous classes of devices. Concepts like imprecise computation and approximate computing in real-time systems are highly related and have been explored since at least the early 1990s [6]. More recent examples include throttling background apps on android [7] and wearable devices that discard video frames or reduce resolution when battery life is low [8]. Adaptation in intermittent computing devices faces different challenges than those listed, as they suffer from frequent power failures, have severely constrained resources compared to any other adaptive system, and are usually limited in capability because of optimizing for cost.

Challenge 1: When to Adapt

The practicality of deciding when to adapt is crucial. It is hard to estimate the overall energy availability in an environment, much less predict future energy, even for repetitive harvesting situations, since intermittent devices are operating on the margins of energy. While every environment has different energy harvesting modes or trends [9] (e.g., diurnal shifts, activity-based kinetic harvesting), finding a general method to discern the energy environment is challenging because of the diversity of devices. Implementing things like maximum power point tracking and advanced circuitry and computation to record and analyze trends at run-time in the energy harvesting environment requires significant energy, space, and time. Critically, these methods are not power failure resilient, meaning that data cannot be sampled regularly due to the power failures that could lead to incorrect predictions about the energy harvesting environment.

Challenge 2: What to Adapt

Responding to energy availability (by degrading or upgrading performance) should be configurable to provide flexibility and control to the programmer. Programmers should have the freedom to change (or not) any task or task variable, at once or in sequence. For instance, rather than degrading one variable, e.g., the number of sensor samples, the programmer might prefer an adaptation scheme where the number of samples is increased, but the parameter related to the radio's transmission power is decreased. Also, the notion of performance is different for every application. For example, many battery-free sensing applications can be seen as composed of a SENSE → COMPUTE → SEND loop. In a high-energy mode, the application can perform SENSE → SEND type execution without any adaptation. When the energy is moderate, the application can switch to SENSE → FILTER → SEND type execution (Figure 1). Fine-grained adaptation can be supported in this mode by decreasing the number of samples, filtering out data, etc. And finally, the application can switch to SENSE → INFER → SEND → INFERENCE type execution in a low-energy mode. Providing a general framework for enabling this broad array of adaptation techniques is challenging. Having numerous options gives decision fatigue and, more importantly, obscures the difference in performance from different selections.

REHASH: Heuristic Adaptation Platform

We have devised *REHASH*, a software framework, runtime system, and user-facing tool for developing intermittent applications that respond to dynamic energy availability and provide the best quality of service. One of the main features of REHASH is employing heuristic adaptation, which uses signals stemming from intermittent execution caused by energy arrivals and dynamic energy availability in the environment. Based on these signals, REHASH uses a preset or developer-defined heuristic function to estimate the current energy availability of the environment with low overhead. The output of the heuristic function gives the *adapt-up* or *adapt-down* decision (Figure 2). Based on these decisions, the REHASH task scheduler triggers the adaptation strategy that implements the necessary task upgrade/degrade operation accordingly. Adaptation takes into account the various *knobs* that are indicated by the developer. Then, the task scheduler executes the next (adapted) task: ensuring the forward progress of the computation and, meanwhile, responding to the energy dynamics. Application developers have the domain knowledge to specify an adaptation strategy, which will guide how tasks change based on the *up/down* signal.

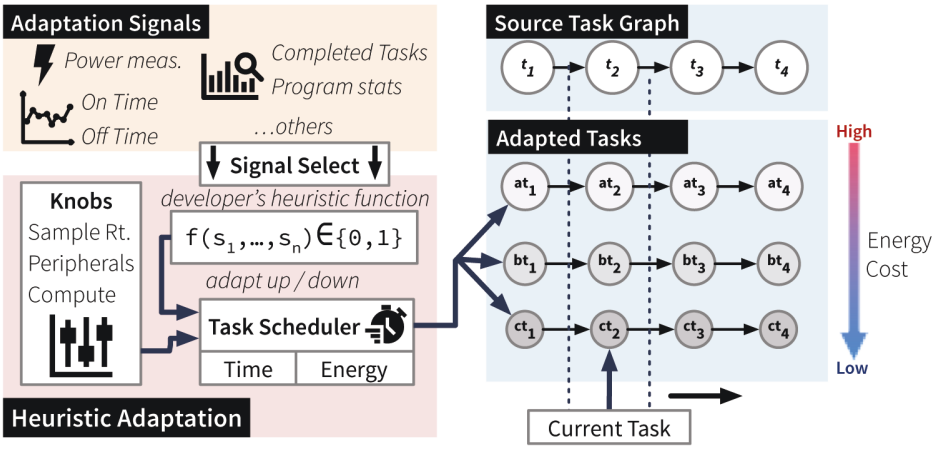

FIGURE 2. Overview of our REHASH framework for heuristic adaptation. The hardware measures signals of the state of the energy harvesting environment, such as on-time, off-time, task count, and recent events. A developer-defined heuristic function takes these signals into a logic equation that decides whether to adapt up (increasing energy use) or degrade performance (decreasing energy use). The heuristic function embodies the developers' application goals. The next adapted task replica is chosen, while the task scheduler executes the next (adapted) task.

Goals and Design Features of REHASH

REHASH aims to enable adaptation for intermittent systems in four key ways.

1. Portable energy approximation: It is difficult to provide a general model to estimate energy consumption and harvesting, considering various hardware and energy harvesting circuitry. Heuristic adaptation offers an all-around and portable way to estimate energy availability and make adaptation decisions.

2. Many signals: REHASH allows the selection and integration of many signals for adaptation. These signals can be inputs into a transform to account for history, for example, using functions like min/max/average/EWMA to get the best decision for the adaptation.

3. Many knobs: REHASH allows the programmer to select among many *knobs* to change based on energy availability. These knobs include sampling rates, transmit power, neural network depth, peripherals configuration, etc.

4. Flexible and fine-grained adaptation control: REHASH allows the programmer to control the overall adaptation strategy, including the type and level of code degradation, the particular tasks to adapt or not, and the signals to trigger adaptation. *When, where, and how* to adapt (or not) is completely controlled by the programmer.

Signals: Tracking the E nergy Dynamics

Before a system can adapt, it has to have some reliable mechanism for understanding the energy harvesting environment. REHASH relies on signals that are a statistical measure or count of the last execution before a power failure. These signals change in response to the energy harvesting environment and the energy demand of the application. Many adaptation signals can be integrated into our REHASH framework. The time interval length from the time the capacitor is fully charged and the device starts operating to the time the device fails (i.e., the *on-time*) can be exploited as an adaptation signal. *On-time* captures the relation between the incoming energy rate and the energy demand of the application. If the incoming energy is high or the tasks of the application require less energy to execute, one can expect a longer *on-time*. The inverse is true

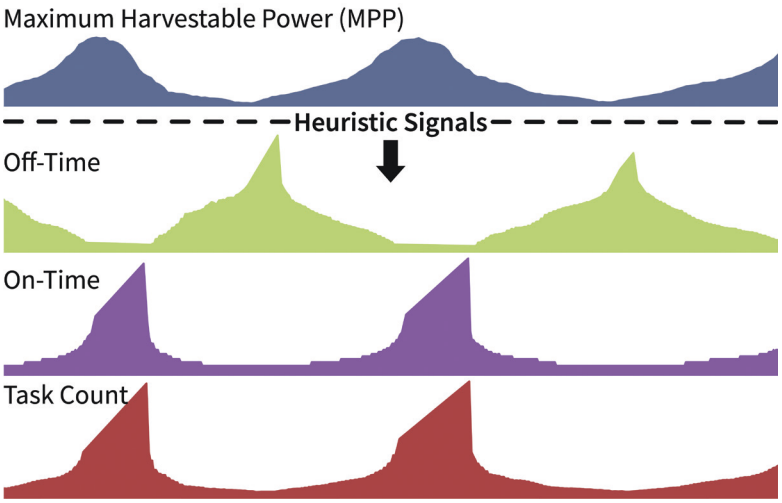

FIGURE 3. REHASH uses heuristics derived from program execution patterns to track the maximum power harvestable from an energy environment, providing a useful estimate of energy availability trends.

for *off-time*, which measures how long it takes to charge an empty capacitor – if short, energy is high. A similar adaptation signal can be *task-count* with more tasks completed, meaning more energy available. All these signals track MPP (Figure 3), are maintained through power failures by REHASH, and are very lightweight, requiring little or no extra circuitry and little to no computation to derive.

Heuristic Function: When to Adapt

In REHASH, a *heuristic* function is a developer-defined logic statement that takes an equation composed of measured signals and calculates a binary outcome, i.e., *adapt-up*, or *adapt-down*. The outcome of the heuristic function decides when to adapt tasks. As depicted in Figure 2, the REHASH heuristic is a function denoted by *f*. The inputs of the heuristic function are the adaptation signals, denoted by *s* in the figure. REHASH can accept any heuristic function that can use many or only one adaptation signal to output the adaptation decision, e.g., only *on-time* or *on-time* and the *task-count*, etc. REHASH also records past history, i.e., it holds the values of the adaptation signal in the previous active cycles before the most recent power failure. Integrating more history means taking the signal values of previous active cycles into account. The heuristic function can be any equation or relation expressed in a mathematical form (with the constraint that excessively complicated heuristics will take longer to compute and may become less interpretable). For example, a heuristic function can be created that compares the latest measured value of a single adaptation signal, e.g. the *task-count*, with the average value of the corresponding adaptation signal stored in the history. If the current value of the signal is less than the average, the application should *adapt-down*, since fewer tasks are completed, which likely means the energy environment is trending low. Otherwise, the application should *adapt-up* since more energy is available to execute more tasks.

Knobs: What to Adapt

In REHASH, *knobs* are application-specific variables controlling internal/external peripherals and sensors, compute, and communication that can be tuned to use more/less energy based on the outcome of a heuristic function. Battery-free energy-harvesting devices support various applications. A single *up/down* adaptation routine cannot be generalized for every application. The choice of the adaptation strategy, i.e., *what to adapt*, must be decided by the developer instead of the runtime. REHASH provides appropriate APIs to define these knobs and allows developers to write adaptation routines so that these *knobs* can be tuned to provide the best quality of service.

Fine-tuning Heuristics and Knobs

Too many choices and flexibility can lead to an increased tradeoff space. Programmers may become overwhelmed with the *knobs*, *heuristics*, and possible configurations. For example, when choosing between degrading the sample rate or the communication rate when energy is low, the developer might be wondering what the impact of a certain decision will be on performance. The developer has little insight into how well the program will perform in the wild, or even if the program as written can get things done under the most likely energy harvesting environment. We, therefore, develop a simulation tool, REHASH-explorer[1], with visual feedback that helps developers conceptualize the impact of design decisions on energy efficiency and application performance. It allows users to analyze different adaptation strategies, i.e., the *knobs*, with either REHASH's pre-defined or user-defined custom heuristics in various energy harvesting conditions and reports different statistics like i) total available energy, ii) total energy consumed, iii) the number of tasks executed, iv) the number of app completions, v) the number of power failures, and vi) average accuracy of inferences performed.

Bringing it all Together: How to Adapt

Finally, the application developers can take adaptation signals, the *heuristic* function that takes transformed adaptation signals to make an *adapt-up* or *down* decision, the *knobs* and variables they want to change, and then designate tasks adapted within the program. At this point, the programmer is merely making connections (*glue* code) between the heuristic adaptation approach and the tasks themselves as REHASH controls everything else.

EVALUATION: DOES HEURISTIC ADAPTATION REALLY WORK?

We evaluated REHASH intending to understand the benefits and drawbacks of adaptation in terms of performance (defined differently by each application), platform's flexibility and generality, overhead, and user experience. We explore heuristic adaptation with varied energy harvesting modalities and diverse applications: machine learning, activity recognition, and greenhouse monitoring, and find that the adaptive version of our ML app performs up to 46%

[1] https://adaptationprofiler.github.io/adaptation-profiler/

more classifications with only a 5% drop in accuracy; the activity recognition app provides 76% better coverage with only nominal down-sampling; and find that heuristic adaptation leads to higher throughput versus non-adaptive in all cases. While the full paper [1] presents a comprehensive evaluation of our system, here we highlight some general lessons that we learned from our evaluation.

Choice of adaptation signal matters: Both *on-time*- and *off-time*-based heuristics perform better than non-adaptive executions in all energy harvesting environments, meaning that they capture changes in energy harvesting conditions. However, *task-count* shows different behavior in each application. Since the neural network-based classifier is just a computational application and does not involve any real sensor or camera, *task-count* works better as the size of the task remains fixed, whereas activity recognition and greenhouse monitoring applications use sensor and radio modules involving I/O operations. A SEND task can take 1000x more energy than a COMPUTE task. The *task-count* signal treats both these tasks the same, triggering adaptation at the wrong time, and therefore is not a good signal for tracking MPP in all kinds of applications.

Adaptation is triggered at the right time: As stated earlier, the signals used by REHASH efficiently capture MPP. With the diverse set of applications that we use in our measurement study, where I/O and high energy radio operations are involved, heuristics may trigger adaptation at the wrong time when it is not needed. Therefore, we wanted to see when adaptation is triggered with REHASH. While *task-count*-based heuristic does not trigger adaptation at all desired points because of its inability to capture the effect of high energy I/O operations, we find that both *on-time* and *off-time*-based heuristics accurately trigger down-adaptation when power decreases, and up-adaptation when power increases.

Heuristics are lightweight and give better sensor coverage: Throughput does not always correlate to application quality. Ten samples, all gathered within one second, are not very useful, or worse, are redundant and a waste to gather. Often one sample is enough for every period of a few seconds, minutes, or even hours, depending on the application. We use sensor coverage as a measure of how spread out samples are. We look at the time between two app completions/classifications as a way to measure sensor coverage. This sensor coverage metric is important for motion event detection or continuous monitoring applications like those seen in infrastructure monitoring. We find that all heuristic-based adaptive executions of REHASH show a smaller time difference, indicating better sensor coverage than non-adaptive runs of the same applications.

The full paper [1] includes an in-depth evaluation of heuristic adaptation and offers more insights into how the ability to develop custom heuristics using the *signals* offered by REHASH makes application development flexible for programmers, and why REHASH's adaptation signals may not capture MPP if the frequency of the harvested energy signal is low.

SUMMARY

Intermittently powered energy-harvesting computers and sensing devices have great potential for revolutionizing sensor networks and the Internet of Things. This article has explored the challenges of developing robust energy-aware adaptation for batteryless sensors. We aimed to give developers the flexibility to account for the broad range of applications, hardware, and energy harvesting scenarios that could be encountered. We made heuristic functions from easily gathered signals, explored heuristic adaptation over various applications in multiple energy-harvesting scenarios, and found that more consistent and reliable data delivery can be realized by programmer-informed runtime adaptation to the energy environment. ■

Abu Bakar is a Ph.D. candidate in the Department of Computer Science at Northwestern University. His work focuses on making batteryless systems robust, and resilient to dynamic energy-harvesting conditions. He explores new hardware designs, builds efficient runtime systems, and develops tools to create functional and intelligent applications capable of real-time inference and self-adaptation in extreme energy harvesting conditions. http://abubakar.info/

Alexander Gregory Ross is a Bachelor's/Master's dual degree student at Northwestern University, studying Biomedical Engineering and Electrical Engineering. His recent work focuses on intermittent computing, machine learning, wearable devices, battery-free devices, and dynamic resource allocation.

Kasim Sinan Yildirim is an assistant professor at the Department of Information Engineering and Computer Science, University of Trento, Italy. He works on low-power and networked embedded systems, including intermittent computing, operating systems and runtimes, computer architectures, and low-power wireless protocols. http://sinanyil81.github.io/.

Josiah Hester is the Breed Chair of Design and an assistant professor of Computer Engineering at Northwestern University. He designs computer systems resilient to power failures, applied to health wearables and large-scale sensing for sustainability and conservation. He works toward a sustainable future for computing informed by his Native Hawaiian heritage. josiahhester.com

REFERENCES

[1] Abu Bakar, Alexander G. Ross, Kasim Sinan Yildirim, and Josiah Hester. 2021. REHASH: A flexible, developer focused, heuristic adaptation platform for intermittently powered computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5, 3, Article 87.

[2] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next century challenges: Mobile networking for "smart dust." *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99).*

[3] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. Time-sensitive intermittent computing meets legacy software. *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20).*

[4] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems.*

[5] Bashima Islam and Shahriar Nirjon. 2020. Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently powered systems. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies,* 4, 3, Article 82.

[6] Jane W.S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, Riccardo Bettati, and Jen-Yao Chung. 1994. Imprecise computations. *Proceedings of the IEEE 82*, 1.

[7] Marcelo Martins, Justin Cappos, and Rodrigo Fonseca. 2015. Selectively taming background android apps to improve battery lifetime. 2015 *USENIX Annual Technical Conference.*

[8] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. 2017. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services.*

[9] Abu Bakar and Josiah Hester. 2018. Making sense of intermittent energy harvesting. *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems.*