Competition: Work Smarter, not Harder: Towards a Sustainable IoT

Johannes Göpfert *
johannes.goepfert@tuhh.de
Hamburg University of Technology

Sayedsepehr Mosavat *
sayedsepehr.mosavat@uni-due.de
University of Duisburg-Essen
sayedsepehr.mosavat@hsnr.de
Niederrhein University of Applied Sciences

David Glomsda
david.glomsda@hsnr.de
Niederrhein University of Applied Sciences

Kamil Kaznowski *
kamil.kaznowski@tuhh.de
Hamburg University of Technology

Pedro José Marrón
pjmarron@uni-due.de
University of Duisburg-Essen

Bernd-Christian Renner
christian.renner@tuhh.de
Hamburg University of Technology

Matteo Zella
matteo.zella@hsnr.de
Niederrhein University of Applied Sciences

ABSTRACT

Batteryless devices offer remarkable opportunities for attaining the vision of a large-scale yet sustainable Internet of Things. However, their stringent energy budget also introduces challenges that must be addressed before such devices can be utilized effectively. Energy-efficient, intermittence-aware operation is one of such challenges. In this work, we present an approach to tackle this issue in the context of executing a proof-of-work algorithm on a low-power computation core. Our proposed techniques significantly improve the computation efficiency even in the face of frequent and unpredictable power loss events, a characteristic of batteryless, intermittently operating devices.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; • **Software and its engineering** → Software design tradeoffs.

KEYWORDS

batteryless operation, Hashcash, proof-of-work algorithm, SHA-1

1 INTRODUCTION

Attaining the vision of a planet-scale Internet-of-Things (IoT) requires the deployment of billions of ultra-low-power devices [3]. Relying on traditional batteries as the primary energy supply of these many devices not only imposes a tremendous maintenance overhead in the long run but also comes at a substantial environmental cost. Therefore, *batteryless* operation has been introduced by the research community in the past few years to tackle this challenge [1]. Batteryless devices store the energy they can harvest from their environment in small energy buffers and use that energy as soon as enough has accumulated. This limited energy supply leads to batteryless devices operating *intermittently*, facing several challenges that must be addressed carefully. One of these challenges is ensuring *forward progress*, meaning that after a power loss, the device ideally continues its operation where it was forced to stop due to energy depletion [5].

This work presents our approach to tackle the EWSN'24 Sustainability Competition, which aims at evaluating the performance of intermittent computing systems. Therefore, a proof-of-work algorithm, namely *Hashcash* [2] (See Algorithm 1), is executed on an ultra-low-power processing core, i.e., the TI MSP430FR5994 MCU. To emulate power intermittency and to provide a framework for the evaluation, the E-Cube testbed [7] is used. Goal of the challenge is to calculate as many solutions to the Hashcash problem as possible despite intermittency within a given time frame.

Algorithm 1 Hashcash

```
stem = MAKESTEM(n, challenge, rand)
counter ← 0
while True do
    message ← stem + STR(counter)
    if SHA1(message) has n leading zeros then
        return message
    end if
    counter ← counter + 1
end while
```

2 PERFORMANCE IMPROVEMENTS

When executing a proof-of-work algorithm like Hashcash on a resource-constrained device, optimizing the algorithm for computation efficiency is crucial since performing unnecessary operations consumes valuable time and energy. The cornerstone of Hashcash is the calculation of SHA-1 hashes. Thus, significant improvements can be achieved by optimizing this step. Algorithm 2 shows a simplified version of the SHA-1 algorithm [6]. The hash calculation starts with an array W consisting of 80 32-bit words, which is derived from Hashcash's challenge string.

Regarding SHA-1, we made the following general optimizations:
O1: Reimplementing SHA-1 in Assembly.
O2: Transforming ROTL$(30, c)$ into a ROTR$(2, c)$.

*The authors contributed equally to this work.

Algorithm 2 SHA-1 (Simplified)

$W[0:13] \leftarrow message$ ▷ Initialize SHA-1 message block
$W[14:15] \leftarrow \text{LENGTH}(message)$
for $i \leftarrow 16$ to 79 **do** ▷ 1. Message extension
  $W[i] \leftarrow \text{ROTL}(1, W[i-3] \oplus W[i-8] \oplus W[i-14] \oplus W[i-16])$
end for
$(a, b, c, d, e) \leftarrow IV$ ▷ Initial values
for $i \leftarrow 0$ to 79 **do** ▷ 2. Message processing
  $t \leftarrow \text{ROTL}(5, a) + e + f_i(b, c, d) + K_i + W[i]$
  $e \leftarrow d \leftarrow c \leftarrow b \leftarrow a \leftarrow t$
  $c \leftarrow \text{ROTL}(30, c)$
end for
return $(a, b, c, d, e) + IV$

O3: Rearranging operations to obtain a register-only computation.

O4: Loading SHA-1 related functions into RAM at startup.

Compared to the C implementation that requires 25000 CPU cycles, this optimized approach needs only 5300 cycles.

The SHA-1 message for Hashcash has the following fixed format, where the values of `n`, `resource` and `rand` are fixed for each challenge, while the value of *counter* must be determined by iteration.:

$$\underbrace{\texttt{1:<n>:240415:<resource>::<rand>:}}_{W[0:7] = stem}\ counter$$

We observe that, by the nature of Hashcash, the values of $W[0:7]$ are fixed throughout each challenge and incrementing the *counter* only changes the values in $W[8:12]$. Thus, instead of recomputing the entire message extension for each new *counter* value, we employ a precomputed look-up table to flip the relevant bits and then apply that to the previously expanded message. The new message extension is then calculated as follows, leading to an improvement of about 17 % for the calculation of one hash:

$$W_{\text{new}} = W_{\text{old}} \oplus \text{LUT}[\text{idx}(counter_{\text{new}}, counter_{\text{old}})]$$

Due to the fixed $W[0:7]$ it is furthermore possible to precompute the first eight iterations of the message processing loop. Avoiding these calculations for each new *counter* value leads to an improvement of roughly 10 % for calculating one hash.

3 ENSURING FORWARD PROGRESS

When a device suffers a power loss, the state of volatile memory is lost. Therefore, the current progress must be saved in non-volatile memory, i.e., the internal FRAM of the utilized MCU. We considered two approaches for ensuring forward progress: 1. Using the *Compute-Through-Power-Loss (CTPL)* [4] library; 2. employing a lightweight, checkpoint-based approach to store a minimal amount of data in the FRAM.

The CTPL library mainly aims to create backups of the RAM content and store them in the FRAM before a power loss. To utilize this library effectively, however, it is crucial for the device to reliably detect a power loss event moments before it happens. Nonetheless, due to the hardware setup of the evaluation testbed, such events that can happen in the context of the competition are independent of the device's energy consumption, making them challenging to predict beforehand. Since it is not possible for the device to accurately predict power failures, using a lightweight checkpointing scheme will lead to a more reliable and efficient solution compared to the CTPL library. In addition, as previously discussed, we move several functions used for the SHA-1 computations to the RAM during startup. Therefore, in the face of frequent, unpredictable power loss events, the CTPL library would incur a relatively high overhead for copying the RAM contents to the non-volatile FRAM. Avoiding the CTPL library allows us to use such periods to make further progress with the calculations of SHA-1 hashes.

Consequently, we opt for a checkpoint-based approach. Since storing a value in FRAM only comes with a small overhead, it is possible to make checkpoints frequently. For this, the overall task is split into small, independent parts, i.e., iterations of the SHA-1 hash calculations. We store the Hashcash *counter* in FRAM so that after a power loss, the device can again continue with the last SHA-1 iteration, during which the device ran out of power. After solving a challenge successfully, we store the number of solved challenges and the external FRAM position on which the last solution was stored on the non-volatile memory of the MCU. Storing these values is necessary to enable the device to continue working on the correct challenge after a power loss event and subsequently write the resulting solution in the correct area of the external FRAM holding the solutions. Since these two values must always be consistent and modified together atomically, we employ a ping-pong buffering scheme to prevent potential state corruption.

4 CONCLUSION

The presented work proposes several optimizations to tackle the challenge of efficiently performing energy-intensive computations despite unpredictable and frequent power losses. To achieve this goal, we apply low-level modifications to the underlying PoW-based task to improve the overall performance of the firmware. Moreover, we employ a lightweight checkpointing scheme to ensure efficient forward progress under unfavorable energy availability that can lead to intermittent operation. The introduced approaches have the potential to inspire more efficient, intermittence-aware solutions suited for batteryless devices, ultimately facilitating the realization of a sustainable, large-scale Internet of Things.

ACKNOWLEDGMENTS

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the "DI2T" project (project number 492151098), as well as by the Niederrhein University of Applied Sciences as part of the "ABORA" project.

REFERENCES

[1] Saad Ahmed et al. 2024. The Internet of Batteryless Things. *Commun. ACM* 67, 3 (2024), 64–73.
[2] Adam Back. 2002. Hashcash - A Denial of Service Counter-Measure. http://www.hashcash.org/papers/hashcash.pdf Accessed on September 20, 2024.
[3] Albert Cohen et al. 2018. Inter-disciplinary research challenges in computer systems for the 2020s. *National Science Foundation, USA, Tech. Rep* (2018).
[4] Texas Instruments. 2015. Intelligent System State Restoration after Power Failure with Compute Through Power Loss Utility. https://www.ti.com/tool/TIDM-FRAM-CTPL Accessed on September 20, 2024.
[5] Brandon Lucia et al. 2017. Intermittent computing: Challenges and opportunities. In *SNAPL 2017*.
[6] National Institute of Standards and Technology (US). 2015. *Secure hash standard*. Technical Report. Washington, D.C.
[7] Markus Schuß and Carlo Alberto Boano. 2024. E-Cube: Towards a First Benchmarking Facility for Battery-Free Systems. In *Proceedings of the 4th International Conference on Information Technology for Social Good (GoodIT)*. 399–403.