Leveraging Micro Energy Sources in Energy Harvesting Wireless Sensor Networks

Sven Pullwitt, Patrick Tudyka, Lars Wolf
Institute of Operating Systems and Computer Networks
TU Braunschweig
Braunschweig, Germany
{pullwitt, tudyka, wolf}@ibr.cs.tu-bs.de

***Abstract*— Effectively utilizing very small energy sources, like ambient thermal gradients, in a Wireless Sensor Network (WSN) offers great potential for novel applications. Further, without over provisioning the energy source or storage of an Energy Harvesting-WSN, it can be operated more ecologic.**

We reduce the overhead associated with peripheral initialization by dynamically adapting the boot process of sensor nodes operating in a burst style operation from micro energy sources. For the evaluation, we implement a framework that allows reproduction of real-world conditions for Thermoelectric Generator energy harvesters. Based on real-world data our study shows an improvement of $27\,\%$ in the number of tasks executed with only half the energy required for certain tasks. Finally, we discuss the factors influencing the effectiveness of our dynamic boot approach w.r.t application scenario.

***Index Terms*—energy harvesting, wireless sensor network, thermal electric generator**

I. Introduction

The power supply for low power devices is an important problem for WSNs operating in harsh outdoor environments. Since batteries would limit network lifetime and complicate maintenance, energy harvesting provides a promising approach. Potentially allowing indefinite, maintenance-free operation, using ambient energy sources in WSNs comes with great opportunities but also presents a unique set of challenges. One of the main problems when using energy harvesting is the generally hard to predict amount and temporal characteristic of power available for an application [5]. This issue is most commonly addressed by over provisioning [4]: Dimensioning the energy source so that the average energy production far exceeds the expected requirement of the application and using an energy storage that is big enough to bridge the longest periods of power outage expected.

However, keeping all efforts to conserve resources in WSNs in mind, this approach is wasteful. Moreover, given the wide range of different usage scenarios, a sufficiently large ambient energy source might not be present in many cases. To allow the use of ambient energy sources, such as thermal gradients and vibrations, many of which usually deliver a low amount of power compared to the application requirements, an intermittent operation of the nodes is required.

As a result of extensive research in this area, for many aspects of an Energy Harvesting (EH) system, specific, advanced protocols and solutions are readily available in special Operating Systems (OSs), such as RIOT-OS [2]. However, using an operating system inevitably comes with some overhead, which given the limited and complex power supply in EH-WSNs, has to be seen critically. The boot process represents a significant portion of this overhead, as devices and peripherals need to be initialized. In battery-powered systems that do not use energy harvesting, a given amount of energy is available at the time of deployment, which does not get replenished over the lifetime. In these systems, the overhead resulting from the boot process can be minimized by sleeping in between tasks and avoiding the boot process before the next task execution. However, in energy EH-WSNs which are designed to allow usage of very small energy sources, reboots in between tasks are to be expected [1]. While many OSs provide a modular design, allowing selection of components during compile time, in an intermittently powered system not all tasks might be executed before the next loss of energy. Thus, not all components and peripherals might be needed at each boot, as different tasks often utilize different subsets of peripherals. Dynamic adaption of the components of an OS during boot can reduce the overhead in such systems.

The contribution of this paper is twofold. First, we analyze the possibilities of conserving energy during boot by dynamically adapting the peripheral initialization in the widespread WSN-OS RIOT. Second, we present an evaluation based on the emulation of real-world data and outline factors that influence whether the approach of a dynamic boot can be an asset in conserving energy in EH-WSNs.

II. Related Work

Many systems exist that utilize micro energy sources such as thermal gradients, e.g., between the air and rock [18], concrete [14], soil [6], [9], [11], [15], [19] or even the human body [13] to power WSN nodes. All these have to face the challenge of efficiently utilizing an energy source that is comparatively small in magnitude, as well as difficult to predict.

For testing and evaluation purposes, being able to recreate real-world conditions repeatably in a lab is useful. Hence, test-beds such as in [17] are widely used in this research area. The authors present a test-bed that allows the reproduction of both thermal gradients and visible light conditions in a lab setup. Additionally, an electronic load emulation allows the emulation of different load scenarios.

A framework to emulate energy harvesting applications is presented in [8]. It is capable of the emulation of visible light, Radio-Frequency (RF) and kinetic energy using data collected from the real world. While the approach applies to the evaluation of the dynamic boot approach presented here, the framework lacks the capability of reproducing thermal gradients.

Another framework for recording energy traces from real-world harvesters and replaying them to nodes in a repeatable fashion is in [7]. Instead of emulating the physical conditions at the harvester in a lab, the electrical output is measured, logged and then reproduced.Yet, during the recording the energy is not used by a sensor node, but rather by a dummy load. Hence, dedicated data recording is required which impedes using data from real-world deployments of EH-WSNs.

While sleeping in between task executions will reduce the number of boots and, thus, the overhead resulting from the boot process, this approach might not always be viable in real-world deployments. In the deployments in [1], in most cases, the energy was not sufficient to sustain sleep between tasks. For energy sources, such as Thermoelectric Generators (TEGs) or piezo-electronic harvesters, the sleep current could amount to a significant portion of the harvested energy.

[12] outlines that the comparatively large start up time of crystal oscillators presents a source of large overhead during start up of wireless Internet of Things (IoT) devices. Albeit using a different approach, the authors tackle a similar problem to the one presented in our work, the overhead resulting from state transitions when using duty cycling to conserve energy. A hardware approach for minimizing the start-up energy of a crystal oscillator is presented and evaluated.

An approach that allows for the execution of long-running tasks on platforms that suffer from frequent loss of power by using checkpoints is presented in [16]. Yet, the creation of checkpoints presents an overhead. Hence, WSNs that only perform short-running tasks, such as collecting sensor data, can benefit from a solution that minimizes overhead. Further, skipping the initialization of peripherals that are not needed during a burst could offer additional energy savings.

While checkpoints can restore the program state in case of a power loss, peripherals may need to be handled separately, as restoring the state prior to the power loss might require additional steps. In [3] a kernel is presented addressing this challenge. Still, the problem of the significant overhead of peripheral initialization remains. The authors state that the initialization of the peripherals accounts for a large portion of the time spent during boot.

In order to effectively utilize small energy sources in WSNs, checkpointing systems such as [3], [10], [16] will not cover all use cases, as they fail to address the problem of the significant overhead during boot.

III. Dynamic booting

Using small temperature deltas as the power source of a sensor node can result in very limited energy, even sleeping in between the execution of tasks may not be possible. Using a small energy storage, which very small energy sources can fill, often results in depletion of the energy after the execution of very few tasks. In such systems, it is possible that a different task needs to be performed at each boot. To keep track of the task schedule but turn the main processor on and off, a low power Microcontroller Unit (MCU) can be used which is kept active whenever possible, albeit with a lower operating voltage and possibly a reduced sleep current.

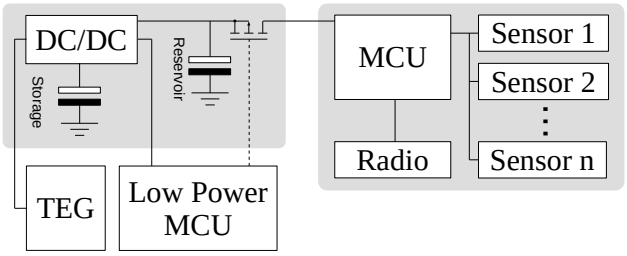

Fig. 1. Typical energy harvesting system for burst style operation from low-power sources such as TEGs. Energy conversion, storage and switching can be integrated into a single device such as the LTC3109

A. Background: peripheral initialization approaches

A straightforward approach to peripheral initialization is to initialize all peripherals during boot. This has the risk of higher overhead as there might be peripherals that are initialized but are not used before the depletion of the available energy. To avoid this, peripherals could be initialized the first time they are used. However, it can be beneficial to perform the initialization during boot in certain scenarios. An example of such a case can be an Analog to Digital Converter (ADC), as these peripherals can influence the choice of clock source for the platform. For instance, a low-power RC oscillator might be used as a clock source for the system clock whenever the ADC is not used. However, when the ADC is used, which achieves greater accuracy with a more accurate clock source, a crystal oscillator is used, which then can also be used as the system clock source. Further, when initializing on first use, the first access to the peripheral might take longer. This might be unwanted for applications that rely on the timing. In addition to this, because of the latency on first use, additional care has to be taken so that all running components are in a low-power state while waiting for the initialization to finish. Indeed, ensuring this is more straightforward during boot by carefully designing the initialization order, while doing this during run-time might be significantly more complex. Initializing peripherals each time they are used and de-initializing them after use is a third approach, which solves the problem of varying access times. However, in short-lived applications, the overhead originating from the frequent initialization can quickly become a significant contributor to the overall energy consumption.

B. Use case TEG harvester

A typical harvester utilizing a main and a low-power MCU is shown in Figure 1. Energy is collected from the TEG and

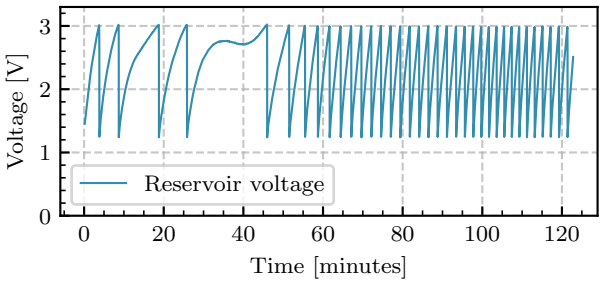

Fig. 2. Voltage level in the reservoir capacitor during burst style operation of the MCU in a real-world deployment

stored in the reservoir capacitor. Once the reservoir is filled sufficiently, the main MCU can be turned on, and a task can be executed. Whenever the reservoir is filled, and energy is collected from the TEG, this excess energy can be stored in a storage capacitor. Handling of the voltage conversion and management of the storage and reservoir capacitors can be done by specialized chips, such as the widely used LTC3109 by Analog Devices. In order to use small energy sources, the reservoir should be kept as small as possible, i.e. sufficient for a single task execution, as this reduces the time required to charge before a task can be executed. In an ideal world, the storage capacitor would be infinitely large, as this avoids wasting energy in case both the reservoir and storage capacitor are full. However, the storage capacitor in this system is used to fill the reservoir without additional voltage conversion whenever there is not enough energy available from the TEG. For this to be possible, the voltage level in the storage capacitor needs to be greater than in the reservoir. To allow this, the ratio between the storage and the reservoir capacitor has to be selected based on the amount of energy expected in the deployment.

The voltage level in the reservoir capacitor in a real-world deployment is shown in Figure 2. Once the capacitor is charged to a defined voltage, in this case 3 V, the main MCU is activated and executes a task. After the task execution, the main MCU gives a signal to the low-power MCU and is turned off, and the reservoir capacitor is recharged. The next task is then executed after a minimum wait time, which allows setting a maximum duty cycle, and once the reservoir is recharged. Turning the main MCU off in between tasks executions allows to save energy, as no sleep current is present. As the recharge time is large compared to the task execution time, sleeping in between tasks would consume a considerable share of the energy available, even with very low power consumption during sleep.

The main MCU initializes the peripherals during boot, such as sensors and the radio. However, not all peripherals might be needed depending on the task to be executed. During the execution of the next task, after the next boot, a different set of peripherals might be needed, and others might not be required.

C. Proposed dynamic boot approach

Here we propose a dynamic boot approach that extends RIOT [2], a popular operating system for WSNs, so that only those modules required for the current task are initialized. During boot, the operating system core is initialized before the initialization of peripherals and modules. This initialization is not modified in our dynamic boot approach. It will take a set amount of time and therefore a set amount of energy, depending on the MCU in use. The energy required for this core initialization, in the following E_{core}, is independent of the modules. After this, in case of the standard, unmodified boot approach, the modules and peripherals are initialized. In our dynamic boot approach, a check is performed for each module prior to the initialization. If the module is not needed for the task, its initialization is skipped. For this approach, knowledge of the required modules, and therefore the current task, are required during the boot. Utilizing the low-power MCU, this information can be delivered. With a static schema of tasks, defined during compile time, the modules required for each task are known.

Using the setup in Figure 1, the low-power MCU can inform the main MCU about the task to be executed, i.e., by using IO-Pins, which can be read with minimal overhead. An alternative to using a low-power MCU could be using a simple real-time clock (RTC) with a calendar to facilitate task execution depending on the time and date. Even without an additional device for advanced scheduling, this approach can be used. Each time the reservoir capacitor is filled, a burst can be executed, and the last executed task could be stored in non-volatile memory. For ease of use during the implementation the modules to be skipped are given in a blacklist rather than the modules required.

Depending on the nature of the peripheral, skipping the initialization might save a considerable amount of execution time and therefore energy. Of course, not all peripherals take equal amounts of time during initialization. Given a set of N modules included during compile time, each of the modules $n \in N$ will require a set amount of energy (E_n) during initialization.

With a set of M different tasks executed by the sensor node, each task $t \in M$ will require a subset of the N modules. In the standard, static boot approach, which initializes all modules regardless of whether or not they are required, the energy used during boot can be given as:

$$E_{\text{boot,static}} = E_{\text{core}} + \sum_{n=0}^{N} E_n$$

Now, since there are $|M|$ different tasks, the energy required during boot for executing all these tasks with the sensor node being turned off in between task execution can be given as:

$$E_{\text{tasks,static}} = |M| \cdot \left(E_{\text{core}} + \sum_{n=0}^{N} E_n \right)$$

For the dynamic boot approach the energy required during boot changes depending on the current task ($t \in M$). Let $a_{n,t} = 1$ if module n is required for executing task t and $a_{n,t} = 0$ otherwise. There is some energy required for checking whether the module is required for the given task

(E_{check}) based on the static schema, independent of $a_{n,t}$. In addition to this, the task needs to be known before the initialization, which takes energy E_{schedule}, depending on the method of determining the next task. Therefore, the energy required during the dynamic boot process for task t is:

$$E_{\text{boot,dynamic},t} = E_{\text{core}} + E_{\text{schedule}} + \sum_{n=0}^{N} E_{\text{check}} + E_n \cdot a_{n,t}$$

With this, the energy required for executing all $t \in M$ tasks using the dynamic boot approach is:

$$E_{\text{tasks,dynamic}} = \sum_{t=0}^{M} \left(E_{\text{core}} + E_{\text{schedule}} + \sum_{n=0}^{N} E_{\text{check}} + E_n \cdot a_{n,t} \right)$$

The first part of the overhead, E_{schedule}, introduced by the dynamic boot approach results from obtaining the next task during boot, in order to know the required peripherals. This overhead will vary depending on the method of scheduling. If, for instance, an RTC is used for determining the next task, a bus access is required and causes overhead during boot. In our setup, the low-power MCU handles the scheduling. Therefore, during the boot of the main MCU, only minimal overhead is introduced by reading the IO-Pins set by the low-power MCU.

The second part of the overhead, E_{check}, results from checking whether the peripheral is required for the current task. Checking for each peripheral whether or not it is in the blacklist can be achieved with very little overhead, as this is a simple list lookup. The size of the list is limited by the number of modules and peripherals used and will, in many cases, be significantly smaller when only energy-intensive initializations are skipped.

It is possible that there are modules for which E_{check} is greater than E_n. If a module requires only a few simple initializations, such as configuring IO-Pins, the lookup in the blacklist might require more energy. In these cases, there might be a task t with $(a_{0,t}...a_{n,t})$ such that $E_{\text{boot,dynamic},t} > E_{\text{boot,static}}$. Therefore, using the dynamic boot will result in a higher energy consumption on this task. However, a different task will likely require different modules, possibly resulting in significantly reduced energy consumption during boot compared to the static boot. The higher energy consumption in one task would then be compensated for by the energy-saving in another, allowing an overall lower energy consumption.

Thus, the dynamic boot approach is expected to be most effective if the application matches any of two cases. First, if a module exists with $E_n > m \cdot (E_{\text{schedule}} + nE_{\text{check}})$, which is not required by all tasks, using the dynamic boot approach will save energy based on this single module alone. Second, consider an application with many modules, most of which require E_n that is at least somewhat larger than E_{check}. If most tasks only use a small subset of the modules, the dynamic boot approach can reduce the energy required during boot.

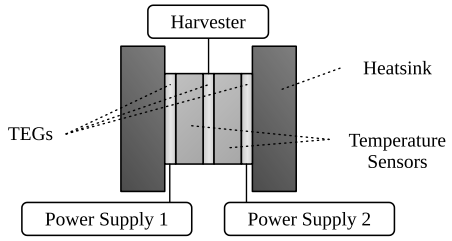

Fig. 3. Setup for repeatably emulating a real-world TEG harvester

IV. Evaluation Framework

The evaluation aims at comparing the proposed dynamic boot approach with the standard static boot approach using a real harvester with a sensor node in a realistic real-world scenario. Measuring the system in a real-world deployment has a significant limitation for comparing the dynamic boot approach with a static boot. Since the temperature changes from day to day, the temperature difference presented at the TEG will change as well, and measurements made on successive days will produce different amounts of energy. Using multiple harvesters to compare the two approaches simultaneously is possible, but hardware tolerances and differences in the placement of the harvesters will reduce the accuracy of the acquired results. To overcome these limitations, a dedicated lab setup that emulates previously recorded real-world conditions in a lab setup is beneficial for this evaluation. As the main point of the evaluation is the comparison of the two approaches in a realistic scenario, the emulation should be able to reproduce the conditions with minimal deviation in between runs.

A. Temperature gradient emulation

Since a TEG harvesting setup converts a temperature gradient into electrical energy, see Section III, this energy's amount and temporal characteristic is the determining factor for the performance and lifetime of the sensor node connected to the harvester. Hence, a possible approach for the emulation of the real-world conditions is to reproduce the voltage and current produced by the TEG in the real-world. An essential requirement is the availability of this data from a real-world deployment, which requires current and voltage measurements at the input of the harvester. This measurement can be challenging, given that losses, i.e. due to shut resistors used for current measurement, at the output of the peltier element should be minimized. Additionally, emulating the real-world conditions based on the voltage and current output, which changes depending on the TEG and harvester in use, limits the flexibility of the setup when it comes to comparing different harvesting hardware. To overcome these limitations, the temperature difference can be emulated instead. This approach allows for changes in the harvesting hardware, which might be required for evaluating different setups. Further, measuring the temperature at the TEG is minimally invasive to the real-world harvester compared to measuring the voltage and current.

To allow realistic, reliable results, an emulator is designed to reproduce temperature differences observed at a real-world TEG harvester in the lab. The components of the emulator are

presented in Figure 3. A harvester, identical to the ones used in the real world, is connected to a TEG of type PKE 127 A 0020. In a real-world usage scenario, the temperature difference (ΔT) at this TEG would be collected from the environment. The real-world data used in this emulation was recorded using a harvester with one TEG thermally coupled to the soil and the other to the ambient air. In the emulator setup, two temperature sensors, coupled with two additional, identical TEGs are placed at each side of the TEG used by the harvester. The sensors used are TSic-506F sensors with an accuracy of $0.1\,\mathrm{K}$. The two additional TEGs are connected to remotely controllable power supplies, which are used in combination with temperature sensors to allow regulation of the temperatures at each side of the harvesting TEG using a proportional–integral–derivative (PID) controller. Using this setup, data from a real-world harvester fitted with temperature sensors in a similar setup can be recreated in a lab. Thus, a realistic and repeatable ΔT can be presented to the TEG harvester.

B. Performance data recording

The emulator can accept arbitrary temperature setpoints to be emulated at both sides of the harvesting TEG. Hence, it is possible to emulate various different scenarios like a constant ΔT of a given magnitude and polarity over an extended period or the reproduction of temperatures observed in a real-world harvester. During the emulation, all data relevant to the performance of both the emulator and the harvester and the sensor node are recorded. The data collected from the emulator includes the setpoints and the actual temperatures at both sides of the harvesting TEG, as well as the current set at the power supply. The voltages at the reservoir and storage capacitor are recorded at the harvester. Additionally, the supply voltage of the low power MCU and the current of the main MCU are logged. For the sensor node, the status of the main MCU, the execution and duration of tasks and the transmitted wireless messages are logged. This comprehensive data set is subsequently used to evaluate both the emulator and the dynamic boot approach.

V. Evaluation

A TEG harvester, as presented in Section III is now used to evaluate the presented approach. This shows the energy saving achieved by the dynamic adaptation of the boot process to reduce the time required for booting. With the framework described in Section IV, the harvester is used in a lab with conditions representing a real-world deployment. Before analyzing the effect of the dynamic boot approach on the energy consumption, the performance of the ΔT emulation during the two test runs is evaluated.

Figure 4 shows temperatures recorded by a real-world TEG harvester during a time span of $24\,\mathrm{h}$. During the $24\,\mathrm{h}$ span, which corresponds to the time from midnight to midnight the following day in the real-world data, the temperatures at both sides of the harvester vary with the ambient temperature. This period is emulated twice for the evaluation, and the

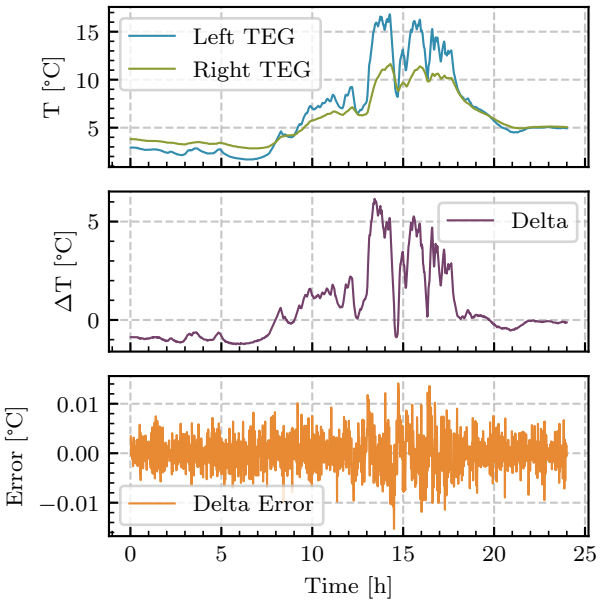

Fig. 4. Temperatures measured by a real-world TEG harvester (top) and resulting temperature delta at the TEG (middle). Error in the temperature delta, which was reproduced in the lab (bottom)

harvester is used to convert the energy and allow the sensor node to perform a number of tasks. The sensor node performs the same tasks with the same application code, once with the standard, static boot approach and one using the newly proposed dynamic boot. The $24\,\mathrm{h}$ period of real-world data, which is presented in Figure 4 at the top, contains spans of low and high ΔT, as well as slow and fast changes in the temperature gradient.

This $24\,\mathrm{h}$ period is emulated twice in real-time for the evaluation in two successive runs. The setup during both runs is identical, the only difference being the boot process in the operating system on the main MCU on the sensor node. During the two evaluation runs, the data on the performance of the emulator, the harvester and the sensor node is collected by the framework, as described in Section IV-B. In the following, the performance of the emulation of the real-world data is analyzed regarding accuracy and repeatability before the performance of the two boot approaches during the runs is presented.

A. Emulation accuracy

One side of the real-world harvester is thermally coupled to the ambient air, whereas the other side is coupled to the soil. The recorded temperatures are fed to the emulator for the evaluation, which recreates them in the lab. In the emulation setup, the left TEG is used to emulate the ambient air side, whereas the right TEG emulates the soil temperature side of the real-world data. Keep in mind that these temperatures are not simply the air and soil temperatures observed in the real world but rather the temperatures measured directly at the TEG in the real-world deployment. Figure 4 shows the performance of the emulator w.r.t the real-world measurement data during the first of the two emulation runs. In the top in

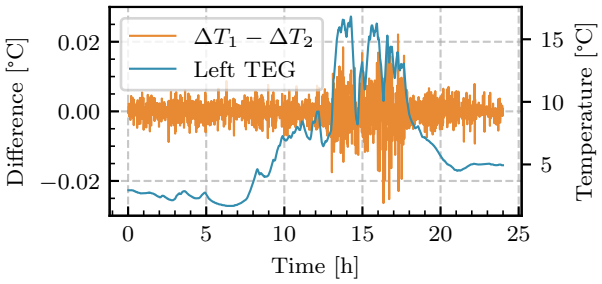

Fig. 5. Difference in the emulated ΔT between two sequential runs of the same real-world data

Figure 4, the real-world measurement data for both sides of the harvesting element, used as the setpoint for the emulator is shown. The resulting temperature delta at the harvesting peltier element in the real-world harvester, which can be seen in Figure 4 in the middle, varies from approx. $-1.2\,°C$ to $6.2\,°C$. A change in the polarity of the temperature difference can also be observed. In the night hours from midnight to approx.$8\,h$ and from approx. $20\,h$ to midnight the following day, the ΔT is negative, meaning the air side is colder than the soil. While repeatability is more relevant than accuracy to the evaluation presented here, an accurate emulation will provide realistic conditions. The achieved error in the emulated temperature delta, that is, the difference in the ΔT of the lab setup and the ΔT measured in the real world, is presented in Figure 4 at the bottom. As expected from a PID controller, the error tends to be higher whenever there are more prominent changes in the temperature. Overall, the accuracy is below $\pm 0.01\,°C$ for most of the emulation run, with some spikes above and below this value to $-0.0174\,°C$ and $0.0168\,°C$. Due to the fact that the errors produced during the run are fairly symmetric around zero, the overall average error of only $2 \times 10^{-4}\,°C$ is comparatively low.

B. Emulation repeatability

While the high accuracy in reproducing the real-world conditions achieved with the emulation setup is helpful in the evaluation, the most important point in evaluating the dynamic boot approach is the repeatability of the emulation. As the goal of the emulation is to evaluate the dynamic and static boot approaches in the same conditions, the framework needs to be able to reproduce the ΔT in sequential runs with as little deviation between the runs as possible. The performance of the emulator with respect to repeatability is presented in Figure 5 as the difference in the emulated ΔT between the two emulation runs, with the static and dynamic boot approach. For reference and to outline the times with fast changes in the emulated temperature, the absolute temperature of the left TEG is shown as well. As was the case with the accuracy of a single run, presented in Figure 4, a quicker change in the emulated temperature will increase the error. The minimum and the maximum difference in ΔT between the runs was $-0.0264\,°C$ and $0.0221\,°C$, respectively. Again, due to the symmetric nature of the difference, the overall difference between the runs is only $1 \times 10^{-5}\,°C$.

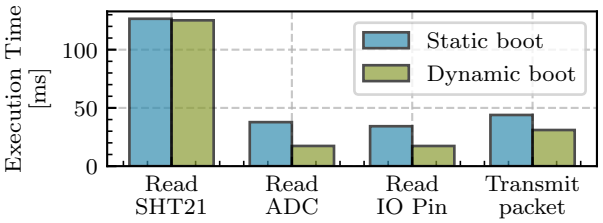

Fig. 6. Average time used for completion of each task during the evaluation

C. Application operation principle

During the evaluation, the main MCU performs four different tasks representing typical tasks for a sensor node. The four tasks consist of three measurements which are read and stored in Ferroelectric Random Access Memory (FRAM), and one transmission task, during which all measured values are read from the storage and transmitted wirelessly. The measurements are taken using an SHT21 temperature sensor, an ADC and a simple IO Pin, to represent different complexities and usage of peripherals. The low-power MCU turns on the main MCU and instructs it to execute a task, whenever sufficient energy is available. In between the measurement tasks, the MCU is turned off for one second to allow the reservoir to be filled. After the transmission task, a longer off period of $10\,s$ limits the sample rate, as it is typical for measuring values with low dynamics in WSNs.

In the evaluation, all tasks are executed sequentially. The MCU is booted once during each burst operation and executes a single task. Whenever too little energy is available to recharge the reservoir in the wait time of $1\,s$, or $10\,s$, respectively, the next task is executed as soon as enough energy is available. After completing the task, signalled by the main MCU using an IO-Pin, the main MCU is turned off by the low-power MCU, therefore conserving any remaining energy in the reservoir.

D. Task execution time

For each task, the average execution time during the evaluation is shorter using the dynamic boot approach, as can be seen in Figure 6. Therefore, the time savings achieved are greater than the overhead introduced from reading the task to be executed from the low-power MCU and checking the modules to be skipped in the initialization.

Overall, measuring a value using the SHT21 temperature and humidity sensor takes the most time of all the tasks. The SHT21 measurement task is only marginally faster using the dynamic approach. Therefore, skipping the initialization of the IO-Pin, the ADC, and the radio is only a small improvement in the execution time. Looking at the results of the ADC measurement, the difference between the dynamic and static approach is more pronounced. Performing the ADC task takes approximately $34\,ms$ without and $17\,ms$ with the dynamic boot, which is an improvement of $100\,\%$. The same applies to reading the IO-Pin, it takes a very similar amount of time, and the improvement from using the dynamic boot approach is almost identical. When performing the transmission task,

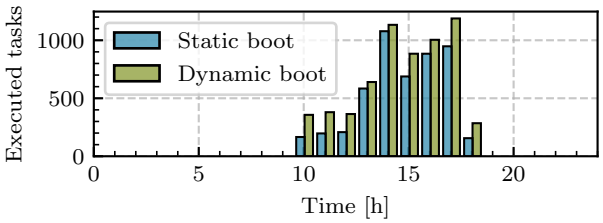

Fig. 7. Number of tasks executed per hour during the evaluation duration of 24 hours

the total time required is larger than for the ADC and IO-Pin measurement. Lowering the execution time of the task from 44 ms to 31 ms, the dynamic boot offers an improvement of 70 % as a result of not initializing the unused peripherals. In general, the improvement achieved by the dynamic boot approach varies greatly depending on the executed task. While the task of reading a value from the SHT21 sees only little improvement, the other tasks show a large reduction in execution time. As described in Section III, this stems from the fact that the module for the sensor has a very high overhead compared to the other modules. Since it is used for a single task only, all other tasks can benefit from skipping this module. For the task requiring the SHT21, only a little time can be saved from skipping other modules during boot.

E. Number of tasks executed

Figure 7 shows the number of tasks executed per hour by the main MCU during the emulation. First, it can be seen that the temperature gradient during the nighttime was not large enough to generate enough energy to execute any tasks. With greater ΔT, more energy is available to the harvester. The harvester can use both polarities of the ΔT. Therefore it can use temperature differences independent of which side of the TEG is the warmer one. However, the harvester requires the absolute value of the ΔT to be greater than approx. 1 °C before being able to produce electrical energy. Consequently, in the time span used in the evaluation, the MCU operates during the day hours, not enough energy is harvested while at night. While operation during the night hours is possible in general and observed in the real-world harvester, in the 24 h selected for the evaluation, the ΔT was not large enough.

Using the static boot, the main MCU was able to execute a total of 4909 tasks during the 24 h run. Over the same time span, in the run with the dynamic boot approach, 6235 tasks have been executed, therefore in the evaluation scenario, the improvement was 27 %. Intuitively, the lower ΔT in the morning hours results in fewer tasks than the afternoon hours with greater ΔT. Looking at the difference in the number of executed tasks between the two boot approaches, it is evident that the improvement in the morning hours when using the dynamic boot approach is much more significant than in the afternoon hours. This concludes that the dynamic boot approach is more effective in situations with lower ΔT and, therefore, lower power. This effect, however, is not a result of lower energy savings at higher ΔT, but rather an artefact seen when using the number of tasks executed as a metric for

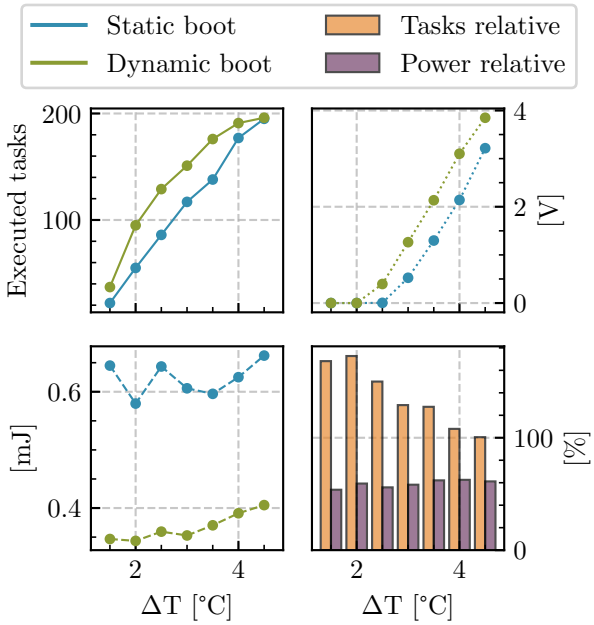

Fig. 8. Results of the test scenario with different values for ΔT. Absolute number of executed tasks per ΔT (top left); Average energy consumed per task (bottom left); Average voltage of the storage capacitor (top right); Relative number of tasks executed and average energy consumed per task between dynamic and static boot (bottom right)

a more efficient boot process. Since the application is limited in the maximum duty cycle, the time between execution of the tasks is determined by two factors: The minimum wait time of 1 s for the measurement tasks and 10 s for the transmission tasks and the time required to recharge the reservoir capacitor, which depends on the ΔT at the TEG. Therefore, for growing ΔT, the maximum duty cycle becomes the limiting factor for the number of tasks executed instead of the available energy. Hence, the power saving achieved with the dynamic boot will impact the number of tasks less for higher ΔT.

F. Artificial conditions scenario

To further evaluate this observation, a different test scenario is used. Instead of real-world temperature data, the emulator generates a ΔT from 0 °C to 4.5 °C at the TEG for equal periods of 10 min. The two boot approaches are again used in sequential runs. All capacitors are drained after every ΔT step. The results of these runs are presented in Figure 8. The number of tasks executed during the test periods for each ΔT step is shown in the top left. Similar to the results of the real-world runs, the dynamic boot outperforms the static boot by a factor of over 1.5 for small ΔT, while the difference is smaller for larger ΔT. For ΔT =4.5 °C the dynamic boot approach results in the same number of tasks executed per time interval. Therefore, the limiting factor is the maximum duty cycle of the application.

From the average energy used per task, shown in Figure 8 in the bottom left, it can be seen that, although the dynamic boot approach will not result in more tasks being executed for ΔT =4.5 °C, the difference in the energy consumption between the two approaches stays roughly constant. Therefore,

the boot approach does not get less effective with higher amounts of energy available. Applications with a higher duty cycle or higher energy requirement will benefit from this approach even at higher ΔT. Another trend that can be seen from the average energy used per task is that for higher ΔT the energy used per task increases slightly, even though the tasks are the same. This effect is a result of different voltage levels in the reservoir capacitor. When there is little energy available, and the subsequent task execution is due before the reservoir capacitor is charged sufficiently, the task will be executed as soon as the harvester signals a charged reservoir. This signal is given slightly below the nominal output voltage of $3.3\,\mathrm{V}$ at around $3.0\,\mathrm{V}$. Because the main MCU is perfectly capable of operating at voltages as low as $1.8\,\mathrm{V}$, this behaviour is not a problem. Therefore, whenever enough energy is available for the reservoir to be filled to the nominal voltage, the energy consumption will be slightly higher as the current consumption does not change with a change in the voltage.

The average voltage in the storage capacitor in Figure 8 in the top right indicates the amount of excess energy available for each ΔT step. As the storage capacitor is only charged whenever the reservoir is filled to the nominal voltage, it can be seen that with increasing ΔT the application does not utilize all energy available.

In the bottom right in Figure 8 the performance of the dynamic boot approach w.r.t power consumption and tasks executed is shown relative to the static approach. It can be seen that the energy consumption is fairly constant. The dynamic boot requires an average of just over $50\,\%$ of the energy required in the static boot per task. In terms of tasks executed, the dynamic boot approach allows executing more than $150\,\%$ of the tasks for low ΔT. This advantage falls with higher ΔT, until it reaches $100\,\%$ at a ΔT of $4.5\,^{\circ}\mathrm{C}$.

VI. Conclusion

Effectively using very small energy sources, like ambient thermal gradients, enables a variety of novel WSN-applications. Furthermore, it makes such WSNs both more economical and ecologic by removing the need for over-provisioning of the harvester and energy storage. A very limited energy supply combined with minimal storage capacity results in an intermittent, burst operation. Sleeping in between tasks might not be possible in many applications due to a lack of energy. Therefore, the boot process of a sensor node becomes an important factor in the energy efficiency of the node and the overall network. By dynamically adapting the boot process depending on the task that needs to be executed, we minimize the overhead resulting from initializing unused peripherals. To evaluate the performance of approaches for TEG energy harvesting in WSNs, we implemented a setup for emulating realistic conditions from real-world measurements both accurately and repeatably. In our evaluation, adapting the boot process can improve the execution time up to $100\,\%$ depending on the task and the peripherals used. Our example application executed $27\,\%$ more tasks using the dynamic boot, with a potential for an even greater improvement for different applications with a higher duty cycle or power consumption. Finally, we have outlined the factors influencing the effectiveness of using the dynamic boot approach in a series of artificially generated temperature scenarios.

References

[1] M. Afanasov, N. A. Bhatti, D. Campagna, G. Caslini, F. M. Centonze, K. Dolui, A. Maioli, E. Barone, M. H. Alizai, J. H. Siddiqui, and L. Mottola. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, 2020.

[2] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt. Riot os: Towards an os for the internet of things. In *2013 IEEE Conference on Computer Communications (INFOCOM) Workshops*, 2013.

[3] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac. Sytare: A lightweight kernel for nvram-based transiently-powered systems. *IEEE Transactions on Computers*, 68(9), 2019.

[4] B. Buchli, F. Sutton, J. Beutel, and L. Thiele. Towards enabling uninterrupted long-term operation of solar energy harvesting embedded systems. In B. Krishnamachari, A. L. Murphy, and N. Trigoni, editors, *Wireless Sensor Networks*. Springer International Publishing, 2014.

[5] A. Cammarano, C. Petrioli, and D. Spenza. Pro-energy: A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks. In *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, 2012.

[6] U. Datta, S. Dessouky, and A. Papagiannakis. Harvesting thermoelectric energy from asphalt pavements. *Transportation Research Record*, 2628(1), 2017.

[7] K. Geissdoerfer, M. Chwalisz, and M. Zimmerling. Shepherd: A portable testbed for the batteryless iot. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, 2019.

[8] J. Hester, L. Sitanayah, T. Scott, and J. Sorber. Realistic and repeatable emulation of energy harvesting environments. *ACM Trans. Sen. Netw.*, 13(2), apr 2017.

[9] N. Ikeda, R. Shigeta, J. Shiomi, and Y. Kawahara. Soil-monitoring sensor powered by temperature difference between air and shallow underground soil. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(1), Mar. 2020.

[10] H. Jayakumar, A. Raha, and V. Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 330–335, 2014.

[11] E. Lawrence and G. Snyder. A study of heat sink performance in air and soil for use in a thermoelectric energy harvesting device. In *Twenty-First International Conference on Thermoelectrics, 2002. Proceedings ICT '02.*, 2002.

[12] J. B. Lechevallier, R. A. R. Van Der Zee, and B. Nauta. Fast amp; energy efficient start-up of crystal oscillators by self-timed energy injection. *IEEE Journal of Solid-State Circuits*, 54(11), 2019.

[13] L. Mateu, C. Codrea, N. Lucas, M. Pollak, and P. Spies. Human body energy harvesting thermogenerator for sensing applications. In *2007 International Conference on Sensor Technologies and Applications (SENSORCOMM 2007)*, Oct 2007.

[14] A. Moser, M. Erd, M. Kostic, K. Cobry, M. Kroener, and P. Woias. Thermoelectric energy harvesting from transient ambient temperature gradients. *Journal of electronic materials*, 41(6), 2012.

[15] S. Pullwitt, U. Kulau, R. Hartung, and L. C. Wolf. A feasibility study on energy harvesting from soil temperature differences. In *Proceedings of the 7th International Workshop on Real-World Embedded Wireless Systems and Networks*, RealWSN'18, 2018.

[16] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on rfid-scale devices. *SIGPLAN Not.*, 46(3), Mar. 2011.

[17] L. Sigrist, A. Gomez, M. Leubin, J. Beutel, and L. Thiele. Environment and application testbed for low-power energy harvesting system design. *IEEE Transactions on Industrial Electronics*, 68(11), 2021.

[18] L. Sigrist, N. Stricker, D. Bernath, J. Beutel, and L. Thiele. Thermoelectric energy harvesting from gradients in the earth surface. *IEEE Transactions on Industrial Electronics*, 67(11), 2020.

[19] S. A. Tahami, M. Gholikhani, R. Nasouri, S. Dessouky, and A. Papagiannakis. Developing a new thermoelectric approach for energy harvesting from asphalt pavements. *Applied Energy*, 238, 2019.