An Emulation-based Method for Lifetime Estimation of Wireless Sensor Networks

Wilfried Dron*, Simon Duquennoy†, Thiemo Voigt†‡, Khalil Hachicha* and Patrick Garda*
*UPMC Univ Paris 6, UMR 7606, Laboratoire d'Informatique de Paris 6;
CNRS, UMR7606, LIP6;
F-75005, Paris, France
Email: {wilfried.dron,khalil.hachicha,patrick.garda}@upmc.fr
†SICS Swedish ICT AB, Sweden
Email: {simonduq,thiemo}@sics.se
‡Uppsala University, Sweden

***Abstract*—Lifetime estimation in Wireless Sensor Networks (WSN) is crucial to ensure that the network will last long enough (low maintenance cost) while not being over-dimensioned (low initial cost). Existing solutions have at least one of the two following limitations: (1) they are based on theoretical models or high-level protocol implementations, overlooking low-level (*e.g.*, hardware, driver, etc.) constraints which we find have a significant impact on lifetime, and (2) they use an ideal battery model which over-estimates lifetime due to its constant voltage and its inability to model the non-linear properties of real batteries. We introduce a method for WSN lifetime estimation that operates on compiled firmware images and models the complex behavior of batteries. We use the MSPSim/Cooja node emulator and network simulator to run the application in a cycle-accurate manner and log all component states. We then feed the log into our lifetime estimation framework, which models the nodes and their batteries based on both technical and experimental specifications. In a case study of a Contiki RPL/6LoWPAN application, we identify and resolve several low-level implementation issues, thereby increasing the predicted network lifetime from 134 to 484 days. We compare our battery model to the ideal battery model and to the lifetime estimation based on the radio duty cycle, and find that there is an average over-estimation of 36% and 76% respectively.**

I. Introduction

The network lifetime is a key parameter in Wireless Sensor Network (WSN) characterization [1]. Its accessibility at early design stages (*e.g.*, the firmware development) could enhance the efficiency of the application and the network by providing the developers with useful insights. Estimating lifetime, however, is challenging. It typically requires a network simulator, a detailed description of the application, an accurate battery model and a consideration of the hardware behavior/limitations.

The literature offers several solutions for WSN lifetime estimation. Most of them have a number of limitations, for example (1) they often use high-level descriptions to model the application in combination with (2) an ideal battery model (if they use a battery model at all) and/or (3) do not model the electrical behavior of the hardware components. To address these issues, we present a method that combines a node emulator, a technical specification-based battery model and a lifetime estimation tool (*c.f.*, Fig. 1). The node emulator executes the application firmware, compiled for its target micro-controller/platform, in the network simulator. Built on technical specifications, our battery model takes into account the most important characteristics of the battery, including parameters such as the non-linear discharge curve and supply voltage drifts. By combining our battery model with the emulated application, our lifetime estimation tool estimates the lifetime taking into account the electrical characteristics of the hardware (*e.g.*, cut-off voltage or current consumption variations with supply voltage).

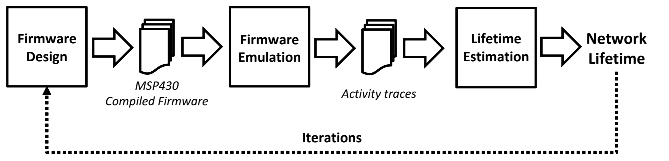


Fig. 1. **Lifetime Estimation Method:** from a MSP430 application firmware to its lifetime estimation. The firmware is designed iteratively based on the application objectives and output of our lifetime predictor. This method also allows to quantify the impact of application and network layer design choices on power consumption.

We demonstrate our method in a case study on a RPL/6LoWPAN-based data collection application. The fine-grained output of our lifetime estimation tool made it possible to redesign certain drivers and low-level aspects of the firmware. As a result, we increased the predicted application lifetime by a factor of 3.6 on the average, while not altering its operation. We also present experimental results that compare our technical specification-based lifetime prediction model to an ideal battery model and to duty cycle based estimation. Our results show that the ideal battery model and the duty cycle based overestimate the lifetime with 36% and 76% respectively.

This paper is organized as follows. The next section presents the background of this work and discusses related work. Section III details our method. Section IV presents a case study that shows how we used our tool to increase the application lifetime of a data collection network by a factor of three. We then compare in Section V the ideal battery model to a technical-specifications-based battery model in several scenarios. Before concluding, we discuss the limitations of our approach in Section VI.

II. Background and Related Work

Before reviewing the existing lifetime prediction solutions, we start this section with a short explanation about the application modeling and the battery behavior. We then discuss related work.

A. Application Modeling

Application software can be modeled at different levels of abstraction, resulting in different complexity/accuracy tradeoffs. As an example, emulators like MSPSIM [2] have a description level that is as low as the instruction of the MCU. They run the exact same firmware image, compiled for the target platform, as the actual application. As a result, there is no difference between the application modeled using emulation and the executed application. In contrast, TOSSIM [3] cross-compiles the application code and runs it directly on top of an abstracted hardware. Consequently, some applications that can run into the simulator might not run on the target platform [4]. To summarize, there is a tradeoff between the description level and the performance of the simulation: the higher the level of description, the faster the simulation and the lower the level of description, the more accurate the simulation.

B. Battery Behavior

Primary batteries are electro-chemical power sources. In contrast with wired power sources, they carry a limited amount of energy. This amount of energy is called *nominal capacity* if the battery is new or *residual capacity* (or *residual* in short) if it has been partially used.

The capacity (nominal or residual) varies with the ambient temperature and the instantaneous current draw (variation with the current draw is also known as *capacity-rate effect*) [5], [6]. In other words, the available amount of energy changes over time according to the aforementioned factors. The term *effective capacity* refers to the energy actually available considering battery's temperature and current draw. Note that a specific current draw can produce a drift in the battery supply voltage due to its *internal resistance*. Furthermore, the supply voltage also varies with the residual and the temperature.

One other property of batteries is the *relaxation effect* [7], [8]. When a high current is drawn from the battery, its capacity decreases which implies that the available energy is lower than the nominal value. This assumption is valid if the draw remains the same until the end of the battery life. If, for instance, the current draw returns to a value that is beyond the nominal current, the battery will *recover* some capacity.

C. WSN Lifetime Estimation

There exist number of lifetime estimation tools for WSN based on network simulators with different levels of abstraction.

One notable example is mTOSSIM [9], which extends the TinyOS [10] simulator TOSSIM [3] to enable lifetime estimation. It does so using a super-capacitor to model the power supply of the nodes. The super-capacitor behavior is very different from battery behavior (*c.f.*, Section II-B). As a consequence, this model cannot be used to estimate the lifetime of battery-operated nodes. The main drawbacks of mTOSSIM are the lack of a suitable battery model and the fact that the code is cross-compiled with TOSSIM, ignoring the hardware constraints such as thin-timing and hardware latency. The latter drawback is also shared by PowerTOSSIM [11] and PowerTOSSIM Z [12], other solutions based on TOSSIM.

Network simulators such as NS-2/3 [13] or OMNeT++ [14], [15] run high-level protocol implementations, and are not aimed at lifetime estimation. The Energy Framework [16] is an OMNeT++ extension for power consumption and lifetime estimation. While the initial release of this framework has been included in other frameworks like MiXiM [17] or Mobility [18], its lifetime estimation relies on an ideal battery model. Mikhaylov and Tervonen have addressed this limitation [19] by introducing a different battery model that they validated experimentally. Unfortunately, this validation does not consider battery supply voltage variations due to the current draw. Furthermore, it neglects the *internal resistance* of the battery and the *relaxation effect*. Another OMNeT++ framework extension has been introduced to address this issue [20], enabling low level description of the hardware characteristics. It provides also a way to model any battery using its technical specification.

Compared to our approach, the above solutions share the limitation of simulating abstracted applications that hide low-level implementation aspects that we capture through firmware emulation. To summarize, network lifetime estimation using (1) an emulated application with a (2) non-ideal battery model and a (3) low-level description of the node hardware has not yet been achieved.

III. Approach

In our lifetime estimation method, we first emulate the target application firmware. Based on the emulation results, we then predict the network lifetime. This can be used by application designers to predict network lifetime and to design their system iteratively. Our method (illustrated in Figure 1) consists in the following steps:

1) **Firmware Design** Designing and compiling the application firmware;
2) **Firmware Emulation** Emulating the firmware (possibly in a multi-node scenario where the radio medium is simulated);
3) **Lifetime Prediction** Predicting the lifetime from the emulation traces with our specification-based battery model;
4) **Design Iteration** Draw conclusions. Loop over to step 1 as long as needed.

This method has several advantages. First, by working on the actual application firmware, it avoids the common pitfall of ignoring certain low-level implementation aspects, or overlooking the effect of a given component (*e.g.*, frequent MCU wakeups, in a scenario that was initially thought as radio-intensive). Second, by using a specification-based battery model rather than an ideal model, we take into account the non-linear properties of batteries, giving results that are both more conservative and more accurate. This method is well-suited for iterative design, where the application designer refines the firmware until it attains the application performance objectives (*e.g.*, in reliability and latency) while reaching a given target lifetime. In this section we detail the emulation environment, our lifetime prediction framework, and the battery behavior it models.

A. Firmware Design

The first step of our method is the design and build of the initial firmware image. Working on compiled firmwares makes our approach independent of the operating system and the language used to program the application. In addition, working on the actual firmware rather than a high-level description of a protocol, as many other approaches do, makes it possible to measure the effect of all implementation aspects. Our case study on an out-of-the-box Contiki firmware (see Section IV) shows a number of examples where seemingly unimportant design decisions eventually shortened the network lifetime significantly. With our firmware-based method we were able to reveal and solve these issues.

B. Firmware Emulation

The second step of our method consists in emulating the application firmware. In this paper, we use the *T-Mote Sky* as example platform, and use MSPSim [21] to emulate sensor nodes. We modify MSPSim to log all changes in component state into traces files as $\langle timestamp, component, state\rangle$ tuples. In multi-node scenarios, we use the Cooja network simulator [22], which is able to simulate a network of emulated nodes (with MSPSim in our case). This results in a setting where the radio medium is simulated but the nodes emulated, running their actual firmware all the way from the application down to the radio drivers. Assuming the application has a periodic behavior, we run it for a given suitable duration, *e.g.*, 3 hours, and collect the logs before moving to the lifetime prediction step.

In cases the sensor network is event-driven or has non-periodic behavior, one option is to run it with some expected workload for a long enough duration. For example, for a system that issues alerts upon detecting presence in a room, simulate the system for several day with a realistic event rate.

C. Lifetime Prediction

Our lifetime prediction framework [20] uses the output traces of the emulations as its input. It plays the entire trace a first time, and then loops over it until the end of life of the node. In scenarios that involve a network bootstrap (*e.g.*, building a routing topology), we identify from the emulation traces a point where the network has stabilized, and have the lifetime predictor play the bootstrap part only once, and then loop over the rest of the trace.

The end-of-life of a node is defined as being the time until the supply voltage of the battery is too low to supply it. This specific voltage value is known as the "cut-off" voltage. In the case of the *T-Mote Sky* for instance, the cut-off voltage value is 2.7 Volts when the flash memory is used and 2.2 Volts otherwise [23]. The battery and the *T-Mote Sky* mote models are briefly introduced in the followings sub-sections.

MSP430	Typical		Worst		Average	
Voltage	2.2V	3.0V	2.2V	3.0V	2.2V	3.0V
LPM3 (μA)	1.1	2.0	1.6	2.6	1.35	2.3
LPM0 (μA)	50	75	60	95	55	85
Active@3.9Mhz (mA)	1.287	1.95	1.56	2.343	1.4235	2.145

CC2420	Typical
Power Down (μA)	20.0
Idle (mA)	0.426
RX (mA)	18.8
TX@0dBm (mA)	17.4

TABLE I. **TI MSP430F1611 and TI CC2420 current consumption**

1) Battery Model: Our battery model is based on technical specifications. It is different from the classical ideal battery model because:

- Its supply voltage varies according to its *residual*, the instantaneous current draw value and its *internal resistance*;
- Its discharge curve varies with its *residual* and the current draw value;
- It models the *capacity-rate* and *relaxation* effects.

Therefore, it is by essence, closer to the real battery behavior than the ideal battery model (*i.e.*, linear discharge curve and fixed supply voltage). Note that our model does not take into consideration temperature effects.

We model the impact of the current draw on the *relative capacity* using the equivalent current draw I_{eq}. The latter is defined in the following equation:

$$I_{eq} = \frac{C_{nominal}}{C_{relative}(i(t))} \times i(t) \tag{1}$$

where $C_{nominal}$ is the nominal capacity and $C_{relative}$ is the relative capacity (according to the instantaneous current draw), both expressed in mAh. The variable $i(t)$ is the instantaneous current expressed in mA. The factor between $C_{nominal}$ and $C_{relative}$ is called the *overdraw factor*. The battery *capacity-rate effect* is modeled into $C_{relative}(i(t))$ and the *relaxation effect* is modeled into the *overdraw factor* (*c.f.*, Sec. II-B). The battery *residual* Res at time $t + \Delta t$ is estimated using the following equation:

$$Res(t + \Delta t) = Res(t) - I_{eq} \times \frac{\Delta t}{3600} \tag{2}$$

where $Res(t)$ is the previous residual value in mAh, I_{eq} is the *equivalent current draw* (*c.f.*, eq.1) in mA, Δt the time while the I_{eq} current was drawn expressed in seconds. The value 3600 stands for the conversion of Δt in hours.

2) T-Mote Sky Model: The *T-Mote Sky* platform is modeled following the guideline of our lifetime prediction framework [20]. The framework requires the components to be modeled in both their behavior (functional model) and their power consumption (power model). The functional model is the trace player. The power model reproduces the electrical characteristics of the *T-Mote Sky*. The *T-Mote Sky* embeds a Texas Instruments MSP430F1611 micro-controller [24] and a Texas Instruments CC2420, an IEEE 802.15.4-compliant radio transceiver [25]. Their consumption figures are presented in Table I. These figures are extracted from the Texas Instruments' technical documentations [24], [25].

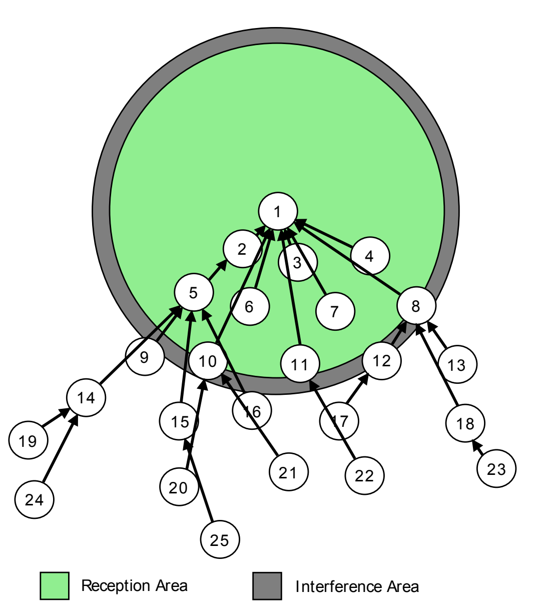


Fig. 2. **Network Topology in Cooja**: the RPL/6LoWPAN data collection application we use has 24 nodes connected through a 4-hop RPL topology to the network root, node #1.

D. Iterative Design

Finally, the output of our lifetime prediction framework provides the application designer not only with the expected lifetime but also with:

- *The current consumption profile* which is the time series of instantaneous current consumption. It makes periodic radio and MCU activities appear clearly for example.
- *The lifetime consumption distribution* which is the contribution of every component to the overall battery capacity consumption through the whole lifetime. It helps discovering anomalies, *e.g.*, components that consume a larger portion of the battery capacity than expected.

Based on this information, application designers can improve their design and implementation to avoid unwanted hotspots or may re-consider high-level application objectives. increasing the lifetime may require to have nodes sleep for longer periods, which in turns increases latency. In other cases, a lifetime above the expectations makes it possible to increase the application performance further than initially planned. In this paper we focus on the software aspects of the design, but in cases where the hardware is co-designed with the software, design decisions on the hardware itself can also be made based on the output of our lifetime prediction framework.

IV. Case Study

This section presents our case study. We use the default RPL/6LoWPAN collect example available in the Contiki OS as scenario for experiments[1]. The network topology is depicted in Fig. 2. The MAC layer is ContikiMAC, Contiki's default low-power listening MAC. We set the wake-up frequency to the default of 8 Hz, which leads to a baseline radio duty cycle, *i.e.*, when there is neither traffic nor radio noise, of 0.6%. The application requirement is to be able for each node of the network to transmit one data packet per minute to the network border router. We use *T-Mote Sky* as the sensor node platform (*c.f.*, Sec. III-C2). We use the *Duracell Plus Power MN1500* technical specification [26] to build our battery model (*c.f.*, Sec. III-C1). The mote is supplied by two 1.5V batteries. Therefore, we equip the node model with two instances of this battery model. We set the cut-off voltage of the battery model to 1.1 Volts each (2.2V in total) since our application does not use the external flash memory [23].

[1]The Contiki-2.6 release, and the `examples/ipv6/rpl-collect` application.

		Sleep	MCU	Radio	DC	Lifetime
Example	Av.	**57.4%**	**20.1%**	**22.5%**	**0.798%**	**133.94 days**
	Max.	58.3%	20.3%	26.2%	0.988%	135.99 days
	Min.	54.1%	19.7%	21.4%	0.742%	126.75 days
	Std. Dev.	1.11%	0.16%	1.23%	0.062%	2.38 days
Enhanced	Av.	**13.0%**	**5.5%**	**81.5%**	**0.802%**	**484.04 days**
	Max.	13.4%	8.4%	83.5%	0.981%	515.11 days
	Min.	12.1%	4.3%	79.0%	0.745%	383.60 days
	Std. Dev.	0.38%	0.96%	1.00%	0.062%	35.16 days

TABLE II. **Consumption Profiles of the Example and Enhanced Firmwares.** Our enhanced firmware reduces the consumption of the MCU and that of sleep mode. This results in a 3-fold improvement of the lifetime: from 134 days to over a year.

A. Lifetime Estimation and Instant Consumption

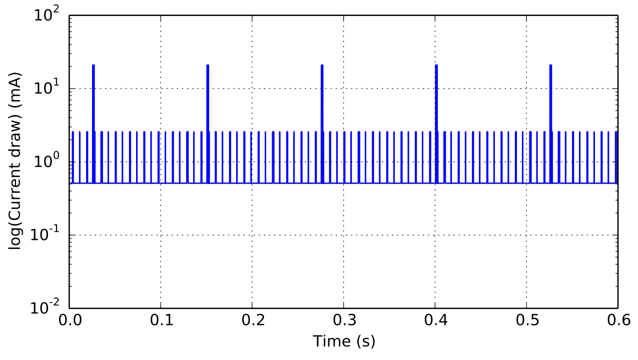
The execution of the binary example "out of the box" using our method reveals an average lifetime (averaged over all nodes in the network) of 134 days. As shown in Table II ("Example" row), the average consumption of each node of the network can be attributed for 57.4% to sleep mode, 20.1% to CPU, and 22.5% only for radio listening and transmission.

We now look at a single node produce an energy profile of it. We choose node #2, one hop from the root (were the bottleneck of the network often lies [27]). Figure 3a shows the node's current consumption as a function of time, in a selected window of 0.6 seconds. This time series reveals two sources of energy waste: (1) frequent peaks to about 2.57mA, due to periodic CPU activity, and (2) consumption of 511μA during sleep periods. These results highlight that the MCU is waking up too frequently and that the battery capacity spent in sleep mode is too high. We address both issues in order to reach a longer lifetime, without affecting the application performance (nodes sending to the root at 1 minute interval with 100% end-to-end delivery ratio and average duty cycle around 0.8%).

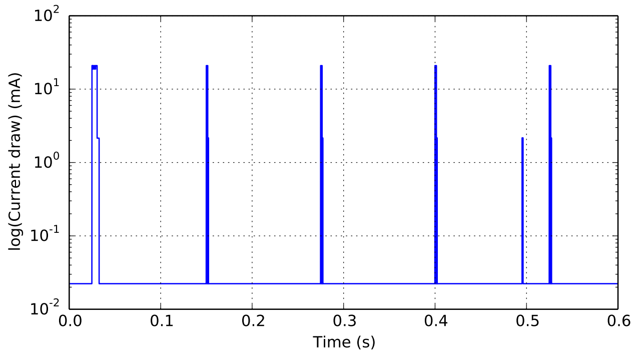
B. Minimizing power consumption

We address the first issue by disabling an unnecessary software timer that was checking for serial line input at a frequency of 64 Hz. We then were able to decrease the frequency of Contiki's software clock and timer engine from 128 Hz to 2 Hz; a change that did not affect the application performance as no remaining timer in the system was set to more than 2 Hz.

The second issue is that the radio was never turned into the *power down* mode. Instead of consuming 20μA the chip was using 426μA, which correspond to the *idle* mode consumption

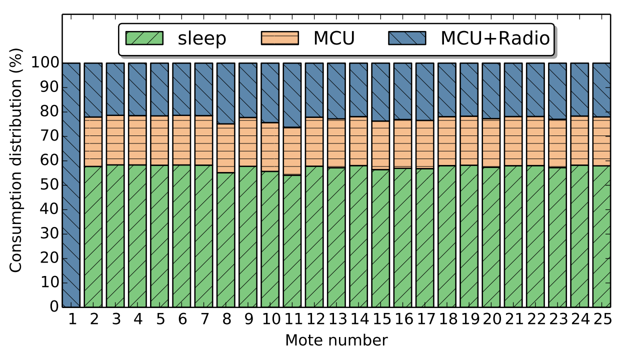


(a) Example Firmware

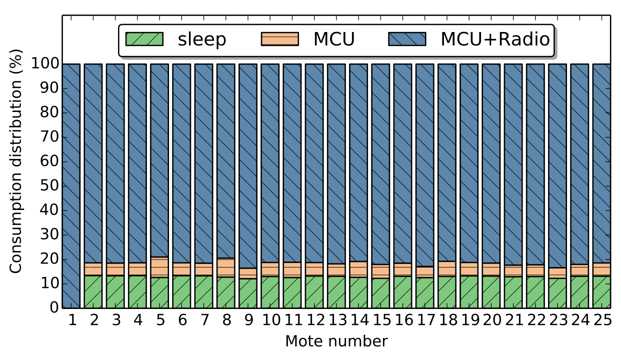


(b) Enhanced Firmware

Fig. 3. **Power Trace of the Example and Enhanced Firmwares.** Note the log scale. Our enhanced firmware suppresses a number of unnecessary CPU wakeups (64 Hz peaks in the "Example" case) and lowers the baseline by using the CC2420 *power down* mode instead of *idle*.



(a) Example Firmware



(b) Enhanced Firmware

Fig. 4. **Power consumption distribution of the Example and Enhanced Firmwares.** The enhanced firmware reduces MCU usage and reduces the baseline cost in sleep mode. Its consumption profile is consequently dominated by the radio.

(*c.f.*, Sect.III-C2). At the time of writing we are currently implementing the missing features in the cc2420 driver and ContikiMAC protocol that will make it possible to use the *power down* mode during long periods of sleep. Note that such implementation requires some changes in the ContikiMAC timings, as waking up from *power down* mode requires to stabilize the crystal oscillator again, which takes a longer time than waking up from *idle*. To get an estimate of the results after implementing these changes, we run our Lifetime Predictor again now assuming *power down* instead of *idle*.

C. Enhanced firmware

The current consumption profile of the enhanced firmware is shown in Figure 3b. Our improvements reduce the sleep consumption from 511µA to 22.3µA. It is also noticeable that there is less MCU activity (2.57mA spikes).

Table II reports the lifetime consumption distribution figures for both firmware versions. Figure 4a and Figure 4b present the same results for every individual node using respectively the example and the enhanced firmware. The fraction of the battery capacity consumed in sleep mode is reduced from 57% ("Example" row) to below 13% ("Enhanced" row) which is expected since node are sleeping most of the time (average duty cycle of 0.8%, *c.f.*, Fig. 6). The fraction of the MCU consumption is reduced to a similar extent, by almost 4 times. Overall, the radio is now the component responsible for most of the battery's capacity consumption (81.5% on the average), which is expected in this duty cycled data collection scenario.

As a result, we increased the lifetime of the initial firmware by more than three times, while fulfilling the application requirements of 1 packet/minute with no significant change in average duty cycle (*c.f.*, Tab. II). The lifetime estimated for each node is presented in the Figure 5.

V. Results

In this section we present the results of two experiments.

The first experiment deals with lifetime estimation using a technical specification-based battery model in comparison with the ideal battery model and with duty cycle based estimations. The purpose of this experiment is to showcase the impact of the estimation methods on the estimated lifetime and provide an understanding of the expected difference when using an ideal battery model or the duty cycle based lifetime estimation method.

The second experiment is a study on the impact of the wake-up period over the lifetime of the nodes when using our battery model or the duty cycle based estimation. This experiment demonstrates how the lifetime predicted by our method can differ from the duty cycle based estimation in more or less radio-intensive scenarios. This allows designers to better understand the relation between duty cycle and lifetime,

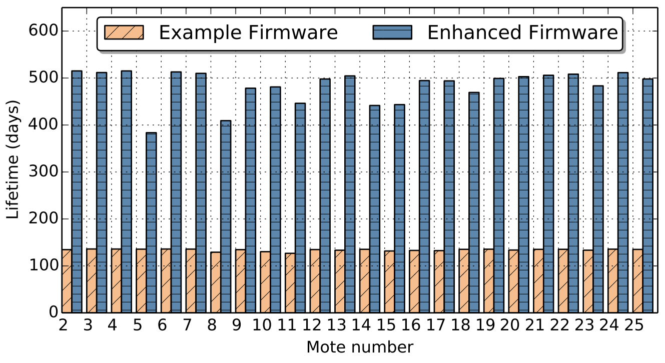


Fig. 5. **Lifetime Estimates for the Example and Enhanced Firmwares.** Our enhanced firmware increases lifetime dramatically for all nodes.

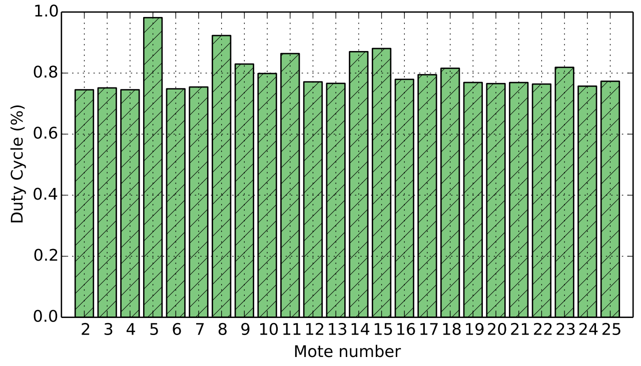


Fig. 6. **Per-node Duty Cycle:** The distribution of the duty cycle over the network is homogeneous, with all nodes in the range 0.75% to 1%.

helping application design where the duty cycle is the only available metric.

We use the same setup as in the case study, and use the "enhanced" version of the firmware (*c.f.*, Sect. IV).

A. Duty cycle based Estimation

Before reviewing our results, we briefly introduce the duty cycle-based lifetime estimation method we compare against.

The duty cycle over a single period DC_T is computed using the following formula:

$$DC_T = \frac{t_{on}}{t_{on} + t_{off}} = \frac{t_{TX} + t_{RX}}{T} \quad (3)$$

Where t_{TX} and t_{RX} are respectively the time spent transmitting and the time spent receiving or listening and T is the length of the period, expressed in seconds. In our experiment we consider T as being the whole observation time. In other words, the duty cycle is observed over the complete lifetime of the node. Its value is depicted for each node of the network in the Figure 6. It is noticeable that the distribution of the duty cycle is quite homogeneous over the whole network.

To estimate the lifetime of a single node using the duty cycle, we compute the average consumption i_T using the following equation:

$$i_T = \frac{(t_{TX} \times i_{TX}) + (t_{RX} \times i_{RX})}{T} + (1 - DC_T) \times i_{PD} \quad (4)$$

Where i_{TX}, i_{RX} and i_{PD} are the current consumed respectively in TX, RX and *power down* mode by the RF transceiver in mA. We then compute the duty cycle-based lifetime estimate LT as follows:

$$LT = \frac{C_{nominal}}{i_T} \quad (5)$$

Where $C_{nominal}$ is the nominal capacity of the battery expressed in mAh (3360mAh in our case) and i_T the average consumption over one period expressed in mA.

B. Comparison of Lifetime Estimation Methods

Figure 7 presents the lifetime predictions using our approach, the ideal battery model and the duty cycle based estimation method. Our battery model estimates a median lifetime of 497.94 days. The duty cycle based estimation is the most optimistic with 877.07 days. The ideal battery model results are intermediate, reaching 680.87 days.

We look at the difference of the ideal battery model and the duty cycle based method prediction, relatively to our prediction. It appears that the difference is almost constant between our model and the ideal one (36.74% average) and between our model and the duty cycle (76.24% average). Regarding the duty cycle based lifetime estimate, node #5 is slightly standing out with a relative difference of 84.8%. This node has the highest duty cycle (close to 1%, *c.f.*, Fig. 6), and is also responsible for relaying packets from nodes #9, #14, #15 and #16 to the border router #1 (*c.f.*, Fig. 2). As a consequence, its consumption repartition is 12.64% spent in sleep mode, 8.37% for the MCU only and 78.99% for the radio+MCU which means that it consumes less energy in the radio than the rest of the network (*c.f.*, Tab. II, "Enhanced" row). In this case, the duty cycle based method overestimates the lifetime slightly more that for other nodes.

C. Impact of the Wake Up Period

In this experiment, we focus on three nodes only since the nodes' lifetime is quite homogeneous in the network (*c.f.*, Tab. III). We selected node #5 which has the shortest lifetime with 383 days, node #6 which has among the longest lifetimes with 513 days and node #17 which has a 494 days lifetime (closest to the median lifetime or 417 days). We vary ContikiMAC wakeup in the range 16 Hz to 2 Hz and observe its impact on lifetime.

	Tech-based Model	Ideal Model	Duty Cycle
Median	497.94 days	680.87 days	877.07 days
Max.	515.11 days	704.97 days	904.68 days
Min.	383.60 days	518.91 days	708.93 days
Std. Dev.	35.16 days	49.77 days	52.97 days

TABLE III. **Lifetime estimation summary**

Table IV reports the duty cycles for different ContikiMAC channel check rates (CCR) and nodes. As expected, the duty cycle decreases with the CCR. There are a few exceptions that we attribute to the statistical distribution, which is slightly different from one simulation to the other. Note that the order of the nodes is not conserved among runs for the same reason.

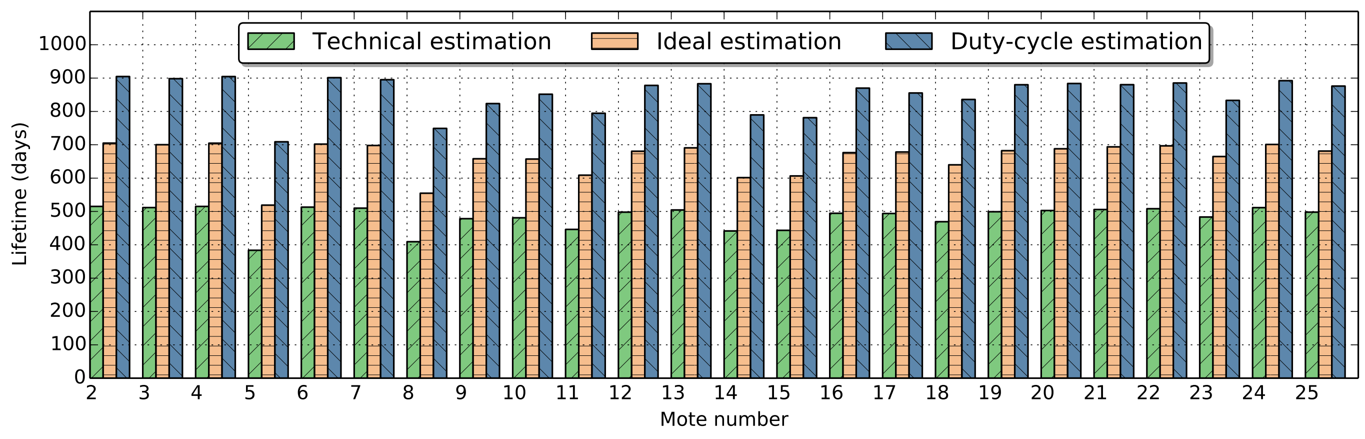


Fig. 7. **Comparison Between our Model, the Ideal Battery Model and Duty Cycle Based Lifetime Estimation.** For all nodes, the three methods rank similarly, with our method yielding the most conservative results. The relative difference between different methods is nearly constant accross different nodes.

Node	CCR=16Hz	CCR=8Hz	CCR=4Hz	CCR=2Hz
5	1.657 %	0.982 %	0.414 %	0.487 %
6	1.489 %	0.748 %	0.411 %	0.301 %
17	1.520 %	0.795 %	0.451 %	0.298 %

TABLE IV. **Effect of CCR on Duty Cycle**

Table V shows the lifetime estimates based on our approach and on the duty cycle based estimation. We find that while the lifetime increases when the duty cycle decreases, the relation is not proportional.

Technical Specification Based Battery Estimation

Node	CCR=16Hz	CCR=8Hz	CCR=4Hz	CCR=2Hz
5	247.43 days	383.59 days	825.15 days	555.57 days
6	277.35 days	512.95 days	831.10 days	932.67 days
17	273.10 days	493.91 days	775.19 days	946.69 days

Duty-cycle Based Estimation

Node	CCR=16Hz	CCR=8Hz	CCR=4Hz	CCR=2Hz
5	438.16 days	708.94 days	1476.01 days	1296.32 days
6	484.21 days	901.32 days	1484.43 days	1881.45 days
17	474.87 days	855.34 days	1378.72 days	1895.89 days

TABLE V. **Lifetime Estimation at Different Channel Check Rates** With both duty cycle based estimation and our model, the lifetime increases at a lower CCR. However, the ratio between both estimates is not constant. This is attributed to different CCR leading to different energy profiles, with a variable contribution of the radio to the overall consumption.

Figure 8 shows a comparison of the consumption distribution. The graph shows that the contribution of the radio to the overall consumption decreases to 56% in the worst case (node #5, CCR=2 Hz). The relative consumption of the MCU reaches more than 20%. As a consequence, estimations based on the duty cycle are biased explaining the difference in the estimation results (*c.f.*, Tab. V). This bias was outlined as well for high duty-cycle (100% down to 25%) in an experimental study on the network lifetime [28].

VI. Discussion

The main limitation of our method is the lack of ground truth, a limitation we share with many other efforts that incorporate battery models. We base our model on the technical specifications of the battery, assuming that these specifications

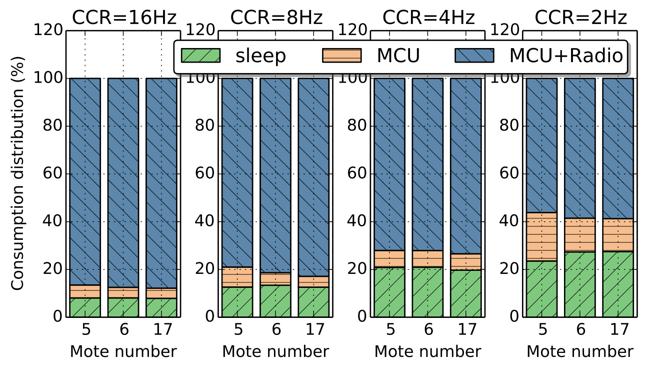


Fig. 8. **Power Consumption Distribution under Different ContikiMAC Channel Check Rated (CCR) for nodes #5, #6 and #17.** Less frequent channel checks lead to a higher relative contribution of sleep mode and MCU – adversely affecting the accuracy of duty cycle based lifetime estimation.

are close to the actual battery behavior in common settings. The ideal battery model is known to over-estimate lifetime by 40%-50% on the average (for the closest cases) [19]. In our experiments, we find a similar ratio between the ideal and our model. We are therefore confident our lifetime prediction is close to the ground truth.

We are currently designing an experiment that will allow us to validate the battery behavior in a shorter timespan than the almost 500 days median lifetime of the nodes in our case study. The experiment will enable us to address another limitation: the fact that we consider the temperature as being constant. The technical specifications we build our model upon hold only for an ambient temperature of +21 C. Modeling the effect of temperature over the whole operation range of the battery (-20 C to +35 C) would require new series of measurements, as a complement to the official technical specification.

Finally, even though we use firmware emulation to produce our traces, the fact that we simulate the radio medium leads to some inaccuracies. In real deployments, nodes experience losses and react with retransmissions and routing topology adaptation. There is, however, very little one can do to predict the occurrence of events accurately under the entire lifetime of the network. Getting closer lifetime estimates would require to actually pre-deploy the application in its target environment, record all component state changes as we do during emulation, and then run our lifetime predictor on the traces. This is impractical in most cases, as deploying may be too expansive if possible at all in early design stages, and as logging all state changes from software would have significant side effects and require much storage space (our experiments produced between 5000 and 20000 state changes per node and per minute). In contrast, we believe our approach is very practical, enabling low-cost iterations at early stages of the application design.

VII. Conclusions

In this paper we introduced a method to estimate the network lifetime using an emulated application with a non-ideal battery model and a low-level description of the node hardware. We implement this method and demonstrate its applicability to several scenarios. In a case study, we increased the initial lifetime of our application by a factor greater than three while keeping the same performance and fulfilling the application requirements. Moreover, we highlighted the fact that certain design decisions that appear not to influence the lifetime significantly from a designer perspective, do have in fact a strong impact on it. Our experiments demonstrated the limitations of the ideal battery model and the bias that is introduced when the duty cycle is used to estimate lifetime. Both methods overestimate the lifetime dramatically, in the range of 36 to 76%.

Acknowledgment

The authors would like to thank the EIT ICT Lab, which funded this collaboration between SICS, Sweden and the LIP6 laboratory, France. This work was also partly funded by SSF, the Swedish Foundation for Strategic Research, and VINNOVA, Swedens Innovation Agency.

References

[1] I. Dietrich and F. Dressler., “On the lifetime of wireless sensor networks,” *ACM Transaction on Sensor Network*, pp. 1–38, 2009.

[2] *MSPSIM Emulator git-hub*. [Online]. Available: https://github.com/mspsim/mspsim

[3] *TOSSIM*. [Online]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM

[4] *TOSSIM: A Simulator for TinyOS Networks*. [Online]. Available: http://www.tinyos.net/tinyos-1.x/doc/nido.pdf

[5] D. Pomerantz, “The characterization of high rate batteries,” *Consumer Electronics, IEEE Transactions on*, vol. 36, no. 4, pp. 954–958, 1990.

[6] K. Mikhaylov and J. Tervonen, “Experimental evaluation of alkaline batteries’s capacity for low power consuming applications,” in *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, 2012, pp. 331–337.

[7] L. M. Feeney, L. Andersson, A. Lindgren, S. Starborg, and A. A. Tidblad, “Using batteries wisely,” in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys)*, 2012.

[8] C. Rohner, L. Feeney, and P. Gunningberg, “Evaluating battery models in wireless sensor networks,” in *Wired/Wireless Internet Communication*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7889, pp. 29–42.

[9] J. Mora-Merchan, D. Larios, J. Barbancho, F. Molina, J. Sevillano, and C. Len, “mTOSSIM: A simulator that estimates battery lifetime in wireless sensor networks,” *Simulation Modelling Practice and Theory*, vol. 31, no. 0, pp. 39 – 51, 2013.

[10] *Tiny OS*. [Online]. Available: http://www.tinyos.net/

[11] *PowerTOSSIM*. [Online]. Available: http://www.eecs.harvard.edu/~shnayder/ptossim/

[12] E. Perla, A. O. Catháin, R. S. Carbajo, M. Huggard, and C. Mc Goldrick, “Powertossim z: realistic energy modelling for wireless sensor network environments,” in *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*. ACM, 2008, pp. 35–42.

[13] *NS-2/3*. [Online]. Available: http://www.isi.edu/nsnam/ns/

[14] *OMNeT++*. [Online]. Available: http://www.omnetpp.org/

[15] A. Varga and R. Hornig, “An overview of the omnet++ simulation environment,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, pp. 60:1–60:10.

[16] L. Feeney and D. Willkomm, “Energy framework: An extensible framework for simulating battery consumption in wireless networks,” in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 2010, pp. 20:1–20:4.

[17] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin, “Simulating wireless and mobile networks in omnet++ the mixim vision,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, pp. 71:1–71:8.

[18] W. Drytkiewicz, S. Sroka, V. Handziski, A. Koepke, and H. Karl, “A mobility framework for omnet++,” in *3rd Int’l OMNET++ Workshop*, 2003.

[19] K. Mikhaylov and J. Tervonen, “Novel energy consumption model for simulating wireless sensor networks,” in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress on*, 2012, pp. 15–21.

[20] W. Dron, K. Hachicha, and P. Garda, “A fixed frequency sampling method for wireless sensors power consumption estimation,” in *New Circuits and Systems Conference (NEWCAS), 2013 IEEE 11th International*, 2013, pp. 1–4.

[21] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón, “Cooja/mspsim: Interoperability testing for wireless sensor networks,” in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ser. Simutools ’09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 27:1–27:7. [Online]. Available: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5637

[22] Österlind, F. and Dunkels, A. and Eriksson, J. and Finne, N. and Voigt, T., “Cross-level sensor network simulation with cooja,” in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, Nov 2006, pp. 641–648.

[23] *T-Mote Sky data sheet*. [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

[24] *Texas Instruments MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller (Rev. G)*. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f1611.pdf

[25] *Texas Instruments CC2420 2.4 GHz IEEE 802.15.4 - ZigBee-Ready RF Transceiver (Rev. C)*. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf

[26] *Duracell Plus Power MN1500 AA*. [Online]. Available: http://media.professional.duracell.com/downloads//datasheets/product/Plus%20Power/Plus_AA_MN1500.pdf

[27] J. Alonso, A. Dunkels, and T. Voigt, “Bounds on the energy consumption of routings in wireless sensor networks,” in *Proceedings of the 2nd WiOpt, Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, Cambridge, UK, Mar. 2004. [Online]. Available: http://www.sics.se/cna/dtnsn/publications/WiOpt04FV.pdf

[28] H. A. Nguyen, A. Forster, D. Puccinelli, and S. Giordano, “Sensor node lifetime: An experimental study,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, March 2011, pp. 202–207.