Surviving the Hair Dryer: Continuous Calibration of a Crystal-Free Mote-on-Chip

Tengfei Chang, Thomas Watteyne, Brad Wheeler, Filip Maksimovic, David C Burnett, Kris Pister

Surviving the Hair Dryer: Continuous Calibration of a Crystal-Free Mote-on-Chip

Tengfei Chang*, Thomas Watteyne* *Senior, IEEE*, Brad Wheeler†, Filip Maksimovic†, David C. Burnett†, Kris Pister†

* EVA team, Inria, Paris, France

† BSAC, University of California, Berkeley, CA, USA

***Abstract*—The single chip micro-mote (SCμM) is a 2×3 mm^2 single-chip crystal-free mote-on-chip. SCμM implements the IEEE802.15.4 and BLE standards and can communicate with off-the-shelf radios compliant to those standards. SCμM exclusively uses on-chip oscillators, including a 2.4 GHz LC oscillator to synthesize the communication frequency, and a 2 MHz RC oscillator to clock the chip rate. The challenge is that the LC oscillator drifts at 2,100 ppm over a temperature range of 45°C, far from the 40 ppm maximum drift mandated by the IEEE802.15.4 standard. While one-shot calibration is possible, any temperature change causes IEEE802.15.4 communication to fail. This paper describes a continuous calibration approach for SCμM to adapt the tuning of its oscillators as the temperature changes. Experimental results show that it allows SCμM to keep communicating with an IEEE802.15.4 radio even under the extreme condition of using a hair dryer to heat up the chip at 3°C/min. Under these conditions, the drift of the LC oscillator stays within the ±40 ppm limit over 94% of the time. Similarly, the drift of the 2 MHz RC oscillator stays within ±1,000 ppm limit 99.98% of the time.**

***Index Terms*—IEEE802.15.4, calibration, crystal-free, single chip micro-mote.**

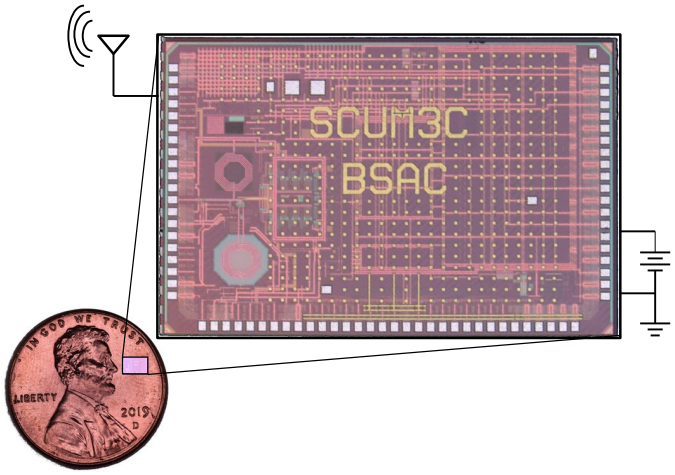


Fig. 1. The single chip micro-mote (SCμM) is a 2×3 mm^2 crystal-free mote-on-chip. It functions without any external components other than a power source and an antenna, and implements the IEEE802.15.4 [2] and BLE standards.

I. Introduction

The single chip-micro mote, SCμM [1], is a 2×3 mm^2, standard-compliant, crystal-free mote-on-chip. It features a Cortex-M0 micro-controller, a 2.4 GHz transceiver, and an optical receiver. SCμM's radio implements the IEEE802.15.4 [2] and BLE standards, allowing it to communicate with off-the-shelf radios compliant to those. The use of on-chip oscillators eliminates the need for external crystal oscillators, and in most cases the need for a Printed Circuit Board altogether. The ultra small form factor of SCμM enables applications where size matters, including medical implants [3], wearables [4] and wireless body sensor networks (WBSN) [5]. SCμM realizes the "Smart Dust" vision, enabling tiny motes be deployed ubiquitously [6].

SCμM is a true single-chip solution, and therefore does not require external (crystal) oscillators. SCμM relies on 4 *internal* ("on-chip") oscillating circuits for timekeeping: the "HF oscillator" (a 20 MHz RC oscillator to clock the Cortex-M0), the 2 MHz RC oscillator (to time the modulating and data rate when transmitting), the 64 MHz RC oscillator (to time the modulating and data rate when receiving, as well as the internal analog-to-digital converters), and the 2.4 GHz "LC oscillator" (to control the communication frequency of the IEEE802.15.4 radio). The reason for using 3 discrete RC oscillators is power consumption concern. An RC oscillator's power scales approximately with frequency. With 3 discrete RC oscillators, SCμM can turn on or off any RC oscillators as needed to save power. The major challenge of using RC/LC oscillators is that they exhibit a drift orders of magnitude higher than crystal oscillators, and that their frequency changes significantly with temperature [7]. SCμM's LC oscillator exhibits a 2,100 ppm drift over a 45 $^\circ C$ temperature ramp [8]; under these conditions typical crystal oscillators drift by 10-40 ppm. It is therefore essential to continuously calibrate the oscillators to be able to keep communicating as temperature changes.

This goal of the research presented in this paper is to design and evaluate a "continuous calibration" approach in which SCμM tracks and compensates for the drift of its oscillators

Corresponding author: Tengfei Chang (email: tengfei.chang@inria.fr).

so it keeps communicating with an off-the-shelf IEEE802.15.4 device over temperature. We conduct this study over SCμM as it is the most advanced standards-based single-chip crystal-free mote to date. Lessons learnt and results do carry over to other chips. To the best of our knowledge, this paper is the first time introducing an overall calibration system for SCμM, including the 2 MHz RC oscillator and the LC oscillator for transmitting and receiving.

To know the frequency offset of the LC oscillator while *receiving*, SCμM monitors the intermediate frequency (IF) of the "I" channel samples. In the design of the SCμM radio, the incoming 2.4 GHz signal is shifted to a 2.5 MHz IF as part of reception. It is much easier to manipulate the incoming signal at the relatively slow IF. Any inaccuracy on the frequency of the LC oscillator directly translates in inaccuracy on the value of the IF.

Firmware running on SCμM can measure the value of the IF by counting its zero-crossings during some window of time [9]. That is, the IF should be 2.5 MHz, which translates into 5 M zero-crossings per second (one low-to-high, one high-to-low per cycle). Over a 100 μs window (the value we use), we expect 500 zero-crossings. To accurately time this window, SCμM uses a 16 MHz down-converted version of its 64 MHz RC oscillator. The calibration routine hence continuously tunes the DAC setting of the LC oscillator to keep close to 500 zero-crossings of the IF signal over a 100 μs window.

To know the frequency offset of the LC oscillator while *transmitting*, SCμM requires the radio it is sending frame to provide feedback about whether it got the frame and, ideally, how "accurate" the frequency was. Interestingly, the Texas Instruments CC2538 IEEE802.15.4 SoC measures the frequency offset of all incoming frames available to the application through its `FREQEST` register. Firmware on the CC2538 can write the value of that register in acknowledgement frames to SCμM.

To tune its 2 MHz RC oscillator, SCμM needs a reference time interval, for example by having an external mote send frames at well-defined times. If SCμM receives frames at a period it knows to be 30 ms, it can use the counter driven by the 2 MHz RC oscillator to count the number of cycles. That is, tune the 2 MHz RC oscillator until the counter counts exactly 2 MHz·30 ms = 60,000 ticks between frames received.

The contributions of this paper are three-fold:

- We propose a novel calibration approach to continuously calibrate the on-chip 2 MHz RC oscillator, and the on-chip 2.4 GHz LC oscillator for both transmitting and receiving.
- Specifically, we describe a hopping technique to track frequency using two frequency settings to avoid losing the target frequency because of the non-linearity of the frequency tuning.
- We demonstrate that the resulting continuous calibration keeps the LC frequency for transmission and reception within $\pm$40 ppm of drift over 94% of the time under the extreme condition of heating up SCμM using a hair dryer. Under the same conditions, the drift of the 2 MHz RC oscillator stays between $\pm$1 000 ppm 99.98% of the time.

The remainder of this paper is organized as follows. Section II summarizes the related work on frequency calibration, with a particular focus on handling temperature changes. Section III is a preliminary study on the drift of SCμM's oscillators over temperature, also highlighting the non-linearity of the frequency tuning. Section IV serves a problem statement for the paper, by showing that SCμM is unable to maintain communication with OpenMote under a temperature change as little as 2°C. Section V describes the proposed continuous calibration approach, specifically how it sweeps over all settings to get the initial settings of LC oscillator, how it computes the frequency offset for the 2 MHz RC oscillator and the 2.4 GHz LC oscillator, and how it handles the non-linearity of the frequency tuning. Section VI demonstrates the correctness of the continuous calibration approach by showing the frequency settings over time, and evaluates its performance by analyzing the distribution of IF counts, frequency offset and 2M counts over time. Finally, Section VII concludes this paper and discusses current and future work.

II. Related Work

This section surveys frequency calibration approaches in low-power wireless electronics, with a particular focus on compensating temperature variation.

Ding *et al.* [10] implement a Frequency Locked Loop (FLL)-based timer that integrates an on-chip trimming approach. This allows the timer to self-calibrate over a -40°C to 80°C temperature range. Trimming is done using two resistors in the clock circuit: the clock circuit is able to switch between the two resistors, resulting two frequencies. By playing with the duty-cycle of the resistors, the clock compensates for the change in frequency induced by a temperature change. This approach results in an 8 ppm/°C drift.

Marin *et al.* [11] present a synchronization protocol for wireless sensor networks, called "Temperature-Aware Compensation (TACO)". The authors discuss the impact vibration interference and mismatched capacitors can have on clock skew, and show how that can be eliminated using proper PCB design. To model the clock skew over temperature, the authors manually measure the skew at multiple fixed temperature points. The clock skew between two adjacent temperature points is estimated through least square fitting. The sensor nodes running TACO use their on-board temperature sensor to continuously compute their clock offset, based on the model mapping temperature to clock skew. It results a clock skew below 1 ppm 92% of the time, when the nodes run the TACO prediction model.

David *et al.* [12] develop an adaptive synchronization method for IEEE802.15.4 network to combat the crystal drift over temperature changes. The approach allows a node in the network to adjust its clock locally based on the estimated drift. Since the drift various over the temperature changes, the proposed approach monitors the temperature through a temperature sensor and re-estimate the drift when the temperature change exceeds a pre-defined threshold. The result shows that adaptive synchronization is able to brings a node with 11 ppm down to 1.5 ppm.

Borja *et al.* [13] propose another synchronization approach for long-term synchronization interval. Instead of simply re-calculating the drift when temperature change exceeds a threshold, it maintains a reference drift table at different temperatures. A fitting model of drift over temperature is developed based on the table of history drifts recorded at different temperatures over time. In simulation scenario, the drift prediction model can manage the accumulated drift over 24 hours within 10 ms, which is about 0.1 ppm.

The major difference between SCμM and the work presented in [12], [13] and [11] is that they are dealing with a crystal-based clock, rather than RC or LC oscillators. Typically, the synchronization approaches presented in these articles rely on sending a packet to obtain the time offset and calculates the drift based on the offset. The radio to send the packet is driven by a crystal clock which has high accuracy. For SCμM, the LC oscillator needs to be firstly calibrated so that SCμM can receive and send a standard-compliant packet. As a result, the synchronization approaches presented in the related articles can not be applied to the oscillators of SCμM for transmitting and receiving packet.

Titan *et al.* [14] present a method to calibrate SCμM for use as an IoT temperature sensor between 0°C and 100°C. The method finds a linear relationship between the ambient temperature and the ratio of these two clock frequencies. The proposed method allow SCμM used as an temperature sensor with 2°C errors. This gives the chance for SCμM to calibrate its radio frequency based on a frequency over temperature model, which is not presented in the article.

Lee *et al.* [15] design a 4.19 MHz real-time clock (RTC) generator that includes temperature compensation. The on-chip temperature compensation circuit controls the value of a divider for the output frequency. According to the input of the on-chip temperature sensor, the logic unit of the circuit computes the corresponding divider value using a pre-programmed algorithm model. That model is built offline, based on the measurements of the divider values over the temperature range of -40°C to 60°C. With this method, the measured frequency error of the generated clock is managed to be below 1 ppm over the same temperature range.

Wheeler *et al.* [16] study the stability of SCμM's LC oscillator. By placing an early version of the SCμM chip running a temperature compensation algorithm into a temperature-controlled chamber, at a constant 25 °C, the LC oscillator drifts by $\pm$40 ppm over 13 hours. This is done using an off-chip current source and regulator. Over a 50 °C temperature ramp, the LC oscillator drifts by 4,000 ppm. The authors propose a receiver-based feedback approach to counteract the impact of temperature changes. Each time SCμM receives a frame, it monitors the "I" channel samples and counts the zero-crossings during a 100 μs window. Based on the measured number of zero-crossings, SCμM adjusts the frequency of the LC oscillator. We use the same technique in the "continuous calibration"(CoCa) solution presented in Section V. However, this is only used for calibrating the frequency during receiv-ing. The CoCa approach in this paper firstly introduces a calibration mechanism for 2 MHz oscillator and a "setting hopping" method to track the frequency over temperature changes, which are presented in Section V-D and Section V-E.

Suciu *et al.* [17] provide a calibration technique for SCμM for both transmission and reception. The LC frequency increases in a non-linear manner with the frequency setting. The authors first linearize the settings over frequency through two approaches: recursive least squares (RLS) and moving average (MA). They sweep the entire linearized range of frequency settings to find the setting that corresponds to each of the 16 IEEE802.15.4 channels. They repeat this over the 5°C to 55°C temperature range, in 5°C increments. SCμM tunes the frequency of its 2.4 GHz LC oscillator based on a model of the frequency over temperature extracted from those measurements. Results show that the average frequency drift on all 16 channels is less than 80 ppm when using the MA approach, a value close to the target 40 ppm mandated by the IEEE802.15.4 standard.

The frequency tracking approach proposed in this paper is a comprehensive calibration approach for SCμM to continuously calibrate the on-chip oscillators (2 MHz RC oscillator, 2.4 GHz LC oscillator for transmission and reception). Unlike Suciu *et al.* [17], our approach does not require to model a non-monotonic DAC using a monotonic function. Our approach allows SCμM to keep communicating with off-the-shelf IEEE802.15.4 devices even when temperature changes quickly. Unlike the approach in Suciu *et al.* [17] which uses linearized settings that vary from chip to chip, our approach function on all chips without needing chip-specific pre-calibration. To deal with the non-linear frequency tuning, we propose a mechanism to constantly "hop" from one setting to another (see Section V-E).

III. Qualifying the On-chip Oscillators

As a preliminary study, we qualify the on-chip oscillators used by SCμM for communicating. We look at the frequency resulting from each frequency settings at room temperature (Section III-A), and how that frequency changes with temperature (Section III-B).

Fig. 2 is a block diagram of the oscillators, counters and related radio components. When transmitting, the rate at which the chips/bits leave the radio is clocked by the 2 MHz RC oscillator. The frequency of the carrier is set by the 2.4 GHz LC oscillator. As detailed in Section III-A, both 2 MHz RC oscillator and 2.4 GHz and LC oscillator are tuned through independent sets of 3 5-bit registers, called `coarse`, `mid`, `fine`. The 2M counter and LC counter are clocked by the two oscillators as a way to measure their frequency. When receiving, the incoming signal from the antenna is first demodulated by the I and Q phase match filter and form the "I" and "Q" channels. The number of zero-crossings on the "I" channel during a 100 μs window is a way to measure the value of the intermediate frequency. The IF is generated through a mixer as a difference between the incoming signal and the LC oscillator. The low intermediate frequency is used for monitoring the variation of the clock frequency, instead of using the 2.4 GHz high frequency. With the IF counter, SCμM can tell the frequency offset of its 2.4 GHz oscillator and adjust the oscillator accordingly as detailed in Sec. V-D.

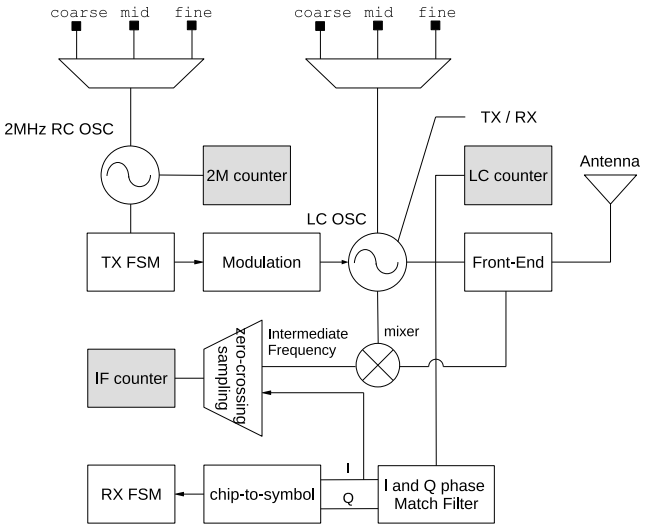


Fig. 2. The interconnection between the on-chip oscillators, counters and the related radio components of SCμM.

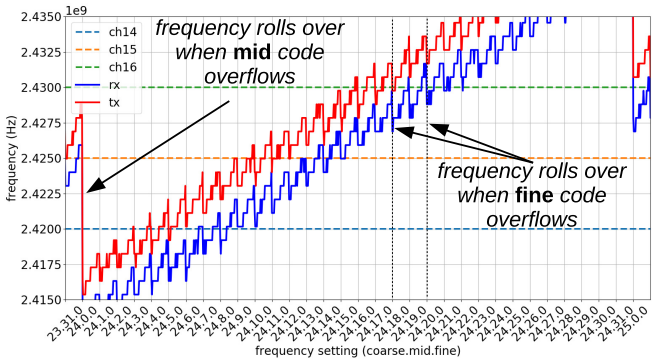


Fig. 3. The LC frequency as a function of the frequency setting, for values from `(23.31.0)` to `(25.0.31)`. While the frequency increases with the frequency setting, the frequency drops when the `fine` or `mid` codes overflow, resulting in a non-linear saw-tooth shape.

A. Controlling Frequency using DAC Settings

The frequencies of the 2 MHz RC oscillator and the 2.4 GHz LC oscillator are controlled by a similar mechanism. For each, the software running on SCμM controls three DACs through three 5-bit DAC registers, resulting in 32,768 frequency settings. The DAC is resistive in the case of the 2 MHz RC oscillator, capacitive in the case of the 2.4 GHz LC oscillator. We call the DAC registers `coarse`, `mid` and `fine`. We represent a setting as `(<coarse>.<mid>.<fine>)`, where each element is an integer number in the [0...31] range.

SCμM comes with the necessary digital blocks for the software to be able to measure the frequency of the oscillators. Each oscillator is equipped with a counter which increments each time the oscillator ticks (possibly through a divider). Software running on SCμM's Cortex-M0 can read the value of the counter, and reset it. This allows the software to arm a timer that fires periodically, and read the value of the counter each time it fires.

For example, the 2.4 GHz LC oscillator clocks its counter through a 960$\times$ divider: the counter increments every $\frac{960}{2.4\times10^9}$ s. This means that, after 1 ms, the counter has incremented by 2,500 ticks. Assuming the 1 ms period is timed accurately, reading a counter value different from 2,500 ticks indicates the LC oscillator is running fast or slow. Similarly, the 2 MHz RC oscillator when used without divider causes its counter to increment by 2,000 ticks every 1 ms.

We use SCμM's 64 MHz RC oscillator to measure the 1 ms window. While that oscillator is not ideal (non-zero jitter), it is valid to use it to compare the measured frequency of the 2 MHz RC oscillator against the measured frequency of the 2.4 GHz LC oscillator.

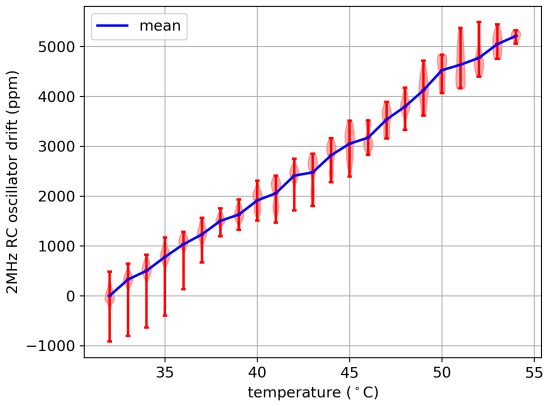
The mechanism above allows us to generate Fig. 3, in which we measure the frequency of the LC oscillator across all frequency settings. Fig. 3 shows a portion of that data, from settings `(23.31.0)` to `(25.0.31)`. Because of the way the radio is designed, there is an offset in the frequency of the LC oscillator depending on whether the radio is configured to transmit or receive. For a given `mid` code setting, changing the `fine` code from 0 to 31 causes the frequency to change by approx. 2.5 MHz, roughly 80 kHz per fine code.

As shown in Fig. 3, when the `mid` or `fine` codes of the frequency setting roll over from 31 to 0, the frequency drops. This is designed intentionally to have overlapping codes, and avoid having frequency "gaps" the LC oscillator cannot oscillate at. It does, however, make calibration harder, as the frequency does not monotonically increase with frequency setting. The 2 MHz RC oscillator exhibits the exact same behavior.

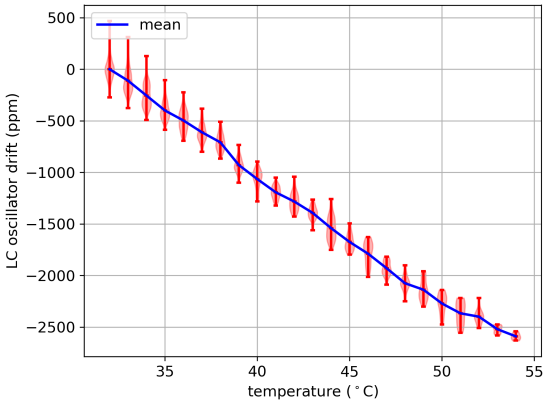
B. Impact of Temperature on Frequency

We fix the frequency setting for the two oscillators and measure how their frequency evolves over a 30°C to 55°C temperature range. We use a hair dryer to heat up the SCμM chip. SCμM reports the measured frequencies of its 2 MHz RC oscillator and 2.4 GHz LC oscillator over its serial interface. To measure the temperature at the SCμM chip, we place an OpenMote [18] next to it. The OpenMote reports the temperature from its Silicon Labs si70x temperature sensor every 500 ms over its serial interface. We use a laptop to record timestamped streams of information from the SCμM chip and OpenMote board. Although a temperature chamber would certainly be a more controlled environment, we believe that our setup is perfectly suitable to show general trends. Without loss of generality, we consider the frequency at 30°C as the reference, and compute drift values against that.

Fig. 4 presents the results for a static 2 MHz RC oscillator setting of `(22.17.15)`, and a static 2.4 GHz LC oscillator setting of `(24.10.22)` for receiving (RX). The drift of both oscillators is roughly linear over temperature. Over a 30-55°C range, the 2 MHz RC oscillator drifts by 5,000 ppm, the 2.4 GHz LC oscillator drifts by 2,500 ppm in RX modes. OpenMote, being compliant to IEEE802.15.4, tolerates a $\pm$1,000 ppm drift on the chip rate. A 2,500 ppm drift of the LC oscillator corresponds 6 MHz at 2.4 GHz, more than the 2.5 MHz tuning the `fine` code from 0 to 31 offers. Any tuning algorithm will therefore need to control at least the `fine` and `mid` codes.



(a) 2 MHz RC oscillator



(b) 2.4 GHz LC oscillator (RX)

Fig. 4. Evolution of the frequency of the 2 MHz RC oscillator and the 2.4 GHz LC oscillator over a 30-55°C temperature range, when tuned statically to `(22.17.15)` and `(24.10.22)`, respectively.

Figure 4 shows that, for SCμM to keep communicating with OpenMote over a temperature change of 10°C, SCμM needs to continuously calibrate both its 2 MHz RC oscillator and 2.4 GHz LC oscillator. The IEEE802.15.4 standard mandates a maximum drift of 40 ppm on frequency and 1,000 ppm on chip rate, translating into maximum drift targets for the 2 MHz RC oscillator and 2.4 GHz LC oscillator, respectively.

IV. Problem Statement and Intuition

To demonstrate the problem, we conduct an experiment with one SCμM chip communicating with one OpenMote. We pre-calibrate SCμM so it knows the frequency settings for successful exchange frames with an OpenMote, using a frequency sweeping mechanism borrowed from [19]. OpenMote also measure temperature using its on-board temperature sensor.

We start the experiment by having SCμM and OpenMote (successfully) exchanging frame at room temperature. This is the initial portion of Fig. 5, which shows both the frequency setting of the LC oscillator (which is constant as no continuous calibration is done), and the temperature. Note that, because of how the experiment is set up, the temperature stops being reported when communication between SCμM and OpenMote

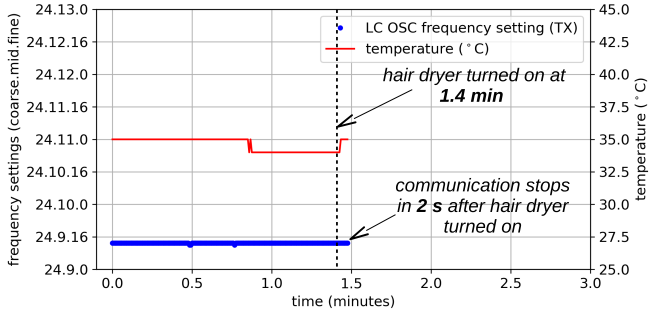


Fig. 5. Showing communication fail when SCμM does not continuously calibrate its oscillators. We pre-calibrate SCμM to it is able to communicate with an OpenMote. Each (blue) dot indicate successful communication. After about a minutes, when turn on a hair dryer pointed at SCμM. Less than 2 s after turning on the hair dryer, communication stops. As an artefact of how the experiment is done, the temperature stops being reported when communication stops.

stops. After about a minute, we turn on an hair dryer pointed at SCμM and OpenMote. Communication stops 2 s after the hair dryer is turned on.

The goal of continuous calibration is to keep communication between SCμM and OpenMote possible even when the temperature changes. To achieve the goal, we need to measure the frequency error of the 2 MHz RC oscillator and the 2.4 GHz LC oscillator so the error can be eliminated by tuning the settings.

V. CoCa: Continuous Calibration

Continuous Calibration (CoCa) is an overall system which allows SCμM and OpenMote to communicate when temperature changes. It consists of custom firmware running on both motes, and implementing a custom communication protocol. The purpose of this paper is to verify that CoCa works; as highlighted in Section VII, our current work in integrating CoCa as a (small) part of a complete standards-compliant protocol stack.

A. Frames Formats

CoCa uses three types of frames: `Beacon`, `Probe` and `ACK`. All frames are 10 bytes long and are composed of a 4-byte preamble, a 1-byte length field, 3 bytes of payload, and a 2-byte Frame Check Sequence (FCS).

The 1-byte length field is set to 5 and used to filter the frames of CoCa. We are certain there is no other 5-bytes long IEEE802.15.4 frame in same experiment field. The payload field of the `Beacon` frame consists of a 1-byte field with the number of seconds remaining in the `Beacon` burst (explained below), and two dummy bytes. The payload field of the `Probe` frame consists of the characters "C" and "F" (for "crystal-free"), and a 1-byte field with the number of milliseconds the OpenMote should wait before sends an `ACK` frame. The payload field of the `ACK` frame consists of a 2-byte field with the value of the temperature sensor, and a 1-byte field with the frequency offset of the received `Probe` frame (read from the CC2538's `FREQEST` register). Because of the

fact that the value of the payload allows one to distinguish a `Beacon`, a `Probe` and an `ACK` frame, there is no explicit "frame type" field.

B. Step 1. Optical Calibration

SCμM features a built-in photo-diode and the necessary circuitry for optical programming. Without needing to connect any wires (which would break the "single chip" nature of SCμM), one programs a SCμM chip by blinking an LED above it. The blinking pattern is very specific, and consists of a command word, followed by the bits in the image to program. Once the firmware is loaded, SCμM resets and starts executing that image. When executing, that firmware receives interrupts each time the photo-diode transits from low to high.

We use that last mechanism to provide the very first "coarse" calibration of the 2 MHz RC oscillator and the 2.4 GHz LC oscillator. Once optical programming is finished, we modify the firmware on the external programming board to keep switching the programming LED on/off 20 times at a fixed 100 ms interval. This blinking pattern generates interrupts on SCμM. We write the SCμM firmware to read and reset the counters of the 2 MHz RC oscillator and the 2.4 GHz LC oscillator at each such interrupt. We use that information to calibrate both oscillators, based on the knowledge that the blinking period is exactly 100 ms. The resulting calibration of the 2 MHz RC oscillator is within the 1,000 ppm target, as long as temperature doesn't change. The calibration of the 2.4 GHz LC oscillator is, however, not within the the 40 ppm target, and therefore does not allow SCμM to communicate with an OpenMote. The next step, the "frequency sweep" (Section V-C), is needed to calibrate the 2.4 GHz LC oscillator.

The optical programming can be done in scale by blinking a LED lamp with large power. So a large amount of SCμM chips with their optical receivers towards to the lamp can receive the light at the same time and be programmed. In the current step-up, we are focusing on single SCμM chip only.

C. Step 2. Frequency Sweep

The goal of the frequency sweep is letting SCμM learn the frequency settings of its 2.4 GHz LC oscillator that allow it to communicate (both TX and RX) with an off-the-shelf IEEE802.15.4 radio such as the OpenMote. Fig. 6 depicts the frequency sweeping step. It is similar to the approach taken by [20], but calibrating only a single channel frequency.

This step starts by having the OpenMote send a bust of `Beacon` frames on one frequency, one frame every 400 μs, for 20 s. When the burst ends, the OpenMote keeps listening for `Probe` frames from SCμM. If it receives one, it sends back an `ACK` frame after the duration indicated in the `Probe` frame.

SCμM starts by configuring its radio to receive mode, and sweeps its LC frequency setting from `(23.31.31)` to `(24.31.31)`, listening for `Beacon` frames. This range of settings is known to contain the frequency at which the `Beacon` frames are sent. Listening for `Beacon` frames at one frequency setting takes about 18 ms: 17 ms for stabilizing the LC oscillator, 1 ms of active listening time. It takes SCμM 18.43 s to sweep through the 1,024 settings. We manually trigger the OpenMote to send the `Beacon` frames at the same time as we have SCμM enter the sweeping step, allowing the sweep to finish by the time the burst of `Beacon` frames ends. SCμM records the best setting on which it received `Beacon` frames as its RX frequency setting.

SCμM sweeps through its setting a second time as soon as the burst of `Beacon` frames ends. This time, for each frequency setting, SCμM transmits a `Probe` frame using that setting, then uses its RX frequency setting to listen for an `ACK` frame. It takes SCμM 3.12 ms to send one `Probe` frame: 2.8 ms for stabilizing the LC oscillator, 320 μs for transmitting the frame. We arbitrarily use value 30 ms for the delay between the OpenMote receiving a `Probe` frame and transmitting the corresponding `ACK` frame. Including the 1 ms idle listening duration, it takes SCμM 34.12 ms between two adjacent `Probe` transmissions. Sweeping the 1,024 settings takes SCμM 34.94 seconds. SCμM records the best setting on which it transmitted a `Probe` frame and for which it received an `ACK` for, as its TX frequency setting.

For TX setting, the "best" is indicated by the frequency offset (FO). If the frequency setting used for transmitting packet returns an ACK with smallest FO , that frequency setting is selected as the "best" setting for TX. For RX setting, the "best" is selected from groups of settings which can receive beacons. Those settings are then grouped according to there `coarse` and `mid` settings. The median of the group with the most settings is selected as the "best" setting for RX. For more details, please refer to Section IV, Algorithm 1 and 2 in [20].

At the end of the frequency sweep step, SCμM has determined its 2.4 GHz LC frequency settings for transmitting and receiving with an OpenMote, on one channel frequency.

D. Step 3a. Frequency Trimming

SCμM uses the frequency settings from step 2 (Section V-C) to send `Probe` frames and listen for `ACK` frames, every 50 ms. Fig. 7 illustrates frequency trimming, which allows SCμM to continuously calibrate all oscillators involved with wireless communication.

To adjust its 2.4 GHz LC oscillator frequency setting for TX, SCμM reads the frequency offset field from the `ACK` frame. The OpenMote is programmed to indicate in that field the frequency offset of the `Probe` frame (which it gets from its `FREQUEST` register). It then applies (1) to change is `fine` code setting, where FO is the frequency offset indicated by the CC2538 in the `ACK` frame, which has a null value (no offset) of 2. One tick in the `fine` code setting of the 2.4 GHz LC oscillator causes a frequency shift of 80 kHz. The resolution of FO is 7.8 kHz: a value of 10 roughly corresponds 80 kHz, i.e. one tick in the `fine` code setting.

$$LC\_TX_{fine} -= (signed)(FO - 2)/10 \quad (1)$$

To adjust its 2.4 GHz LC oscillator frequency setting for RX, SCμM counts the zero-crossing of the IF, as detailed in Section I. It then applies (2) to change is `fine` code setting. Here, IF_count is the number of zero-crossings of the IF, 500 is the target value. Each additional IF zero-crossing translates

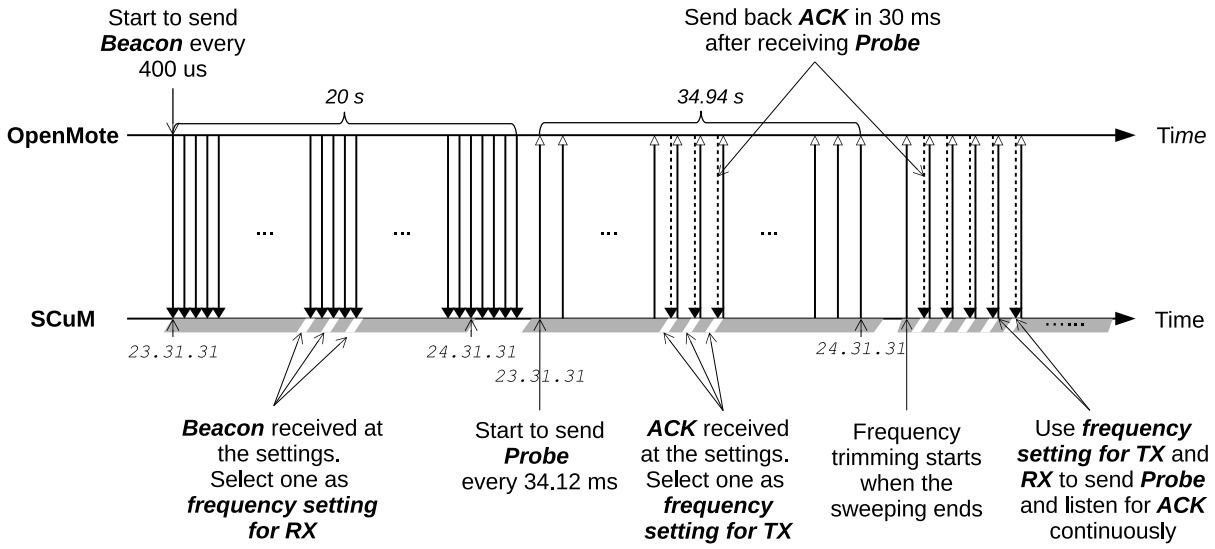


Fig. 6. The frequency sweep step of CoCa. The OpenMote sends a burst of `Beacon` frames on a single frequency, one every 400 μs. SCμM sweeps its 2.4 GHz LC frequency settings from `(23.31.31)` to `(24.31.31)`, listening for `Beacon` frames. It records the best setting on which it receives `Beacon` frames as its RX frequency setting. After sending the burst of `Beacon` frames, the OpenMote continuously listens for `Probe` frames; it sends back an `ACK` frame if it receives one. SCμM does a second sweep through its frequency settings, this time sending a `Probe` frame for each setting and listening for an `ACK` frame. It records the "best setting" (detailed in [20]) on which it transmitted a `Probe` frame it received an `ACK` for, as its frequency setting for transmitting (TX). After this, the frequency trimming step (Section V-D) starts.

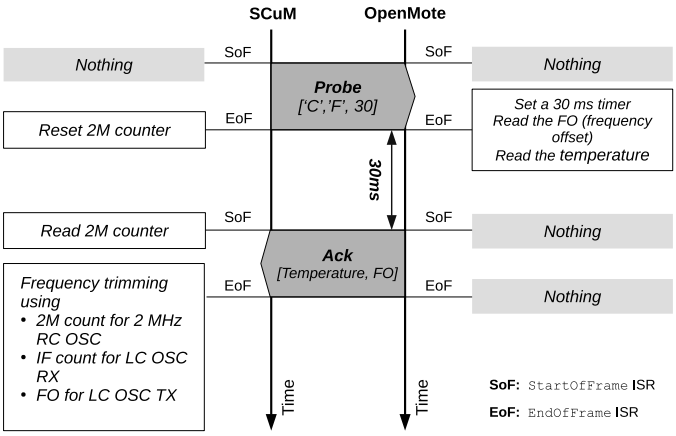


Fig. 7. SCμM resets its 2 MHz RC counter when it just finished transmitting the `Probe` frame. Upon receiving a `Probe` frame, OpenMote records its frequency offset (read from its `FREQEST` register). It then waits exactly 30 ms, before sending an `ACK` frame containing the frequency offset. Upon receiving `ACK` frame, SCμM reads the value of its 2 MHz RC counter to calibrate it 2 MHz RC oscillator setting, uses the frequency offset value to calibrate its 2.4 GHz LC oscillator setting in TX mode, and measures the number of zero-crossing to calibrate its 2.4 GHz LC oscillator setting in RX mode.

to a 5 kHz frequency shift of the 2.4 GHz LC oscillator; 16 zero-crossings hence correspond to 80 kHz.

$$LC\_RX_{fine} += (signed)(IF\_count - 500)/16 \quad (2)$$

To adjust its 2 MHz RC oscillator frequency setting, SCμM measures the time between the end of the `Probe` frame and the beginning of the `ACK` frame with that oscillator. It knows that this duration is 30 ms. It then applies (3) to possible change its `mid` code setting. Here, $2M\_count$ is the value of the 2 MHz counter at the start of the `ACK` frame, 60000 is its target value corresponding to 30 ms. Increments of the `mid` code of 2 MHz RC oscillator frequency setting corresponds to a frequency shift of 1.93 kHz; The resolution of 2M count is 33.33 Hz: a value of 58 roughly corresponds to 1.93 kHz.

$$2M\_RC_{mid} += (signed)(2M\_count - 60000)/58 \quad (3)$$

To limit the impact of jitter in the oscillators, we use values for the measured oscillator frequencies averaged over 10 samples,

Table I summarizes CoCa: the oscillators it calibrates, the corresponding error measurements, the setting it tunes, and the equation used to calibrate.

We do not calibrate the 64 MHz RC oscillator over temperature during CoCa procedures. However, it only has minor influence to the accuracy of LC frequency for RX. Assuming a drift of 5,000 ppm on 64 MHz RC oscillator due to the rapid temperature changes, this results an error of 0.5 us over 100 us when counting the IF zero crossing. It introduces 2.5 IF count error, corresponding to 12 kHz error (5 ppm) in 2.4 GHz LC oscillator. In a mild temperature changing environment, the error for LC oscillator of RX caused by the error of 64 MHz RC oscillator can be ignored.

E. Step 3b. "Hopping" between Frequency Settings

The frequency of the 2.4 GHz LC oscillator increases with the `fine` code of the frequency setting. It is often necessary to tune the frequency beyond the narrow tuning range the `fine` code offers. The challenge is that, as soon as the `mid` code increments and the `fine` code rolls over, the frequency decrements by 3-4 MHz, causing the SCμM chip to loose communication with the OpenMote. This has been detailed in Section III-A.

TABLE I
THE OSCILLATOR, ERROR MEASUREMENTS CALIBRATION AND EQUATION USED IN THE FREQUENCY TRIMMING STEP OF COCA.

Oscillator	Error Measurement	Calibration Step	Calibration Equation
2 MHz RC Osc.	2 MHz counter value	`mid`	Eq. (3)
2.4 GHz LC Osc. (TX)	frequency offset (FO)	`fine`	Eq. (1)
2.4 GHz LC Osc. (RX)	IF zero-crossings count	`fine`	Eq. (2)

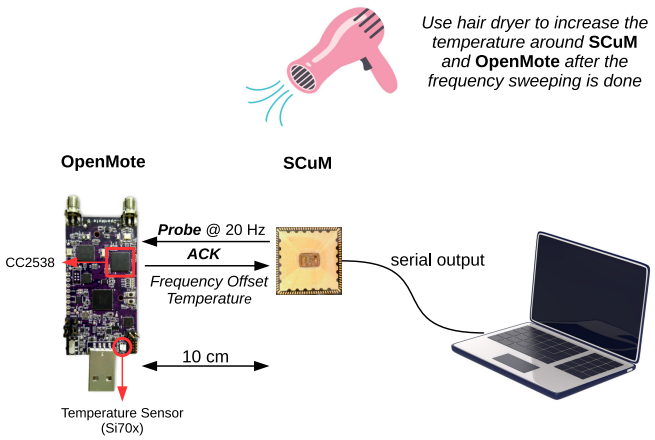


Fig. 8. SCμM sends `Probe` frames to OpenMote, one every 50 ms. The OpenMote sends back an `ACK` frame that contains the frequency offset and the temperature. We use a hairdryer to increase the temperature of SCμM. We place an OpenMote board only 10 cm away from SCμM, and point the hair dryer at both. This allows us to measure the temperature at SCμM.

To address this roll-over issue, we propose a technique called "hopping" in which SCμM tracks the LC frequency using two frequency settings with different `mid` codes. Fig. 9 shows an example where, at time $T1$, those settings are `(24.8.22)` and `(24.9.16)`. We know that these settings result in frequencies close enough that they both function, i.e. the radio is able to receive frames using both. The SCμM firmware alternates between the two settings, "hopping" from one to another at each received frame. Frequency trimming is done on both, independently: the values of both settings evolve as temperature changes.

We blacklist the 5 frequency settings around the point where the `mid` code increments, i.e. any frequency setting between `coarse.mid.30` and `coarse.mid+1.2`. We want to avoid these regions as they are highly non-linear. When one of the two frequency settings falls in the blacklisted settings, its value is replaced by a new frequency settings 24 `fine` codes away from the second setting. The value 24 is determined experimentally. The exact same approach is used on the 2.4 GHz LC oscillator TX setting.

The goal of the blacklist is to notify SCμM to "hop" to another frequency setting in advance before the `fine` code rolls over. A larger frequency setting blacklist increases the safe window size to be notified. There are two factors affect the size of the setting blacklist. A larger frequency setting blacklist is required when: the temperature changing rate increases, or the `Probe` sending rate decreases. The quantitative analysis of the blacklist size with the temperature changing rate and the `Probe` setting rate is out of scope of this paper.

In case a rapid temperature changing causes CoCa loses its communicating frequency, the frequency sweep presented previously can be triggered to restart the calibration process. At the end of frequency sweep, the frequency trimming procedure starts again.

F. A Summary of CoCa

For summarizing, CoCa relies on 3 types of frames to be exchanged between SCμM and OpenMote (Section V-A). CoCa starts by a vert coarse calibration of the oscillator as part of the optical bootloading process (Section V-B). By sweeping through frequency settings, CoCa further fine-tunes the frequency of the 2.4 GHz LC oscillator, allowing SCμM and OpenMote to communicate at a constant temperature (Section V-C). From then on, when sending frames and receiving acknowledgements from an OpenMote, SCμM keeps track of the frequency of the 2 MHz RC oscillator, the IF, and the frequency offset (FO) in the payload of the acknowledgement. It uses this information to continuously adjust the frequency settings, a process we call "frequency trimming" (Section V-D). The CoCa calibration procedure requires to be done every time when SCμM boots after the optical programming.

G. Energy Consumption

SCμM operates at 1.5 V. It consumes 847 μW while transmitting, with an output power of -10 dBm, and 1.03 mW while receiving, with a sensitivity of -83 dBm [1]. In Step 2 (Sec. V-C), intensive `Beacon`, `Probe` and `ACK` frames are received/sent over 2,048 settings and 1,024 settings for calibrating one channel frequency of RX and TX. For each frame to send or receive consumes about 0.35 μC. Hence the whole frequency sweep process consumes $0.35 \times 2048 + (0.35 + 0.35) \times 1024$, about 1.43 mC. For a typical coin cell battery holding 225 mAh of capacity, this step consumes 0.000176% and it's done once per life time. In Step 3a (Sec. V-D), `Probe` and `ACK` are sent and received to track the frequency error every one second. For a 225 mAh battery, this step consumes 0.000864% of its capacity. With a faster sending rate of `Probe`, the consumption increases. However, one `Probe` per seconds has been proved that it can handle a temperature changing rate of 3°C/min, which is much faster comparing to daily temperature changes.

These calculation is estimated in ideal case that the radio can be turned off and turned on immediately when it's needed. As indicated in Sec. V-D, a 30 ms round time requires for SCμM to keep the radio on, which consumes a major mount of energy. To the state of the work, SCμM requires mili-seconds duration to stabilize its oscillator to have radio functioning. Also there is a 150-200μA leaking current from SRAM and

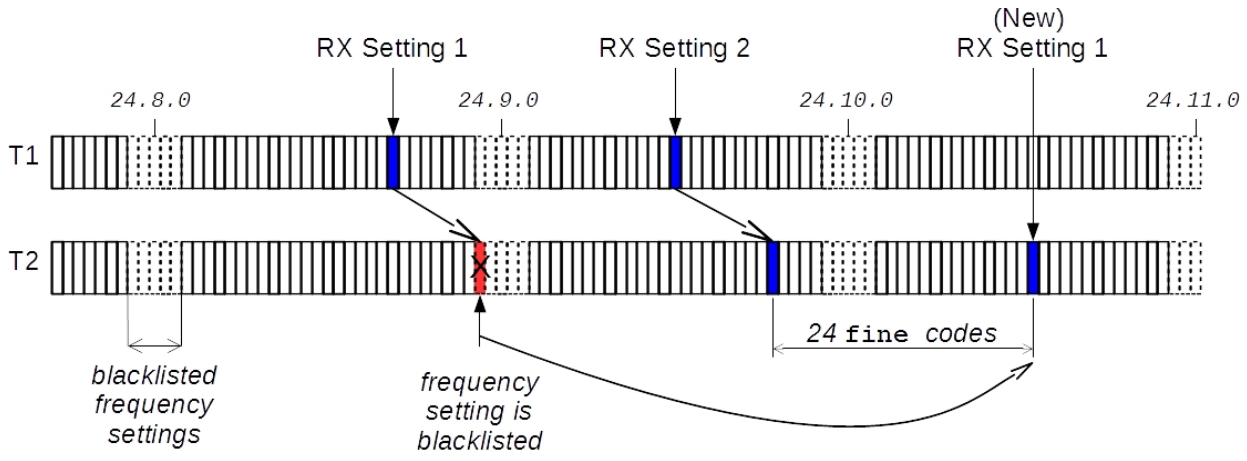


Fig. 9. We blacklist all frequency settings in ranges `(coarse.mid.30)` and `(coarse.mid+1.2)`. At time $T1$, the radio hops between RX frequency settings `(24.8.22)` and `(24.9.16)`. Frequency trimming is used on both independently; at time $T2$, trimming causes the the settings to evolve to `(24.8.30)` and `(24.9.25)`. Since setting `(24.8.30)` falls in the blacklisted settings, it is replaced by a setting 24 `fine` code above the other setting, i.e. `(24.10.17)`.

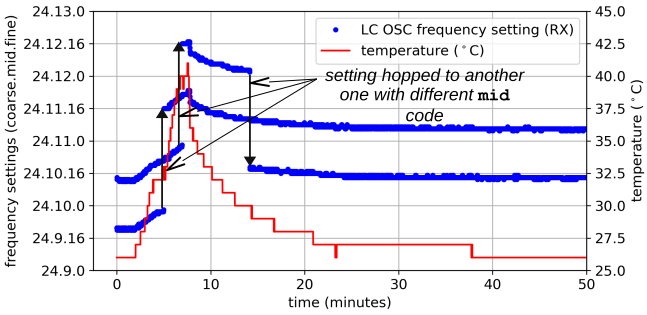


Fig. 10. Evolution of the frequency setting of the 2.4 GHz LC oscillator in RX as the SCμM chip is heated up using a hair dryer, then cools down.

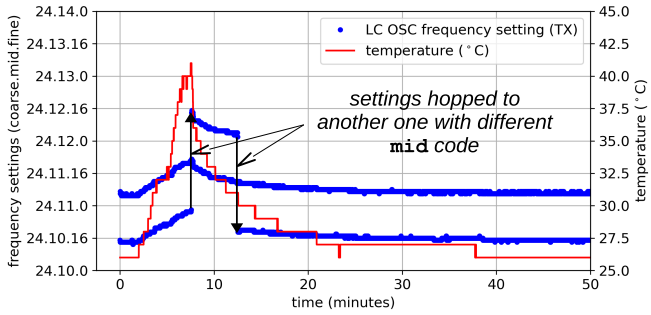


Fig. 11. Evolution of the frequency setting of the 2.4 GHz LC oscillator in TX as the SCμM chip is heated up using a hair dryer, then cools down.

analog circuits which can't be shutoff in this revision. Those issues need to be resolved in the future revision of SCμM.

VI. Evaluation

We start by introducing the experimental setup used to evaluate the performance of CoCa (Section VI-A). Section VI-B discusses the experimental results which show that CoCa allows SCμM to keep communicating with an OpenMote even under the extreme condition of it being heated by a hair dryer.

A. Experimental Setup

The experimental setup is depicted in Fig. 8 and consists of a SCμM chip, an OpenMote board, and a laptop for data recording. SCμM is programmed with CoCa[1].

We use a hair dryer to increase the temperature of SCμM. We place the OpenMote only 10 cm away from SCμM and aim the hair dryer at both motes so they have roughly the same temperature. The OpenMote measures and reports temperature. When the hair dryer is switched on, the temperature of the motes rises from 26°C to 41°C in 5 min, a 3°C/min temperature ramp.

[1] As an online addition to this paper, the source code of CoCa is published under a BSD open-source license at https://github.com/PisterLab/scum-test-code

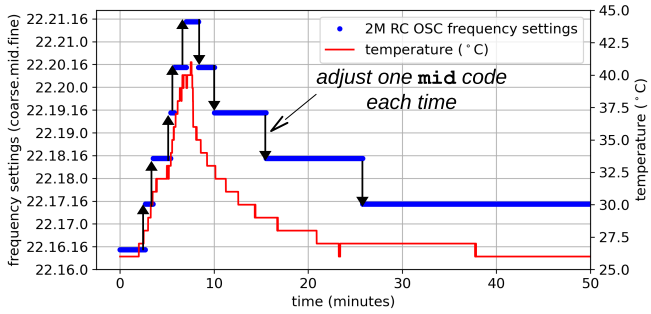


Fig. 12. Frequency setting of the 2 MHz RC oscillator as temperature changes.

We connect a laptop to the serial interface of SCμM and have SCμM report the frequency offset (FO) and temperature values received from the OpenMote, and the frequencies measured of the 2 MHz RC oscillator and 2.4 GHz LC oscillator, and their current frequency settings.

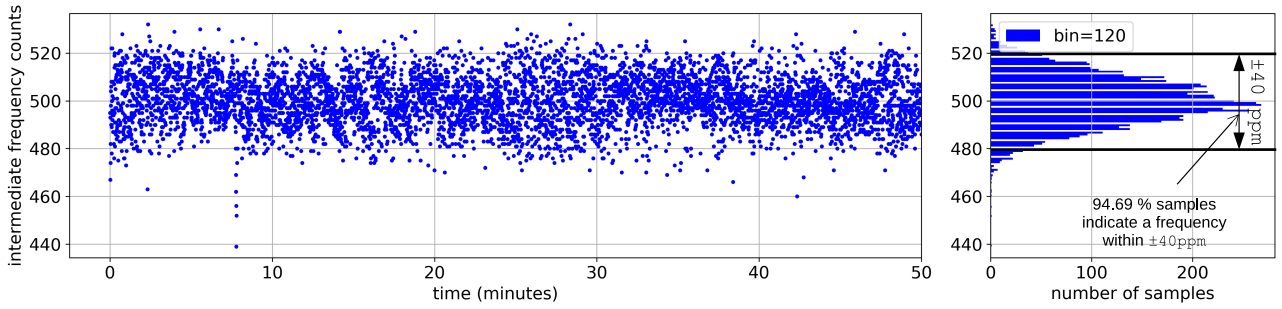


Fig. 13. The number of IF zero-crossings over time (*left*) and the corresponding distribution (*right*). 94.69% of the measurements fall within the ±40 ppm range mandated by the IEEE802.15.4 standard.

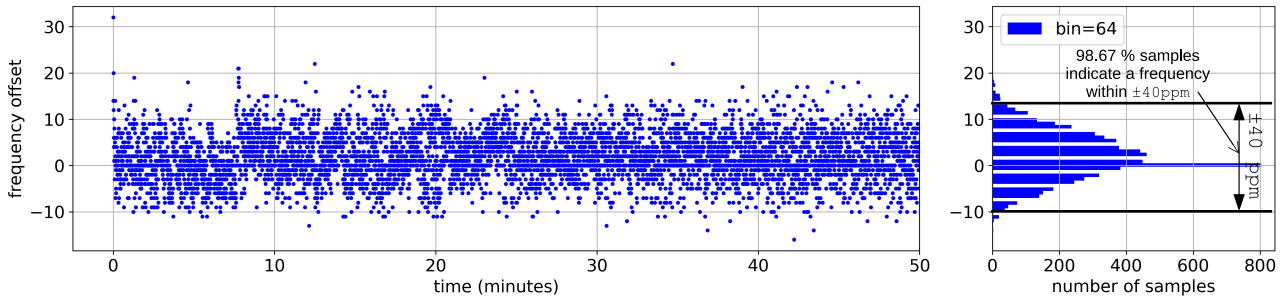


Fig. 14. The frequency offset (FO) evolving over time (*left*) and the corresponding distribution (*right*). 98.67% of the measurements fall within the ±40 ppm range mandated by the IEEE802.15.4 standard.

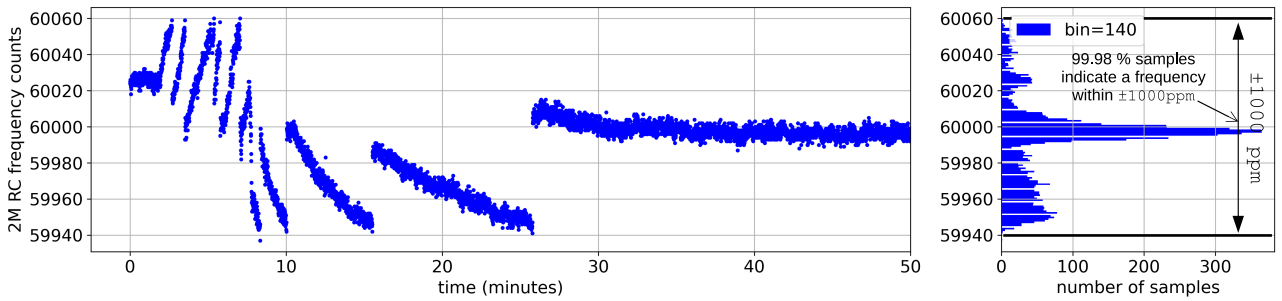


Fig. 15. The 2 MHz count evolving over time (*left*) and the corresponding distribution (*right*). 99.98% of the measurements fall within the ±1,000 ppm range mandated by the IEEE802.15.4 standard.

B. Experimental Results

Fig. 10 shows the frequency setting of the 2.4 GHz LC oscillator in RX changing with temperature. Each blue dot corresponds to an `ACK` frame received from the OpenMote. Because SCμM hops between frequency settings, these dot form two parallel "lines". The red line represents temperature. At minutes 5, 7 and 14, one of the frequency settings enters the blacklisted settings and is shifted.

Fig. 11 is similar to Fig. 10, showing the frequency setting of the 2.4 GHz LC oscillator in TX. Here again, frequency settings are shifted at minutes 8 and 13.

Fig. 12 shows frequency setting of the 2 MHz RC oscillator as temperature changes. Each time the frequency of the 2 MHz RC oscillator needs to adjust, one `mid` code of its frequency setting is incremented or decremented.

We evaluate the performance of CoCa by gathering the frequency offsets of the 2 MHz RC oscillator and the 2.4 GHz LC oscillator, measured by the techniques details in Section I. The left side of Fig. 13 shows the number of IF zero-crossings over time; the right plots the corresponding distribution. An offset of 19 from the ideal IF zero-crossings value of 500 corresponds to approx. 40 ppm, the target drift mandated by IEEE802.15.4. Fig. 13 shows that 94.69% of the IF zero-crossings fall within that range.

The left side of Fig. 14 shows the frequency offset (FO) over time; the right plots the corresponding distribution. An offset of 12 corresponds to the target ±40 ppm maximum drift. The distribution shows that 98.67% of the measurements fall within that range.

The left side of of Fig. 15 shows the 2 MHz count over time; the right plots the corresponding distribution. An offset

of 60 corresponds to the target ±1,000 ppm maximum drift. The distribution shows that 99.98% of the measurements fall within that range.

Though the experiment is conducted under an environ-ment within 30-55°C temperature range, there should be no constrain to apply "CoCa" over a commercial temperature range, i.e. -40-85°C. Actually, "CoCa" is more sensitive to the temperature changing rate rather than changing range. To handle a faster temperature changing rate situation, the faster `Probe` sending rate is needed to notify on-time when the setting reaches to the setting blacklist area. So that SCμM can "hop" to another frequency setting in advance before losing the frequency. For a daily temperature changing rate which is much slower than 3°C per minute, SCμM can keep its radio on the right frequency with "CoCa" without constrain.

VII. Conclusions

This paper introduces a technique called "CoCa" allowing SCμM to continuously calibrate its 2 MHz RC oscillator and 2.4 GHz LC oscillator. We first obtain an initial calibration of the 2 MHz RC oscillator by having the programming board blink its LED periodically after the optical bootloading process. By sweeping the frequency settings of the 2.4 GHz LC oscillator twice, we allow SCμM to communicate with an OpenMote, albeit at a single frequency and a constant temperature. By continuously measuring the frequencies of the oscillator, and getting feedback from the OpenMote of the offset of the SCμM transmit frequency, SCμM continuously trims the frequency as temperature changes. To cope with the saw-tooth shape of the frequency profile, SCμM hops between two frequency settings, and shifts two intermittently to avoid highly non-linear region of the frequency settings. The results shows that the 2 MHz RC oscillator stays within the ±1,000 ppm range mandated by the IEEE802.15.4 standards 99.98% of the time, and that the 2.4 GHz LC oscillator stays within the ±40 ppm range 98.67% and 94.69% of the time (for TX and RX, respectively.).

The implication of results are significant. CoCa allows a mote with a grain-of-rice size to participate in an IEEE802.15.4 network, relying exclusively on its on-chip oscillator. This fact brings up one step closer to the "Smart Dust" vision.

Of course, this paper focuses on verifying an approach such as CoCa works. The system doesn't attempt doing more than just this continuous calibration. The long-term goal for this work is to have SCμM run a full protocol stack such as the 6TiSCH [21] fully standards-based Industrial IoT protocol stack. we are currently working on integrating CoCa into the 6TiSCH protocol and its implementation on SCμM. This includes standardizing a new Information Element, so the frequency offset can be carried in an `ACK` frame in a standards-compliant manner.

References

[1] F. Maksimovic, B. Wheeler, D. C. Burnett, O. Khan, S. Mesri, I. Suciu, L. Lee, A. Moreno, A. Sundararajan, B. Zhou, R. Zoll, A. Ng, T. Chang, X. Vilajosana, T. Watteyne, A. Niknejad, and K. S. J. Pister, "A crystal-free single-chip micro mote with integrated 802.15.4 compatible transceiver, sub-mW BLE compatible beacon transmitter, and Cortex M0," in *Symposium on VLSI Circuits*. Kyoto, Japan: IEEE, 9-14 June 2019.

[2] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Computer Society Std., 22 April 2016.

[3] L. Wu, X. Du, M. Guizani, and A. Mohamed, "Access control schemes for implantable medical devices: a survey," *IEEE Internet Things J.*, vol. 4, pp. 1272–1283, 25 May 2017.

[4] S. Seneviratne, Y. Hu, T. Nguyen, G. Lan, S. Khalifa, K. Thilakarathna, M. Hassan, and A. Seneviratne, "A survey of wearable devices and challenges," *IEEE Commun. Surveys Tuts.*, vol. 19, pp. 2573–2620, 26 July 2017.

[5] S. Movassaghi, M. Abolhasan, J. Lipman, D. Smith, and A. a. Jamalipour, "Wireless body area networks: a survey," *IEEE Commun. Surveys Tuts.*, vol. 16, pp. 1658–1686, 14 January 2014.

[6] B. Warneke, M. Last, B. Liebowitz, and K. S. Pister, "Smart dust: communicating with a cubic-millimeter computer," *Computer*, vol. 34, pp. 44–51, January 2001.

[7] D. C. Burnett, B. Wheeler, L. Lee, F. Maksimovic, A. Sundararajan, O. Khan, and K. S. J. Pister, "CMOS oscillators to satisfy 802.15.4 and Bluetooth LE PHY specifications without a crystal reference," in *Comput. Commun. Workshop Conf. (CCWC)*. Las Vegas, USA: IEEE, 7-9 January 2019, pp. 218–223.

[8] I. Suciu, F. Maksimovic, D. Burnett, O. Khan, B. Wheeler, A. Sundararajan, T. Watteyne, X. Vilajosana, and K. S. J. Pister, "Experimental clock calibration on a crystal-free mote-on-a-chip," in *Conf. on Comput. Commun. Workshops (INFOCOM), CNERT Workshop*. Paris, France: IEEE, 29 April-2 May 2019.

[9] B. Wheeler, "Low power, crystal-free design for monolithic receivers," Ph.D. dissertation, Dept. Electron. Eng. and Comput. Sci. (EECS), Univ. California, Berkeley., 14 May 2019.

[10] M. Ding, M. Song, E. Tiurin, S. Traferro, Y.-H. Liu, and C. Bachmann, "A 0.9pJ/cycle 8ppm/°C DFLL-based wakeup timer enabled by a time-domain trimming and an embedded temperature sensing," in *Symposium on VLSI Circuits*. virtual conference: IEEE, 17 June 2020.

[11] M. Xu, W. Xu, T. Han, and Z. Lin, "Energy-efficient time synchronization in wireless sensor networks via temperature-aware compensation," *ACM Trans. Sens. Netw.*, vol. 12, pp. 1–29, April 2016.

[12] D. Stanislowski, X. Vilajosana, W. Qin, and T. Watteyne, "Adaptive synchronization in IEEE802.15.4e networks," *IEEE Trans. Ind. Informat.*, vol. 10, pp. 795–802, 27 March 2013.

[13] B. Martinez, X. Vilajosana, Dujovne, and Diego, "Accurate clock discipline for long-term synchronization intervals," *IEEE Sensors J.*, vol. 17, pp. 2249–2258, 01 February 2017.

[14] T. Yuan, F. Maksimovic, D. Burnett, B. Wheeler, L. Lydia, and K. S. J. Pister, "Temperature calibration on a crystal-free mote," in *World Forum on Internet of Things (WF-IoT)*. New Orleans, LA, USA: IEEE, 13 October 2020.

[15] D.-S. Lee, S.-J. Kim, D. Kim, Y. Pu, S.-S. Yoo, M. Lee, K. C. Hwang, Y. Yang, and K.-Y. Lee, "A design of fast-settling, low-power 4.19-MHz real-time clock generator with temperature compensation and 15-dB noise reduction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, pp. 1151–1158, 5 March 2018.

[16] B. Wheeler, F. Maksimovic, N. Baniasadi, S. Mesri, O. Khan, D. Burnett, A. Niknejad, and K. S. J. Pister, "Crystal-free narrow-band radios for low-cost IoT," in *Radio Frequency Integrated Circuits Symposium (RFIC)*. Honolulu, USA: IEEE, 4-6 June 2017.

[17] I. Suciu, F. Maksimovic, B. Wheeler, D. C. Burrnett, O. Khan, T. Watteyne, X. Vilajosana, and K. S. J. Pister, "Dynamic channel calibration on a crystal-free mote-on-a-chip," *IEEE Access*, vol. 7, pp. 120 884–120 900, 26 August 2019.

[18] X. Vilajosana, P. Tuset, T. Watteyne, and K. S. J. Pister, "OpenMote: open-source prototyping platform for the industrial IoT," in *International Conf. on Ad Hoc Networks (AdHocHets)*. San Remo, Italy: Springer, 1-2 September 2015.

[19] T. Chang, T. Claeys, X. Mališa Vučinić, Vilajosana, T. Yuan, B. Wheeler, F. Maksimovic, D. Burnett, B. Kilberg, K. S. Pister, and T. Watteyne, "Industrial IoT with crystal-free mote-on-chip," in *Symposium on VLSI Circuits (VLSI)*. virtual conference: IEEE, 17 June 2020.

[20] T. Chang, T. Watteyne, F. Maksimovic, B. Wheeler, D. Burnett, T. Yuan, X. Vilajosana, and K. S. Pister, "QuickCal: assisted calibration for crystal-free micromotes," *IEEE Internet Things J.*, vol. 8, pp. 1846–1858, 11 August 2020.

[21] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: a tutorial," *IEEE Commun. Surveys Tuts.*, vol. 22, pp. 595–615, 4 September 2019.