QuickCal: Assisted Calibration for Crystal-Free Micro-Motes
Tengfei Chang, Thomas Watteyne, Filip Maksimovic, Brad Wheeler, David Burnett, Titan Yuan, Xavier Vilajosana, Kris Pister

To cite this version:

HAL Id: hal-02916810
https://inria.hal.science/hal-02916810
Submitted on 18 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
QuickCal: Assisted Calibration for Crystal-Free Micro-Motes

Tengfei Chang*, Thomas Watteyne* Senior, IEEE, Filip Maksimovic†, Brad Wheeler‡,
David C. Burnett†, Titan Yuan†, Xavier Vilajosana† Senior, IEEE, Kris Pister†
* EVA team, Inria, Paris, France
† BSAC, University of California, Berkeley, CA, USA
‡ Universitat Oberta de Catalunya, Barcelona, Spain

Abstract—The Single Chip Micro Mote (SCµM) is a crystal-free single-chip mote that brings us one step closer to the Smart Dust vision, in particular as it can communicate with off-the-shelf IEEE802.15.4 and Bluetooth Low Energy devices. However, before it can be part of such networks, the crystal-free SCµM chip needs to be able to accurately tune its communication frequency to synchronize to the network. This is a challenge since its on-board RC and LC-based resonating circuits have a drift rate that can be 3 orders of magnitude worse than crystal-based oscillators typically used in today’s radios. This article introduces QuickCal, a solution that allows a SCµM chip to self-calibrate against off-the-shelf devices dedicated to assisting with its calibration. We show that a SCµM chip can self-calibrate against this QuickCal Box in fewer than 3 min. We further validate that, once it has self-calibrated, a SCµM chip can reliably communicate with off-the-shelf IEEE802.15.4 devices. Finally, we demonstrate a heterogeneous network – composed of a SCµM chip and an OpenMote device – implementing a full 6TiSCH Industrial IoT protocol stack, which uses time synchronization and channel hopping. This is the first time that a crystal-free radio is participating in a channel-hopping enabled TSCH network.

Index Terms—Crystal-free, single-chip mote, SCµM, calibration, 6TiSCH.

I. INTRODUCTION

Low-power wireless technology is ubiquitous in today’s Industrial space with standards such as WirelessHART [1], ISA100.11a [2], and 6TiSCH [3]. All of these standards are based on Time Synchronized Channel Hopping (TSCH), a networking technique where nodes synchronize to lower their power consumption, and use channel hopping to increase reliability. Tens of thousands of TSCH networks are deployed today. The major Industrial IoT (IIoT) applications are asset performance monitoring [4], [5], logistic management [6], [7], [8], and real-time control-process [9], [10]. It is predicted that the industrial manufacturing sector will account for 33% of the total IoT applications within the next decade [11].

The Single Chip Micro Mote (SCµM) [13] is a 2×3×0.3 mm³ crystal-free mote-on-chip, which features an ARM Cortex-M0 core, a radio compliant with the IEEE802.15.4 [14] (the Physical layer that 6TiSCH stack builds on) and Bluetooth Low Energy standards, and an optical receiver for programming [12]. Once the firmware is loaded onto the SCµM chip, the chip can then be removed from its development board; all it needs to operate are a power source and an antenna.

an ARM Cortex-M0 core, a radio compliant with the IEEE802.15.4 [14] (the Physical layer that 6TiSCH stack builds on) and Bluetooth Low Energy standards, and an optical receiver for programming. SCµM is specifically designed for micro-robotic applications.

SCµM does not require any external components, except power and an antenna, to operate. In particular, it does not require an external clock source such as a crystal oscillator. Without a crystal, SCµM relies on 4 internal oscillating circuits for timekeeping: the “HF oscillator” (a 20 MHz RC oscillator to clock the Cortex-M0), the “2M 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 downside of such a crystal-free architecture is that the operating frequency of these internal oscillators can drift significantly, e.g., 16,000 ppm during a temperature change of 45 °C [15], much more than the 10-40 ppm typically seen in crystal oscillators. This drift depends on many elements

* Corresponding author: Tengfei Chang (email: tengfei.chang@inria.fr). Copyright (c) 20xx IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.
in addition to temperature, including electronic noise, supply voltage, and manufacturing process. Timing inaccuracies due to this high drift make it difficult for SCµM to communicate since the oscillators are used to set the communication frequency, time the modulation, and maintain general timekeeping. The IEEE802.15.4 standard [14], for example, mandates that all compliant radios use clocks that have a drift of less than 40 ppm.

High drift is a common issue in crystal-free SoC designs, including SCµM. This is because of the fact that the electronic components used in the crystal-free oscillators are affected by the elements mentioned above. In this article, we develop a calibration proposed using SCµM, but our results carry over directly to any other crystal-free SoC design.

In this article, we develop a technique we call “QuickCal”, allowing a crystal-free radio chip to self-calibrate and communicate with off-the-shelf IEEE802.15.4 radios. In this article, we apply and test QuickCal on the SCµM chip. QuickCal relies on a “QuickCal Box” being within range of the SCµM chip. This QuickCal Box contains 16 off-the-shelf IEEE802.15.4-compliant devices (we use the popular OpenMote platform [16]) running firmware specific for this calibration. The CC2538 system-on-chip of the OpenMote uses an external 32 MHz crystal as a (high-accuracy) external timing reference. Because of the requirement to have a QuickCal Box nearby, we qualify our approach as an “assisted” calibration.

The main oscillator running on SCµM is based on an LC voltage-controlled oscillator (VCO) architecture and uses three overlapping 5-bit capacitive digital-to-analog converters (DACS) to control its frequency. The state of these switch capacitors are mapped to registers, allowing the software to set any of the $2^{15} = 32,768$ frequency settings. The calibration process consists of discovering the frequency settings that allow SCµM to both transmit and receive IEEE 802.15.4 frames on each of the 16 frequencies of the 2.4 GHz band defined in the standard. Because of the architecture of SCµM’s radio, the setting for transmitting and for receiving at a given communication frequency is different. This means that the calibration process involves discovering 32 frequency settings.

Since the LC oscillator’s drift changes with temperature and supply voltage, the same 32 frequency settings cannot be used over temperature and supply voltage. A mechanism is required to trigger re-calibration of the 32 frequency settings, which is ongoing work.

As detailed in Section III-C, the proposed QuickCal solution relies on two behaviors. First, we program the devices in the QuickCal Box to send and receive frames on each of the 16 frequencies. Second, we have the SCµM chip sweep its frequency settings, recording the settings on which it can successfully transmit and receive frames to and from the QuickCal Box.

QuickCal is a pioneering effort to not only develop a solution for the time-reference calibration of micro-motes but also establish a methodology for the industrialization of the entire fabrication process, closing the gap between technological breakthroughs and industrial adoption. The contributions of this article are three-fold:

- We characterize SCµM’s 2.4 GHz LC oscillator, in particular the relationship between the frequency setting and the frequency.
- We design the QuickCal solution and associated protocol, which allows a SCµM chip to self-calibrate in under 3 min.
- We evaluate the quality of the resulting calibration in terms of Packet Delivery Ratio (PDR) of the communication between SCµM and an OpenMote and demonstrate a heterogeneous 6TiSCH network.

The remainder of this article is organized as follows. Section II summarizes related work on frequency calibration in wireless systems. Section III characterizes SCµM’s LC oscillator, specifically the relationship between the frequency setting and the frequency. Section IV introduces the QuickCal solution, including the QuickCal protocol, the frame format, and the algorithms used to select frequency setting. Section V evaluates QuickCal by measuring the quality of the resulting calibration and running a full 6TiSCH Industrial IoT heterogeneous network of a SCµM chip and OpenMote boards. Finally, Section VI concludes this article and presents future work.

II. RELATED WORK

Reducing the overall PCB design footprint and development costs are two innovation directions recognized by both industrial and academic researchers. The TI CC2652RB [17] chip is one of the first commercialized crystal-less 2.4 GHz wireless micro-controller. It is a System-in-Package which integrates a Bulk Acoustic Wave (BAW) MEMS resonator consisting of a piezoelectric material sandwiched between two electrodes. This resonator provides a stable and accurate frequency for the transceiver system. While indeed is allows for a crystal-less design on a PCB, it is not a single-chip solution.

Oscillating circuits internal to a chip (RC- or LC-based) can be used as a time reference. Song et al. [18] develop a 3.5 mm×3.5 mm crystal-less transceiver, used as a smart pill to help diagnose gastrointestinal diseases. The signal carrier is in the 402–405 MHz band, driven by a prescaled 1.6 GHz Digitally Controlled Oscillator (DCO). The carrier drift needs to be within ±1,000 ppm in order to be able to communicate with an external hub device outside the body. The transceiver includes a “Tunable Matching Network (TMN)”, which contains two capacitor banks. During calibration, the TMN sweeps the two capacitor banks and measures the TX output amplitude through an ADC. The calibration stops when the TX output amplitude reaches a pre-defined target level. The transceiver is also calibrated using a phase-tracking RX approach [19] while receiving packets from the external hub. The hub is equipped with an external crystal, which serves as a high accuracy time reference. Evaluation shows that the calibration results in ±125 ppm drift with a 5% power supply variation.

Alghaihab et al. [20] create a crystal-less BLE transmitter by using two LC oscillators for transmission and reception, respectively. In the transmitter, a clock recovery circuit with an 8 MHz intermediate frequency provides the references for two PLLs. These are used to lock to the frequency when a sequence of 3 packets on an advertisement channel are received. Once
the packets are received, the LC oscillator for transmission is calibrated, so that the transmitter is able to send packets over the locked frequency. The transceiver is measured to have a sensitivity of -86 dBm at a bit error rate of $10^{-3}$.

SCµM is the first crystal-free micro-mote that can inter-operate with standards-compliant radios (IEEE802.15.4 and Bluetooth Low Energy). Maksimovic et al. [13] give an overview of SCµM. Because of its crystal-free nature, SCµM uses an internal LC oscillator to synthesize the 2.4 GHz communication frequency. The challenge is that this oscillator exhibits a frequency with a variation of 2,100 ppm over the commercial temperature range, orders of magnitude higher than the 10-40 ppm drift typically seen with crystal oscillators. SCµM features a digitally controlled oscillator with a source-degenerated capacitive DAC, which allows the micro-controller to fine-tune the frequency of the oscillator. The software can compensate for the drift of the LC oscillator by tuning the DAC and bring the frequency to within 40 ppm as imposed by the IEEE802.15.4 standard.

Wheeler et al. [21] study the stability of SCµM’s LC oscillator in the absence of temperature changes. 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 ±40 ppm over 13 hours. This was done by 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 that number, SCµM determines whether to adjust the frequency of the LC oscillator.

To run the TSCH protocol, a low-power timer needs to be running continuously for a device to keep a precise notion of time. Khan et al. [22] design a 25 MHz ring oscillator to time the state machine of the TSCH protocol. To calibrate this timer, the authors use an OpenMote [16] as a network time reference, an IEEE802.15.4-compliant off-the-shelf device. The OpenMote sends packets at a fixed interval. By timestamping the reception of the start-of-frame delimiter (SFD) of each of these frames using the oscillator and knowing the period of the frames, Khan et al. [22] calibrate the oscillator’s frequency. This is similar to the same authors’ previous work on calibrating an LC oscillator [23]. This approach yields a maximum synchronization error of 853 µs for a 10-hop TSCH network with a 1 s data generation period.

Similar to the current article, Suciu et al. [15] develop a technique to calibrate the LC oscillator of SCµM. Their approach consists of an offline characterization followed by an online calibration. During the characterization step, the SCµM chip sweeps though all of the settings of its LC oscillator, and the LC counter, which is detailed in III-A, is used to map these settings against the measured oscillator frequency. The mapping is non-linear, so the authors explore different linearization approaches: recursive least squares (RLS) and moving average (MA) [24]. The results of this characterization (i.e. the linearization function) are stored in the chip. During online calibration, the SCµM chip sweeps through its LC oscillator settings once again until it successfully receives a frame sent by an OpenMote. The linearization function then allows is to compute the LC oscillator settings corresponding to the other 15 communication frequencies. The main downside of this approach is that the linearization function is different from mote to mote and potentially different over temperature.

The QuickCal technique we develop in Section III allows us to avoid the offline characterization from [15] altogether. Instead of the offline characterization of every chip, QuickCal relies entirely on frequency sweeping using a QuickCal Box containing off-the-shelf devices as a reference. QuickCal does not rely on any linearization either. One added benefit is that the frequency setting for each of the frequencies is measured experimentally rather than computed, resulting in a more dependable tuning.

In the work [12], a full 6TiSCH stack is running on SCµM and form a hybrid network with OpenMote boards. This did rely, however, on manual pre-calibration of a single frequency, and the 6TiSCH implementation used a single communication frequency, thereby not benefiting from the high reliability channel hopping brings. QuickCal lifts this restriction by allowing SCµM to self-calibrate its LC oscillator for all frequency channels, thereby enabling channel-hopping behavior.

In light of this related work, we start by characterizing SCµM’s LC oscillator, the oscillator which sets the communication frequency and which is the most challenging to calibrate (Section III). We then use the insight to design the QuickCal solution (Section IV). QuickCal is then evaluated in Section V, both in terms of Packet Delivery Ratio (PDR) and running a full 6TiSCH network.

III. Characterizing SCµM’s LC Oscillator

We first introduce the structure of the LC oscillator tuning system. We then show the sawtooth-shaped output frequency of SCµM over the LC oscillator settings on 2 SCµM development boards. Finally, we present a frequency sweep mechanism against OpenMote, and show the frequency distribution over all LC oscillator settings.

A. Overview of the LC Oscillator

Fig. 2 illustrates the structure of the LC oscillator tuning system. The output frequency of the LC oscillator ranges from 2.1 GHz to 2.6 GHz. The frequency is configured by setting three 5-bit registers – named coarse, mid and fine – which control 15-bits capacitive tuning DACs. In this paper, we represent the frequency setting with a 3-value tuple: $c.m.f$, where $c$, $m$ and $f$ represent the value of the coarse, mid and fine registers, respectively. The values of the frequency settings range from 0, 0, 0 to 31, 31, 31, 32,768 steps in total. A divider and a LC counter register are connected to the output of the LC oscillator. The LC counter increments at each cycle of the down-converted LC oscillator clock. The output frequency of the LC oscillator can be calculated from the value of the LC counter. For example, with the divider set...
LC oscillator frequency is to 960, if the LC counter increments by 2549 after 1 ms, the second oscillator and some timing circuitry. Fig. 3. Procedure for measuring the frequency of the LC oscillator using a counter through a divider, allowing us to measure its frequency.

The frequency settings of the LC oscillator for transmitting (TX) and receiving (RX) at the same frequency are different. For example, in Fig. 4(a), to transmit on channel 18 (2.440 GHz), the frequency setting needs to be between 25.18.22 and 25.19.27. For receiving on the same frequency, it needs to be between 25.21.25 and 25.22.28. This difference is partly because of load pulling: during reception, the mixer is turned on while the power amplifier is turned off; during transmission, the mixer is turned off while the power amplifier is turned on. This changes the capacitance of the LC oscillator, resulting in a different register configuration to obtain the same frequency. This difference is also because SC\(\mu\)M uses a low-intermediate frequency (IF) receiver, which requires a difference in the LC oscillator setting between transmit and receive\(^2\). While the latter is specific to the design of the SC\(\mu\)M radio, we believe the effect load pulling to be generic and possibly affecting other designs.

The frequency settings are different from chip to chip. For example, to transmit on channel 18 (2.440 GHz) the frequency setting needs to be between 25.18.22 and 25.19.27 on board Q3 (Fig. 4(a)) and between 25.14.6 and 25.15.6 on board Q8 (Fig. 4(b)). This chip-to-chip variation is primarily due to effective reference mismatch between voltage regulators and DC supply sensitivity of the oscillator. Because of the limited available room for on-chip capacitance for regulator stability, the amplifier input pair needs to be kept small, which results in a large and unpredictable offset.

Because frequency settings are different, it is essential to be able to calibrate these settings automatically. Please note that, while we fine-tune the specifics of the QuickCal solution to SC\(\mu\)M, any crystal-free architecture requires similar (self-)calibration.

C. Computer-Assisted Calibration

We develop the base principle of calibration in this section using a computer. We then turn that base principle into a full self-calibration routine (i.e. which SC\(\mu\)M carries out on its own without computer assistance) in the QuickCal solution (Section IV).

Fig. 6 shows the setup we use: a SC\(\mu\)M chip and an OpenMote are connected to a computer and can send serial data through that connection to the computer. The goal is to determine the LC oscillator frequency setting SC\(\mu\)M needs to use to both transmit to and receive from the OpenMote on any IEEE802.15.4 frequency. Calibration happens in two steps: to calibrate the TX (SC\(\mu\)M → OpenMote) and RX (OpenMote → SC\(\mu\)M). In both cases, we have SC\(\mu\)M sweep through its frequency settings, looking for the setting that “works”.

2 For example, to communicate at 2.405 GHz, the LC oscillator needs to have a frequency of 2.4055 GHz when transmitting and a frequency of 2.4025 GHz when receiving. There is hence a 3 MHz difference between the required settings regardless of load pulling.
To calibrate $\text{SC}\mu\text{M}$’s LC oscillator for TX, we have the OpenMote continuously listening on a particular channel and reporting the RSSI and the frequency offset of the frames it receives. We have $\text{SC}\mu\text{M}$ sweep through all of its 32,768 frequency settings and transmit 3 frames for each setting, one every 5 ms. $\text{SC}\mu\text{M}$ indicates the frequency setting it is using inside the frame it sends. We do the opposite to calibrate $\text{SC}\mu\text{M}$’s LC oscillator for RX: OpenMote continuously transmits on a particular channel (a frame every 7 ms) while $\text{SC}\mu\text{M}$ sweeps through its frequency settings, listening for 15 ms on each setting. We repeat this procedure for each of the 16 channels.

Fig. 7 illustrates the results of TX calibration ($\text{SC}\mu\text{M} \rightarrow \text{OpenMote}$) on channel 13. The x-axis is the frequency setting. The top plot shows the frequency $\text{SC}\mu\text{M}$ transmits on, as in Fig. 4. Each dot in the bottom plots indicates that the OpenMote has received a frame; these plots show the frequency offset and the RSSI for that frame. As expected, the OpenMote receives frames on channel 13 (highlighted in green). The OpenMote also receives frames sent by $\text{SC}\mu\text{M}$ on frequencies 2.270 GHz and 2.343 GHz (highlighted in red) at a much lower RSSI. The CC2538 should not be able to receive signals at 2.270 GHz and 2.343 GHz but, without insight into its design details, we cannot speculate as to the reason it is able to receive $\text{SC}\mu\text{M}$ transmissions at such low frequencies.

Fig. 8 illustrates the results of RX calibration (OpenMote $\rightarrow \text{SC}\mu\text{M}$) on channel 13. The bottom plot is a histogram of the number of frames received by $\text{SC}\mu\text{M}$ for each frequency setting. As expected, $\text{SC}\mu\text{M}$ received frames for a narrow range of frequency settings.

The sweeping technique used in this section can be used to allow $\text{SC}\mu\text{M}$ to self-calibrate without a computer. This is the base mechanism used by QuickCal.

IV. THE QUICKCAL SOLUTION

The goal of QuickCal is to allow a $\text{SC}\mu\text{M}$ chip to self-calibrate its LC oscillator, so that it can communicate with other $\text{SC}\mu\text{M}$ chips and off-the-shelf IEEE802.15.4 devices, for example by forming a 6TiSCH network. QuickCal uses off-the-shelf IEEE802.15.4 devices as references for the calibration. Those devices form groups of 16 OpenMotes, called
Fig. 7. Results of TX calibration (SC\(\mu\)M \(\rightarrow\)OpenMote) on channel 13. The top plot shows the frequency SC\(\mu\)M transmits on, as in Fig. 4. Each dot in the bottom plots indicates that the OpenMote has received a frame; these plots shows the frequency offset and the RSSI for that frame.

Fig. 8. Results of RX calibration (OpenMote \(\rightarrow\)SC\(\mu\)M) on channel 13. The top plot shows the frequency SC\(\mu\)M receives on. The bottom plot is a histogram of the number of frames received by SC\(\mu\)M for each frequency setting.
a “QuickCal Box”. At least one QuickCal Box needs to be within range of each SCµM chip for QuickCal to work, as depicted in Fig. 9.

The OpenMotes of a QuickCal Box run firmware that implements the QuickCal protocol. The firmware on SCµM exchanges messages using the QuickCal protocol with the QuickCal Box to self-calibrate. The self-calibration algorithm SCµM uses is based on the sweeping mechanism explored in Section III-C. The SCµM chip starts with no LC oscillator calibration whatsoever; the objective is to determine 32 frequency settings: one for transmitting and one for receiving at each of the 16 frequencies. Conceptually, QuickCal works as follows. The 16 OpenMotes in the QuickCal Box regularly send frames, each on a different frequency. SCµM sweeps over its frequency settings while listening to determine the settings for receiving. It then sweeps again while sending a frame and listening for an acknowledgement frame on the same frequency, to determine the settings for transmitting. The different timings of the QuickCal protocol, including when to update the frequency setting, when to send the next frame, and how long to wait a frame, are scheduled using the 500 kHz RFTimer clock.

The goal for the remainder of this section is to turn the high-level behavior described above into a working system and provide all necessary details. Section IV-A introduces the QuickCal protocol, based on only 3 frame types. Section IV-B describes the QuickCal Box: its hardware components and some details about the firmware implementation. Sections IV-C and IV-D then detail the exact procedure SCµM uses to self-calibrate its LC oscillator, both for receiving from and transmitting to the QuickCal Box.

A. The QuickCal Protocol

The SCµM chip and the QuickCal Box exchange frames that are of 3 types: CalBeacon, CalProbe, and CalAck. The QuickCal Box continuously and periodically sends CalBeacon frames, which SCµM listens for to calibrate its LC oscillator for reception. To calibrate for transmission, SCµM sends CalProbe frames to the QuickCal Box, which responds with a CalAck frame.

Fig. 10 shows the format of these frames. They are valid IEEE 802.15.4 frames: they start with a 4-byte physical preamble, followed by a 1-byte start-of-frame delimiter (SFD) and a 1-byte length field, and end with a 2-byte frame check sequence (FCS). Because these frames are sent regularly, they are designed to be as short as possible: they each have only 2 bytes of payload. Because they are sent at specific times (see below), they cannot be confused for one another, so a “type” field is not needed.

Each of the 16 OpenMotes in the QuickCal Box sends CalBeacon frames regularly on a different frequency. These contain the channel they are sent on in IEEE notation (i.e. channel 11 corresponds to 2.405 GHz, channel 12 to 2.410 GHz, etc). They also contain a sequence number used for joining (see Section IV-C1). The SCµM chip sends CalProbe frames to the QuickCal Box (see Section IV-D). In the channel field of these frames, the SCµM chip indicates the channel on which it thinks it is sending the frames This allows an OpenMote in the QuickCal Box that receives the CalProbe to decide whether to send back a CalAck frame. As a rudimentary method for OpenMotes not to mistake a CalProbe for another 2-byte frame sent by a nearby IEEE802.15.4 device, CalProbe frames contains the magic field set to the value 0xCF (for “Crystal-Free”).

The QuickCal Box sends a CalAck frame as a response to a CalProbe frame. That CalAck frame contains the
frequency offset (freq_offset field), which the CC2538 on the OpenMote measured when receiving the CalProbe frame (using its FREQ_OFFSET register). This provides the SCµM chip with some information about how close its frequency setting is to the standard frequency (2.405 GHz, 2.410 GHz, etc).

Note that, per the results in Section III, the frequency settings covering the 2.400–2.480 GHz band are located between frequency settings 22.0.0 and 28.31.31 at room temperature. This means that only 7,168 frequency settings span the entire IEEE802.15.4 2.4 GHz band, not the full 32,768 settings.

B. The QuickCal Box

We reuse the OTBox from the OpenTestbed [26], shown in Fig. 11, to serve as a QuickCal Box. A QuickCal Box is composed of 16 OpenMote devices, which we number MM1 through MM16. Each OpenMote is assigned a different frequency channel (channel 11 for MM1, channel 12 for MM2, etc). The OpenMotes synchronize to one another to be able to send their CalProbe frames one after the other.

The OpenMotes in the QuickCal Box implement the schedule depicted in Fig. 12, which continuously repeats. Each timeslot in the schedule is 3 s long; slots are numbered 0 through 15.

This schedule orchestrates when each OpenMote sends and receives, so OpenMotes need to be synchronized to one another. The OpenMote numbered MM1 is the time master. At the beginning of timeslot 0, MM1 transmits 1,000 CalBeacon frames, one every 600 µs, on frequency channel 11 (2.405 GHz). When switched on, other OpenMotes continuously listen to channel 11 for these CalBeacon frames. Each CalBeacon frame in a burst contains a field with an counter that increments for each frame, allows OpenMotes MM2–MM16 to compute when each of the slots in the schedule starts. OpenMotes MM2–MM16 repeat this every time slot 0 repeats (i.e. every 48 s), to stay synchronized to OpenMote MM1.

In slots 1–16, each OpenMote is assigned a different frequency: OpenMote MM2 is assigned channel 12, OpenMote MM3 is assigned channel 13, etc. In addition, each OpenMote is assigned a transmit slot: OpenMote MM2 is assigned slot 1, OpenMote MM3 is assigned slot 2, etc. Similar to OpenMote MM1, each other OpenMote sends 1,000 CalBeacon frames, one every 600 µs, during its transmit slot. Since each OpenMote is configured to communicate on a different frequency, this results in successive 1,000 CalBeacon frame bursts sent by the QuickCal Box across all frequencies. The SCµM chip will use this to calibrate its LC oscillator for reception.

When an OpenMote is not sending CalBeacon frames, it continuously listens. The result is that, at any point in time, one OpenMote will be sending CalBeacon frames, and the others will be listening, each on a different frequency. When listening, if an OpenWSN receives a CalProbe frame, it is programmed to immediately send back a CalAck frame. The SCµM chip will use this to decide whether it can successfully send a CalProbe frame and receive a CalAck frame, as part of calibrating its LC oscillator for transmission.

The SCµM chip calibrates in two steps: it calibrates its LC oscillator to first receive frames from the QuickCal Box (Section IV-C) and then to transmit frames to the QuickCal Box (Section IV-D).

C. QuickCal Step 1: Receiving from the QuickCal Box

The goal of step 1 is for the SCµM chip to self-calibrate for reception. It starts by synchronizing to the QuickCal Box schedule, then sweeps its frequency settings, and then picks the 16 best frequency settings (one for each frequency channel).

1) Initial Synchronization: the SCµM chip starts by listening for CalBeacon frames sent on channel 11 by the QuickCal Box. We know from Section III-B that the frequency setting corresponding to channel 11 is in the 23.0.0–24.31.31 range (2,048 settings). SCµM repeatedly listens for 800 µs on each frequency setting, sweeping through
Box schedule, the SC
the best setting. At the end of one iteration of the QuickCal
settings; it uses the algorithm in Section IV-C3 to determine
process, it records
Alg. 1 to select the “best” frequency setting
settings on which it has successfully received a
CalBeacon
frames on a particular frequency. During that
chip sweeps 2,048 frequency settings while listening for
settings for receiving on each channel.

all 2,048 settings, which takes 1.6 s. The SCµM chip
likely starts this process when MM1 is not transmitting
CalBeacon frames, therefore repeats the sweep until it
receives CalBeacon frames. Once the SCµM chip has
received at least one CalBeacon frame, it computes when
the different slots in the QuickCal Box schedule start.

2) Sweeping Frequency Settings: the SCµM chip knows
when slots 1 through 15 start. During each slot, it listens for
CalBeacon frames on a different frequency. It sweeps its
frequency settings “up”, starting from the frequency setting
select from the previous slot, for 2,048 settings. We know
from Section III-B that the range of settings covers the
2 MHz separating adjacent IEEE802.15.4 frequencies. Fig. 13
illustrates this sweeping process. The SCµM chip repeats this
process over the 16 slots in the QuickCal Box schedule. Each
time, it receives CalBeacon frames for multiple frequency
settings; it uses the algorithm in Section IV-C3 to determine
the best setting. At the end of one iteration of the QuickCal
Box schedule, the SCµM chip has determined the 16 frequency
settings for receiving on each channel.

3) Selecting the Best Frequency Setting: the SCµM chip
sweeps 2,048 frequency settings while listening for
CalBeacon frames on a particular frequency. During that
process, it records \( S = (f_1, f_2, \ldots, f_n) \), the set of \( n \) frequency
settings on which it has successfully received a CalBeacon.
In our experiments, \( n \) has an average of 65. SCµM then uses
Alg. 1 to select the “best” frequency setting \( \hat{f} \) out of \( S \).

The idea is to select a frequency setting which is “in the
middle” of a range of frequency settings where reception is
possible. In case of a temperature change, the same frequency
setting results in a different frequency. To be as robust as
possible, we want to still be able to receive at that other
frequency. Given the saw-tooth shape of Fig. 5, we select a range
of frequency settings that share the same \( \text{mid} \) component.

This algorithm works as follows. It starts by splitting \( S \)
into subsets \( S_{\text{coarse}} \) of frequency settings with the same \( \text{coarse} \) component, such that \( S = S_{\text{coarse}} \cup S_{\text{mid}} \). It selects \( S_{\text{mid}} \), the subset
with the most components. It splits \( S_{\text{mid}} \) into subsets \( S_{\text{most}} \) of
frequency settings with the same \( \text{mid} \) component, such that
\( S_{\text{mid}} = S_{\text{most}} \cup S_{\text{mid}} \cup \ldots \). It then selects \( S_{\text{most}} \), the subset with the
most components. Finally, it selects the median value of \( S_{\text{mid}} \)
as the “best” frequency \( \hat{f} \).

Algorithm 1: Selecting the receive frequency setting

Result: \( \hat{f} \), the frequency setting to use

\[
S = f_1, f_2, \ldots, f_n; \\
S_{\text{coarse}} = \text{Set}(x_{\text{coarse}} = \text{mid}; \\
i = 1; \\
\text{while } x \leq n \text{ do} \\
\text{if } \text{len}(S_{\text{coarse}}) = \max(\text{len}(S_{\text{coarse}}), \text{len}(S_{\text{mid}}), \ldots) \text{ then} \\
\quad c_{\text{max}} = ci; \\
\quad S_{\text{mid}} = S_{\text{coarse}}; \\
\quad \text{break}; \\
\text{end} \\
i++; \\
\text{end} \\
S_{\text{coarse}} = \text{Set}(f_{\text{coarse}}) \text{ where } x_{\text{coarse}} = c_{\text{max}} \text{ and } \text{len}(S_{\text{coarse}}) = j; \\
\text{while } j \leq n \text{ do} \\
\text{if } \text{len}(S_{\text{coarse}}) = \max(\text{len}(S_{\text{coarse}}), \text{len}(S_{\text{mid}}), \ldots) \text{ then} \\
\quad m_{\text{max}} = mj; \\
\quad S_{\text{coarse}} = S_{\text{coarse}} \cup S_{\text{mid}}; \\
\quad \text{break}; \\
\text{end} \\
j++; \\
\text{end} \\
\hat{f} = \text{median}(S_{\text{coarse}})
\]

D. QuickCal Step 2: Transmitting to the QuickCal Box

The goal of step 2 is for the SCµM chip to self-calibrate
for transmission. It is already synchronized to the QuickCal
Box schedule and has just completed step 1.

1) Sweeping Frequency Settings: the core idea of step 2
is the same as step 1. The main difference is that, instead of
listening for CalBeacon frames sent by the QuickCal Box,
the SCµM chip sends CalProbe frames to the QuickCal Box
and then listens for CalAck frames sent by the QuickCal Box
as acknowledgements. The SCµM chip uses the frequency
setting it is sweeping to send CalProbe frames but uses
the frequency setting determined in step 1 when listening
for CalAck frames. As shown in Fig. 14, the SCµM chip
sends CalProbe frames on channel 11 (2.405 GHz) in slot
1, i.e. one slot after MM1 has transmitted CalBeacon frames
on channel 11, which we used in step 1.

A CalProbe frame is 320 µs long. After receiving it, the
QuickCal Box sends a CalAck approximately 300 µs later.
A CalAck frame is 320 µs long. To allow for enough time
for the Ack to arrive, the SCµM chip sends
one CalAck frame every 1.2 ms.
Fig. 14. QuickCal Step 2: transmitting to the QuickCal Box. SCμM sweeps its frequency settings while transmitting CalProbe frames on all 16 frequencies to the OpenMote devices in the QuickCal Box and waiting for CalAck frames. It sends CalProbe frames to OpenMote MM1 on channel 11 in slot 1, it sends CalProbe frames to OpenMote MM2 on channel 12 in slot 2, etc. The SCμM chip receives CalAck frames for multiple frequency settings; it uses the algorithm in Section IV-D2 to determine the best setting.

Algorithm 2: Selecting the transmit frequency setting

Result: $f$, the frequency setting to use

$S = f_1, f_2, \ldots, f_n$, frequency settings CalProbe frames are sent on;

$O = o_1, o_2, \ldots, o_n$, frequency offsets contained in corresponding CalAck frames;

$x = 1$;

while $x \leq n$ do

if $\min(abs(o_1), abs(o_2), \ldots)$ then

$f = f_x$;

break;

end

$x++$;

end

Fig. 15. SCμM sends 1,000 CalProbe frames and listens for CalAck frames on each of the 16 frequency channels. We define the packet delivery ratio (PDR) as the portion of CalAck frames received and use it to quantify the quality of the QuickCal self-calibration routine.

A. Packet Delivery Ratio

QuickCal allows SCμM to self-calibrate 32 frequency settings. It then uses those to send and receive data to other devices, including off-the-shelf IEEE802.15.4 devices, such as the OpenMote. To quantify the “quality” of the self-calibrated 32 frequency settings, we have SCμM chip exchange frames with a nearby OpenMote device and measure the packet delivery ratio, i.e. the fraction of frames sent that are successfully received. A PDR of 100% is the ideal case.

Specifically, as shown in Fig. 15, we have SCμM send 1,000 CalProbe frames and listen for CalAck frames the OpenMote is programmed to send as a response. This is done for each of the 16 frequency channels. We then compute the PDR as the fraction of CalAck frames received, indicating the fraction of times the two-way handshake succeeded. We repeat this 200 times to obtain statistically relevant results.

Taking into account the time it takes to repeatedly reprogram the different devices, the experiment is conducted overnight and takes approximately 14 h. Fig. 16 shows the measured PDR as a function of the frequency channel. The average PDR of 13 of the 26 channels is above 90%, one is around 85%, and two are around 70%. This distribution is typical for a cluttered environment (such as the office this test was run in) in which multi-path affects a handful of channels. WiFi interference explains the outliers. Overall, results in Fig. 16 are perfectly in line with off-the-shelf IEEE802.15.4 devices.

B. Energy Consumption

We first evaluate the quality of the frequency setting discovered using QuickCal by measuring the resulting Packet Delivery Ratio (PDR) between the SCμM chip that self-calibrated and a nearby OpenMote (Section V-A). We then analyze the energy consumption of QuickCal based on the charge needed to transmit CalProbe frames and receive CalBeacon and CalAck frames (Section V-B). At last we run a heterogeneous 6TiSCH network composed of a SCμM chip and one OpenMote, channel-hopping over all 16 frequency channels (Section V-C).
and runs only once but more a quick solution to find the initial 16 frequency settings each for TX and RX. For using once in a life time, it is acceptable.

C. A Heterogeneous 6TiSCH Network

With Section V-A confirming the quality of the QuickCal self-calibration routine, we can run a full protocol stack. Specifically, we run the 6TiSCH Industrial IoT protocol stack recently standardized by the IETF [27]. 6TiSCH uses Time Synchronized Channel Hopping (TSCH), in which devices synchronize to one another and channel-hop across all available frequencies. Its synchronized and channel-hopping nature makes 6TiSCH a good “stress-test” for QuickCal.

We port OpenWSN, the reference implementation of 6TiSCH [28], onto the SC$\mu$M chip\(^3\). OpenWSN is already ported onto the OpenMote.

The test setup consists of a SC$\mu$M chip operating as a 6TiSCH mote and joining an OpenMote that operates as a 6TiSCH root. A QuickCal Box is nearby, allowing the SC$\mu$M chip to self-calibrate before joining the network. We purposefully keep the network atomically small and focus on the behavior of the SC$\mu$M mote; its behavior here is representative of its behavior in a larger 6TiSCH network.

The test is conducted in a typical office building, in which several Wi-Fi routers operate on the 2.4 GHz band. During office hours, the continuous interference from Wi-Fi activity can be observed in the communication between OpenMote and SC$\mu$M, as shown in Fig. 17.

The OpenMote sends broadcast frames (Enhanced Beacons and RPL DIO frames) every 0.4 s on average. Once part of the network, the SC$\mu$M chip sends application-level data packets to the OpenMote every 8 seconds. The 6TiSCH guard time (the maximum de-synchronization allowed between the devices) is set to 1.3 ms.

As part of the 6TiSCH protocol stack, the SC$\mu$M chip re-synchronizes to the OpenMote every time they exchange a frame. We instrument the code on SC$\mu$M to output via its serial port by how many micro-seconds it re-synchronizes. Fig. 17 shows the time correction of SC$\mu$M over time. It also shows the type of synchronization: “Packet” indicates SC$\mu$M synchronizes by receiving a broadcast frame from the OpenMote and “Ack” indicates SC$\mu$M synchronizes by receiving an acknowledgment frame from the OpenMote.

Fig. 17 shows that the time correction is well within the 1.3 ms guard time throughout most of the 40 min test. The SC$\mu$M chip does get de-synchronized 4 times during the experiment but resynchronizes in less than 70 s to the OpenMote. De-synchronization is likely due to temperature fluctuations (temperature compensation is left for future work as described in Section VI) or external interference from Wi-Fi. The HF RC oscillator is used for timing the 6TiSCH TSCH protocol and keep synchronized to the network. A small temperature change results in additional drift in the HF RC oscillator, which translates into a larger time offset between the SC$\mu$M chip and the OpenMote. Wi-Fi interference may cause collisions, making it harder for SC$\mu$M to periodically synchronize to the OpenMote.

Interference from the QuickCal Box could also impact the 6TiSCH network between SC$\mu$M and OpenMote. After QuickCal runs, SC$\mu$M participates in a 6TiSCH network. It is possible that, at the same time, the QuickCal Box assists another SC$\mu$M chip to calibrate. Since they are running on the same 2.4 GHz band, there is a non-zero chance the 6TiSCH frames collide with the frame sent by QuickCal Box. The impact of this contention is limited because of the channel hopping nature of the 6TiSCH communication.

The QuickCal Box repeatedly transmits on each channel for 3 s with an order of channels which is known a priori. When SC$\mu$M running 6TiSCH stack sends a frame on channel 11 and a QuickCal Box is simultaneously sending frames on the same channel, the the 6TiSCH transmissions likely fail. SC$\mu$M will then retransmit in the next slotframe (roughly 2 s later, assuming a 101 slot slotframe length and a 20 ms timeslot duration). The channel hopping nature of 6TiSCH ensures that retransmission happens on a different channel, avoiding further contention with the QuickCal Box.

VI. CONCLUSION AND FUTURE WORK

The Single Chip Micro Mote (SC$\mu$M) is the first standards-compliant single-chip crystal-free mote-on-chip. A crystal-free architecture has two key advantages. First, cost: devices do not
need to include crystal oscillators (which can cost as much as the radio itself) and, in the extreme case, not even a printed circuit board. Second, size: the resulting design can be the size of a grain of rice, which opens up the possibility of using these devices at the heart of micro-robots. However, using a crystal-free architecture is challenging as the internal RC and LC oscillators we are left with suffer from much worse frequency stability.

QuickCal is a self-calibration routine that allows the SC\textsubscript{μ}M chip to be able to communicate with off-the-shelf IEEE802.15.4 devices and eventually heterogeneous networks. Using QuickCal, the SC\textsubscript{μ}M chip self-calibrates by communication with a QuickCal Box: off-the-shelf IEEE802.15.4 devices dedicated to assisting crystal-free chips to self-calibrate. QuickCal is the combination of the QuickCal Box, the QuickCal protocol, and the self-calibration routine implemented by the SC\textsubscript{μ}M chip. The goal of self-calibration is for the SC\textsubscript{μ}M chip to determine 32 calibration values, one for transmitting and one for receiving at each of the 16 frequencies.

We implement QuickCal using a combination of OpenMote devices and SC\textsubscript{μ}M chips. We show that it takes SC\textsubscript{μ}M less than 3 min to fully self-calibrate. We then confirm that this calibration allows it to communicate with an off-the-shelf IEEE802.15.4 device on all frequencies. Finally, we demonstrate a heterogeneous network with both a SC\textsubscript{μ}M chip and an OpenMote forming a 6TiSCH network. This is particularly challenging as 6TiSCH relies on tight synchronization and channel hopping.

The technique presented in this article relies on 16 OpenMotes dedicated to calibrating SC\textsubscript{μ}M chips. While in article only a single SC\textsubscript{μ}M chip is calibration, the same 16 OpenMotes can be used to calibrate hundreds or thousands of SC\textsubscript{μ}M chips. Even then, the cost of the 16 OpenMotes is significant. Our current work focuses on using 16 OpenMotes which are already part of a network to temporarily switch to a calibration role, possibly driven by a central orchestration authority.

The work presented in this article opens up further avenues for future work. First, we are working on SC\textsubscript{μ}M to continuously adapt its frequency settings as it participates in a network, allowing it to adapt, for example, to temperature changes without losing connectivity. Second, while having a QuickCal Box allows for SC\textsubscript{μ}M chips to operate, we acknowledge that deploying a QuickCal Box is impractical. We are working on integrating the QuickCal protocol into the 6TiSCH network to allow the SC\textsubscript{μ}M chip to self-calibrate against an operating network without requiring dedicated hardware.

REFERENCES


Tengfei Chang received his PhD degree in Computer Sciences from the University of Science and Technology, Beijing in January 2018. In 2014, he was a visiting scholar at the University of California, Berkeley. He joined Inria Paris in 2015 as a research engineer then postdoctoral research lead. He is currently leading the OpenWSN project, an open-source project founded at UC Berkeley. He is one of the main implementers of the IETF 6TiSCH protocol stack. He is active in the standardization activity where he serves as the editor the 6TiSCH Minimal Scheduling Function (MSF) standard. His research interests are wireless sensor and actuator networking, swarm robotics and embedded system design.

Thomas Watteyne is an insatiable enthusiast of low-power wireless mesh technologies. He holds an Research Engineer position at Inria in Paris, in the EVA research team, where he leads a team that designs, models and builds networking solutions based on a variety of Internet-of-Things (IoT) standards. He is Senior Networking Design Engineer at Analog Devices, in the Dust Networks product group, the undisputed leader in supplying low power wireless mesh networks for demanding industrial process automation applications. Since 2013, he co-chairs the IETF 6TiSCH working group, which standardizes how to use IEEE802.15.4e TSCH in IPv6-enabled mesh networks, and is member of the IETF Internet-of-Things Directorate. Prior to that, Thomas was a postdoctoral research lead in Prof. Kristofer Pister’s team at the University of California, Berkeley. He founded and co-leads Berkeley’s OpenWSN project, an open-source initiative to promote the use of fully standards-based protocol stacks for the IoT. Between 2005 and 2008, he was a research engineer at France Telecom, Orange Labs. He holds a PhD in Computer Science (2008), an MSc in Networking (2005) and an MEng in Telecommunications (2005) from INSA Lyon, France. He is Senior member of IEEE. He is fluent in 4 languages.

Brad Wheeler received a B.S. degree in electrical and computer engineering from the University of Missouri, Columbia, in 2008, and a PhD degree in electrical engineering from the University of California, Berkeley, in 2019. His research interests include system and circuit level design for low-power communication systems.

Filip Maksimovic received a B.S. degree in electrical engineering and aerospace engineering from the University of Colorado at Boulder, and a PhD degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, in 2018. His research interests include low-power radio frequency communication and wireless sensing.
David C. Burnett received the B.S. and M.S. degrees in Electrical Engineering from the University of Washington focusing on circuits, embedded systems, and oceanographic instrumentation. He recently completed the Ph.D. in Electrical Engineering and Computer Sciences at the University of California Berkeley as an NDSEG Fellow focusing on crystal-free fully-integrated wireless sensor nodes. Prior to his Ph.D., he was Member of Technical Staff at Sandia National Laboratories, Livermore, Visiting Lecturer at Da Nang University of Technology, Vietnam, and Electrical Engineer for an experimental ROV at McMurdo Station, Antarctica. He has served on various ACM SIGGRAPH conference committees, serving as submissions juror and responsible for special technical projects and data networks. His research interests include RF communication, low-power circuit design, and field-deployable sensor systems. He is a Senior Member of the IEEE.

Titan Yuan received a B.S. degree in electrical engineering and computer sciences from University of California, Berkeley, in 2019 and is currently pursuing his M.S. degree in electrical engineering and computer sciences at University of California, Berkeley, advised by Prof. Kristofer S.J. Pister. He was named a Siebel Scholar in 2019 and is interested in working on embedded systems and developing new and exciting applications for SCPuM.

Xavier Vilajosana is principal investigator at the Wireless Networks (WiNe) research Lab at UOC and full professor at the Computer Science, Telecommunications and Multimedia department. Xavier is also co-founder of Worldsensing, and OpenMote Technologies. Until March 2016 Xavier has been senior researcher at the HP R&D Labs. From January 2012 to January 2014, Xavier was visiting Professor at the University of California Berkeley holding a prestigious Fulbright fellowship. In 2008, he was visiting researcher of France Telecom R&D Labs, Paris. Xavier has been one of the main promoters of low power wireless technologies, co-leading the OpenWSN.org initiative at UC Berkeley, and promoting the use of low power wireless standards for the emerging Industrial Internet paradigm. He also contributed to the industrialization and introduction of Low Power Wide Area Networks to urban scenarios through Worldsensing. Xavier is author of different Internet Drafts and RFCs at the IETF, as part of his standardization activities for low power industrial networks. Xavier is contributing actively at the IETF 6TiSCH, 6Lo and ROLL Working Groups. Xavier holds more than 30 patents, more than 50 high impact journal publications and have contributed with several demos, tutorials and courses in the field of low power wireless networks. Finally Xavier is IEEE Senior Member and founding member and vocal of the IEEE Sensors Council in Spain.

Kristofer S. J. Pister received a B.A. degree in applied physics from UCSD, in 1986, and M.S. and Ph.D. degrees in electrical engineering from UC Berkeley, in 1989 and 1992, respectively. From 1992 to 1997, he was an Assistant Professor of electrical engineering with UCLA, where he helped in the development of the graduate MEMS curriculum and coined the phrase Smart Dust. Since 1996, he has been a Professor of electrical engineering and computer sciences with the University of California at Berkeley, Berkeley. In 2003 and 2004, he was on leave from UCB as the CEO and then the CTO of Dust Networks, a company he founded to commercialize wireless sensor networks. He has participated in the creation of several wireless sensor networking standards, including Wireless HART (IEC62591), IEEE 802.15.4e, ISA100.11a, and IETF RPL. His research interests include MEMS, micro robotics, autonomous microsystems, and low-power circuits.