Temperature-Resilient Time Synchronization for the Internet of Things

Atis Elsts, Xenofon Fafoutis, Simon Duquennoy, George Oikonomou, Robert Piechocki, Ian Craddock

Temperature-Resilient Time Synchronization for the Internet of Things

Atis Elsts, *Member, IEEE*, Xenofon Fafoutis, *Member, IEEE*, Simon Duquennoy, George Oikonomou, Robert Piechocki and Ian Craddock, *Fellow, IEEE*

***Abstract*—Networks deployed in real-world conditions have to cope with dynamic, unpredictable environmental temperature changes. These changes affect the clock rate on network nodes, and can cause faster clock de-synchronization compared to situations where devices are operating under stable temperature conditions. Wireless network protocols such as Time-Slotted Channel Hopping (TSCH) from the IEEE 802.15.4-2015 standard are affected by this problem, since they require tight clock synchronization among all nodes for the network to remain operational. This paper proposes a method for autonomously compensating temperature-dependent clock rate changes. After a calibration stage, nodes continuously perform temperature measurements to compensate for clock drifts at run-time. The method is implemented on low-power IoT nodes and evaluated through experiments in a temperature chamber, indoor and outdoor environments, as well as with numerical simulations. The results show that applying the method reduces the maximum synchronization error more than 10 times. In this way, the method allows reduce the total energy spent for time synchronization, which is practically relevant concern for low data rate, low energy budget TSCH networks, especially those exposed to environments with changing temperature.**

I. Introduction

There is a growing need to make low-power wireless networks more reliable and more predictable in order to open them up to a wider range of Internet of Things (IoT) applications, such as industrial [1], automotive, and e-health applications. Time-Slotted Channel Hopping, as specified in the IEEE 802.15.4-2015 standard [2], is a Medium Access Control (MAC) protocol that offers high reliability and predictability through channel hopping and time-synchronized operation. Because of these properties, TSCH has attracted attention both from the industry and the academia.

TSCH networks deployed in real-world conditions [3] – in particular, in industrial conditions [4] – have to cope with dynamic and unpredictable temperature changes, which affect hardware clock rates. To maintain connectivity in spite of changing clock rates, nodes either have to relax their synchronization requirements (*i.e.*, increase the TSCH guard time), or increase re-synchronization traffic – both of which increase their energy usage requirements [5]. This problem is particularly relevant in low data rate, low energy budget applications, *e.g.* where packets are transmitted every few minutes. Under these conditions, the energy requirements for keeping the network synchronized tend to exceed the energy requirements for data transmission.

This paper proposes *adaptive, temperature-resilient time synchronization*: a method to counteract temperature-dependent clock frequency changes. The method consists of three main elements: (1) one-time calibration of the effect of temperature on hardware clocks, (2) continuous temperature measurements, and (3) continuous temperature-dependent drift compensation relative to the network coordinator. Unlike existing work on temperature-resilient TSCH synchronization [6] [7], the proposed method does not require sending more packets when temperature is changing – instead, nodes compensate the effects of temperature locally and maintain stable clocks. Our method is therefore well suited for networks where the number of packets must be minimized, or high *a priori* predictability (*e.g.*, of energy consumption) is required, and the nodes are equipped with temperature sensors.

We present a thorough characterization of the effects of temperature of clocks, with analytic results and simulations, to motivate our design. Empirical temperature and clock drift measurements inside a temperature chamber are used as the input for these simulations and the analytic model. The simulation results are validated with a controlled experiment in the temperature chamber, as well as by a 15 h experiment indoors and a 15 h experiment outdoors where nodes are exposed to direct sunlight. The results show that the algorithm almost always keeps the TSCH network synchronized using the default settings and 10 min resynchronization interval, and that it more than ten times reduces the required number of synchronization packets.

The algorithm is implemented for SPES-2 [8], a low-power IoT node based on the Texas Instruments CC2650 System-on-Chip (SoC) and equipped with a common off-the-shelf HDC1000 temperature sensor [9]. The implementation builds on the Contiki TSCH code [10], and on the high-accuracy energy-efficient adaptive synchronization for TSCH for CC2650 described by Elsts *et al.* in [11].

Section II clarifies the qualitative and quantitative differ-

This work was performed under the SPHERE IRC funded by the UK Engineering and Physical Sciences Research Council (EPSRC), Grant EP/K031910/1. It was also partly funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 761586 (5G-CORAL), the distributed environment Ecare@Home funded by the Swedish Knowledge Foundation, and by a grant from CPER Nord-Pas-deCalais/FEDER DATA.

A. Elsts, X. Fafoutis, G. Oikonomou, R. Piechocki and I. Craddock are with the Department of Electrical and Electronic Engineering, University of Bristol, Bristol, U.K. (e-mail: atis.elsts@bristol.ac.uk, xenofon.fafoutis@bristol.ac.uk, g.oikonomou@bristol.ac.uk, r.j.piechocki@bristol.ac.uk, ian.craddock@bristol.ac.uk). S. Duquennoy is with Inria, Lille, France, and with RISE SICS, Stockholm, Sweden (email: simon.duquennoy@ri.se).

Supporting data available at https://github.com/IRC-SPHERE/elsts2017-temperature-data.

ences compared to related work; Section III includes a mathematical model of time synchronization in TSCH networks and analyzes the causes of synchronization errors. Section IV describes the empirical investigation of temperature-dependent clock drift in a controlled environment, and Section V presents the adaptive temperature-resilient time synchronization algorithm. Section VI describes the design and evaluation of a simulator for estimating expected synchronization errors from the empirical data, while Section VII describes the experimental setup and evaluation results.

II. Related work

Time synchronization in low-power and lossy network is often done using a dedicated protocol, for example, FTSP [12] or Glossy [13]. However, unlike TSCH, these protocols are not standardized and interoperable, require additional implementation effort, add to the complexity of the system, and often are not optimized for low radio duty cycles.

Adaptive synchronization for TSCH is first described by Stanislowski *et al.* [6] and was initially implemented in the OpenWSN networking stack. The authors show robustness and high accuracy of their method in indoor, outdoor, and temperature oven experiments: 91 μs using 10 messages per 10 minutes during stable temperature. However, when changes in temperature are detected, the authors disable adaptive time synchronization and send a keepalive message immediately.

Chang *et al.* [14] similarly perform indoor, outdoor, and "sudden change" experiments and achieve less than 100 μs error, but have a high number of messages (600 per 10 minutes) for the sudden change experiment. In a further work, Chang *et al.* [7] investigate how TSCH synchronization is improved by adapting message exchange frequency depending on synchronization quality; they report less than 300 μs error by sending 3.5 messages per 10 minutes on average (when T^o is stable).

As opposed to the present work, both Stanislowski and Chang require that keepalive messages are sent from the downstream node to the upstream. Furthermore, the frequency of these messages is environment-dependent, therefore the medium may be excessively polluted by traffic during rapid environmental condition changes, and the performance of the system (*e.g.*, the required energy budget) cannot be reliably predicted before deployment.

Elsts *et al.* [11] substantially reduce synchronization errors in TSCH networks without compromising energy efficiency. The system achieves that by using high-resolution clocks for scheduling of TSCH operations and estimating synchronization errors, while continuing to use low-resolution timing during low-power mode, made possible by hardware-supported synchronization between the node's low-frequency and high-frequency clock subsystems. The authors report less than 2 μs worst-case error on point-to-point links by synchronizing 150 times per 10 minutes on the average.

Masood *et al.* [15] present DISTY, a dynamic stochastic time synchronization mechanism. Their algorithm is based on Kalman filter, and achieves remarkably low drift prediction error, which in theory could be used to implement a high-accuracy time synchronization protocol: for 80 lost packets, corresponding to 320 seconds without synchronization, DISTY has <30 clock tick synchronization error (with 4 MHz clocks as in our setup that would equal to <8 μs). It is extensively tested in temperature-varying conditions with good results. However, DISTY lacks a real implementation on low-power sensor nodes, and that might be challenging, as the calibration stage of the algorithm requires solving multiple linear regressions with $O(nm^2)$ time complexity (where $n = 100$ and $m = 22$ in the setup presented by the authors) and the continuous operation requires "many computations and higher-energy requirements" according to the authors.

DiStiNCT [16] is a time synchronization scheme for distributed wireless sensor networks that are based on imprecise timers (*i.e.* errors of up to 15 000 ppm). The authors present a power-efficient and computationally simple solution that achieves ms-level time synchronization (one transmission per 3 seconds). Qiu *et al.* [17] present R-Sync, a robust time synchronization scheme for the Industrial IoT that focuses on identifying isolated nodes that have lost their synchronization and pull them back in the network. Dong *et al.* [18] present a secure time synchronization scheme that is resilient to sybil attacks (*i.e.* nodes illegitimately claim multiple identities). None of the above works take into account the temperature-dependent clock drifts.

III. Analytic model

A. Background on TSCH

Due to its time-slotted nature, TSCH requires that all nodes in the network are tightly synchronized. The nodes synchronize their clocks with the coordinator node's clock when joining the network, and keep them synchronized while remaining operational. To avoid partitioning of the network, synchronization is always done from upstream to downstream; in particular, each downstream node marks one or more nodes as its *time sources*. Note that using multiple upstream nodes when they are available (*e.g.*, in dense networks) could help to increase the synchronization quality by averaging out the errors; however, in practice, the existing TSCH open-source implementations (OpenWSN [6] and Contiki [10]) allow to have just a single upstream node as the time source.

Two synchronization methods are defined in the standard [2]: (1) message-based, where the downstream node synchronizes its clock upon the reception of a message from a time source, and (2) ACK-based, where the downstream node corrects its clock using a field in the ACK frame received from a time source.

Both TSCH control messages and data messages are used for synchronization. For example, a downstream node is resynchronized upon each data packet sent to and successfully acknowledged by its time source node. However, for low data rate applications, data packets are not sufficiently frequent to maintain the synchronization; thus, explicit synchronization packets need to be transmitted. This leads to link-layer protocol overhead, *i.e.* nodes transmit more synchronization packets than data packets, and motivates the need to increase the maximum resynchronization period.

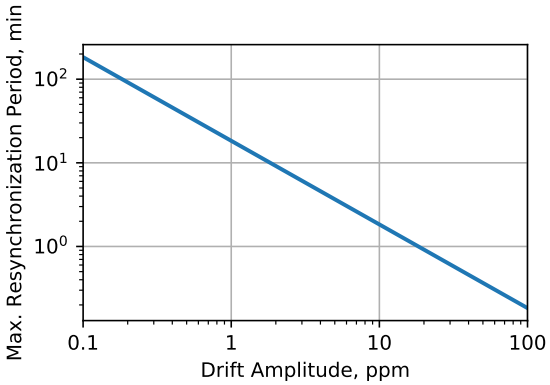


Fig. 1: Maximum resynchronization period depending on the drift amplitude $|\delta|$.

B. Maximum resynchronization period

According to the IEEE 802.15.4 standard, a node must start transmitting each MAC-layer frame exactly τ_o (transmission offset) microseconds after the start of a timeslot. This transmission is preceded by the transmission of the PHY-layer preamble and the Start-of-Frame Delimiter (SFD), which cumulatively takes τ_p microseconds.

TSCH incorporates a guard time to deal with the loss of synchronization. To account for both positive and negative clock drift, the receiver wakes up before the expected start of frame transmission offset and keeps the radio on for at least τ_g microseconds, waiting for reception of an SFD. In the standard, the guard time is equally spaced around the transmission offset τ_o, *i.e.*, the node starts listening at $\tau_o - \frac{\tau_g}{2}$ and ends listening at $\tau_o + \frac{\tau_g}{2}$. It follows that the effective guard time in an implementation that follows the standard to the letter is the asymmetric, since it does not account for the reception of a frame preamble (τ_p).

Given guard time τ_g, the maximal desynchronization for clocks slower than the reference clocks is equal to $e_{max-} = \frac{\tau_g}{2} - \tau_p$, while for clocks faster: to $e_{max+} = \frac{\tau_g}{2}$. The IEEE 802.15.4 standard values ($\tau_g = 2.2\,\text{ms}$, $\tau_p = 160\,\mu\text{s}$) result in $e_{max-} = 0.94\,\text{ms}$ and $e_{max+} = 1.1\,\text{ms}$.

Due to timing errors, TSCH node clocks drift at a rate δ. The synchronization error e after time interval Δt is:

$$e = |\delta|\Delta t. \tag{1}$$

For TSCH to operate without packet loss due to synchronization errors the following inequality needs to hold: $e \leq e_{max-}$, therefore from Eq. 1 one can calculate the maximum resynchronization period for a given guard time (τ_g):

$$\Delta t \leq \frac{\tau_g - 2\tau_p}{2|\delta|}\ . \tag{2}$$

Fig. 1 plots the maximum transmission period for various drift amplitudes. As the drift amplitude increases, more frequent packet transmissions are required to maintain the nodes synchronized.

C. Clock drift due to production spread

Deviation of oscillator crystals from their nominal frequency due to production spread is one of the main reasons for clock drift. The bounds of this deviation ($\pm\epsilon_f$ ppm) are typically specified by the manufacturer.

Let us assume that the timings of a sender, u, and a receiver, v, are scheduled using crystals with frequency errors $\epsilon_u \in [-\epsilon_f, \epsilon_f]$ and $\epsilon_v \in [-\epsilon_f, \epsilon_f]$ respectively. The worst case scenario is when one of the crystals operates with $+e_f$ error, whilst the other operates with $-e_f$ error. As a result, the relative drift due to production spread is:

$$\delta_f = \left(\frac{1}{1+\epsilon_u} - \frac{1}{1+\epsilon_v}\right) \approx \epsilon_u - \epsilon_v, \text{ where } |\delta_f| \leq 2\epsilon_f\ . \tag{3}$$

For example, a TSCH link between nodes that use the a crystal oscillator with $\epsilon_f = \pm 20\,\text{ppm}$ must be able to tolerate a drift of up to $\delta_f = 40\,\text{ppm}$ in the worst case scenario.

D. Clock drift due to differences in operating temperature

Crystal oscillators are also characterized by a temperature-dependent error that depends on the shape of the crystal. Crystal oscillators are typically manufactured in such a way that their frequency dependence on temperature is quadratic. Indeed, a crystal resonates close to its nominal frequency at $T_0 = 25\,^o\text{C}$, but slows down at temperature T at a rate of $B(T-T_0)^2$, where B is the parabolic coefficient. The worst case scenario for a TSCH link is when one of the crystals operates at a temperature T_0, whilst the other operates at a temperature T. The temperature-specific drift δ_T in this worst case scenario is:

$$\delta_T = \left(\frac{1}{1-B(T-T_0)^2} - 1\right) \approx B(T-T_0)^2\ . \tag{4}$$

For the 32.768 kHz crystal oscillator FC-135 ($T_0 = 25\,^o\text{C}$, $B = -0.04\,\text{ppm}$ [19]), the temperature-dependent clock drift is $\leq 1\,\text{ppm}$ in room temperatures (20 to $30\,^o\text{C}$), whereas at $-5\,^o\text{C}$ the drift for this crystal rises up to $\delta_T = 36\,\text{ppm}$.

E. Other sources of clock drift

The synchronization error cannot be detected with perfect accuracy due to inaccuracies in packet timestamps, packet transmissions, scheduling of the TSCH state machine, and granularity of the units of the timestamps. These problems are source for additional clock drifts, denoted by δ_o. As shown in [11], δ_o is very important for maintaining microsecond-level accuracy synchronization. Yet, it is inversely proportional to the period of synchronization (Δt) [11]; hence, it is less significant in applications that generate data infrequently.

Another possible origin of clock drift is the switching of clock sources. In particular, during the active mode, TSCH timing is often based on a high-frequency oscillator, while during low-power modes, the only available active clock source is a low-frequency one, such as the FC-135 oscillator. However, this paper avoids that problem by resynchronizing these two local time sources on each wakeup [11].

F. Avoiding the transmission of synchronization messages

In a practical scenario, the various sources of clock drift are combined. In fact, they can be aggregated or cancel out, depending on the direction of the drift.

$$\delta = \delta_f + \delta_T + \delta_o\ . \tag{5}$$

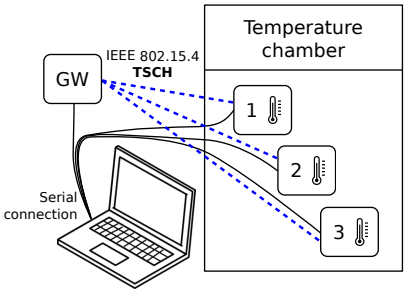


Fig. 2: Experimental setup. GW denotes the network's coordinator.

Eq. 1 indicates an inversely proportional relationship between the clock drift (δ) and the period of synchronization (Δt). In other words, assuming a fixed guard time, a TSCH network has two ways of maintaining the synchronization: either the period of synchronization needs to match δ; or, the drift needs to be compensated in software. Clearly, the second alternative is more desirable because of efficiency reasons, in particular for TSCH in applications that exchange data messages infrequently and consequently need dedicated control messages for synchronization. For example, assuming $\Delta t = 600$ sec and the default TSCH guard time ($\tau_g = 2.2$ ms, $e \leq 0.94$ ms), the clock drift needs to be compensated in software so that the effective drift is $|\delta| \leq 1.57$ ppm.

Related works have proposed algorithms for measuring and compensating the clock drift due to production spread (δ_f) [6] and measurement inaccuracies (δ_o) [11]. This article complements the related work with a mechanism to measure and compensate the temperature-dependent clock drift (δ_T).

IV. Experimental Study and Calibration

A. Setup

Temperature chamber TAS LT 600 (Fig. 3) is used to investigate the drift dynamics under T^o changes. Three sensor nodes are placed inside the chamber, while the network coordinator node is placed outside, where the temperature stays stable. All nodes synchronize directly to the coordinator, which transmits synchronization messages once per second. Nodes in the chamber have USB connections with a laptop, which is used to log synchronization errors. The errors are logged upon each packet reception, while resynchronization is done only once every 10 min in order to simulate a less frequent exchange of packets.

B. Calibration

A calibration step is first performed to learn drift values under various temperature settings. The nodes are cooled down to $-5\,^o$C, then heated up to $+60\,^o$C during a calibration period of 2 h 40 min. Upon reception of each synchronization message each node calculates, saves, and logs over the serial interface the average drift values during the last 12 sec, as well as the corresponding T^o. The HDC1000 temperature sensor, which has an accuracy of $\pm 0.2\,^o$C, is read once per second by a background process. The measurement of the empirical drift is based on the FC-135 crystal oscillator. Thus, he timing

Fig. 3: The temperature chamber.

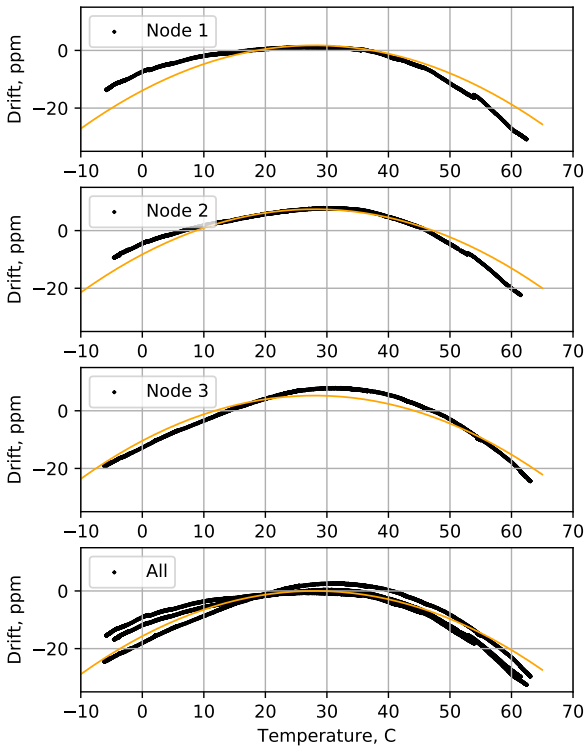


Fig. 4: Calibration results for drift depending on T^o. The narrower line is a fit of the parabola from the analytic model ($B = -0.02$, $T_0 = 28\,^o$C; y-axis offset individually fit for each node to minimize the absolute error).

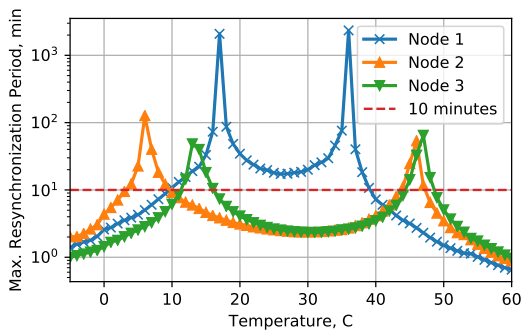


Fig. 5: Maximum resynchronization period based on the empirical drift amplitude. The peaks correspond to circumstances when the drift due to production spread and the temperature-dependent drift cancel out. The dashed line corresponds to the 10-minute threshold targeted in this work.

measurements have a quantization error of up to 30.5 μs [11].

Calibration data (Fig. 4) validate the parabolic shape of the temperature-dependent error curve; however, the parabolic coefficient B of the datasheet corresponds to a worst case scenario, and thus cannot be used in place of empirical measurements if good results are desired (*i.e.*, <1.57 ppm drift, see Section III-F). Indeed, Fig. 4 shows that the dependence is smaller than specified by the datasheet ($B = -0.04$ *vs* $B \approx -0.02$ empirically) and the shape of curve does not exactly match that of a quadratic parabola.

Fig. 5 shows the result of equating the drift amplitude, δ, with the empirical drift of Fig. 4 on Eq. 2, and demonstrates the maximum resynchronization period required for maintaining time synchronization. The peaks identify the circumstances when the drift due to production spread and the temperature-dependent drift cancel out. The horizontal dashed line corresponds to a resynchronization threshold of 10 minutes. Node 1 performs particularly well in temperatures between 10 ^{o}C and 40 ^{o}C. Node 2 and Node 3, on the other hand, require more frequent synchronization even in room temperatures. The results highlight the fact that every individual node is unique. Therefore, temperature compensation based on datasheet estimates is insufficient for effective time-synchronization. Instead, using the proposed temperature-resilient time synchronization algorithm, each node measures and compensates for its individual drift behavior.

Algorithm 1 Adaptive temperature-resilient synchronization

▷ Initialization section; executed when joining the TSCH network
function ONJOINTSCHNETWORK
  $\delta_{estimated} \leftarrow 0$ ▷ Initialize the drift estimate
  $t_{sync} \leftarrow$ TIMENOW( ) ▷ Initialize the last synch. time
end function

▷ Executed on every active timeslot
function ONACTIVETSCHTIMESLOT($ts_i, isRx_i$)
▷ ts_i – the start of the i-th timeslot
▷ $isRx_i$ – whether the i-th timeslot is for Rx
  $e_{sync} \leftarrow 0$ ▷ Reset synchronization error
  $T \leftarrow$MEASURETEMPERATURE( ) ▷ Get current temperature

  if $isRx_i$ **then** ▷ On reception timeslot
    $p \leftarrow$ TRYRECEIVEPACKET( )
    ▷ Check if packet was received and is usable for timesync
    if $p \neq NULL$ AND ISTIMESOURCE($p.src$) **then**
      ▷ The drift learning step
      $t_{expected} \leftarrow ts_i + TschTxOffset$
      $t_{actual} \leftarrow p.SFDtimestamp$
      $t_{now} \leftarrow$ TIMENOW( ) ▷ Get current time
      $e_{sync} \leftarrow t_{actual} - t_{expected}$ ▷ Set the sync error
      $\Delta_{sync} \leftarrow t_{now} - t_{sync}$
      $\delta_{current} \leftarrow e_{sync} \div \Delta_{sync}$
      $\delta_{estimated} \leftarrow$ MOVINGAVERAGEADD($\delta_{current}$)

      if $isCalibrationStage =$ TRUE **then**
        STOREDRIFT(T, $\delta_{estimated}$)
      end if
      $t_{sync} \leftarrow t_{now}$
    end if
  else ▷ On transmission timeslot
    TRYTRANSMITPACKET(GETPACKET( ))
  end if
  ▷ Find the time and type of the next active timeslot
  $(ts_{i+1}, isRx_{i+1}) \leftarrow$ TSCHFINDNEXTTIMESLOT(ts_i)

  if $isCalibrationStage =$ FALSE **then**
    $\delta_T \leftarrow$ LOOKUPDRIFT(T)
    $\Delta t \leftarrow ts_{i+1} - ts_i$ ▷ Time to next active timeslot
    $c_T \leftarrow \Delta t \times \delta_T$ ▷ Temperature compensation
    $c_{history} \leftarrow \Delta t \times \delta_{estimated}$ ▷ History-based compens.
    ▷ Correct for the error and for both compensations
    $ts_{corr} \leftarrow ts_{i+1} - (e_{sync} + c_T + c_{history})$
  else
    ▷ Correct for the error only
    $ts_{corr} \leftarrow ts_{i+1} - e_{sync}$
  end if
  ▷ Reschedule this function for the next active timeslot
  SCHEDULE(ONACTIVETSCHTIMESLOT, ts_{corr}, $isRx_{i+1}$)
end function

V. ADAPTIVE TEMPERATURE-RESILIENT TIME SYNCHRONIZATION

A. The algorithm

Algorithm 1 describes the essential operation of the adaptive temperature-resilient time synchronization method. (For simplicity, the synchronization on ACK is not shown.) The combined empirical data from drift estimates and temperature readings (Fig. 4) are used to construct a node-specific lookup table with average drift values per each point in the ^{o}C scale. In the code, $isCalibrationStage$ is an external parameter that determines whether the calibration is finished; STOREDRIFT and LOOKUPDRIFT handle the operations related to the lookup table that is a mapping between temperature and drift.

Upon each resynchronization event, the node learns the timing error of the local clock, therefore is able to estimate the local clock drift by dividing the error in clock ticks with the number of ticks passed since the last time synchronization. To remove errors caused by imprecise measurements, a moving average filter is applied on several recently learned drift values, resulting in a more accurate cumulative estimate [11].

In the calibration stage, the estimated drift is stored in the lookup table. Once this stage is completed, the lookup table is used to obtain $c_{temperature}$ — the compensation for

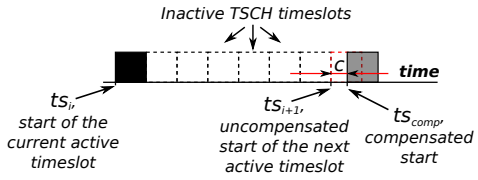


Fig. 6: Compensating for clock drift. Compensation amount c is equal to the time until the next active slot times the estimated drift.

the temperature drift. $c_{temperature}$ is used continuously, even during the slots when no packet is received from a time source. However, when a packet *is* received, it is used to dynamically adjust for the second-order error, *i.e.*, the difference between the actual drift and the drift in the table, and to calculate a new value of $c_{history}$, the history-based compensation. The sum of these two compensations ($c_{temperature} + c_{history}$) is used to adjust ts_{i+1}, the time of the next timeslot (Fig. 6).

The implementation uses error-accumulation free fixed-point arithmetic, and stores drift and time with high granularity (in units of $\frac{1}{1024}$ ppm and $\frac{1}{1024}\,\mu$s respectively) to achieve high accuracy [11]. Lastly, since the TSCH timeslot operation is done in an interrupt context, lengthy function calls must be avoided; therefore the temperature sensor is read by a background process and MEASURETEMPERATURE rapidly returns the most recently read value instead of physically accessing the sensor.

B. Online calibration

The requirement to use a temperature chamber might make the system impractical to use in certain conditions. However, a simple extension of the algorithm allows to replace the separate calibration step by online calibration, performed on demand. A sketch of that extension follows.

Here, the last calibration time is stored in an array, with separate for each temperature value (*e.g.*, for each ^{o}C). Each time a node encounters a specific temperature value it is not yet calibrated for, it enters a calibration mode. During this mode, time synchronization is performed more frequently or the guard time is increased. After learning the drift for this specific temperature value, the node records the drift, and the current time as the last calibration time for the current temperature. Then it reverts back to a more energy-efficient mode of operation. In this way, the node only ever needs to calibrate for temperature values it encounters in the real environment. This online calibration may be periodically repeated, *e.g.*, because the aging of the oscillator crystal changes its drift.

VI. SIMULATOR

A simulator is designed to validate the analytic results and show the behavior of Algorithm 1. The simulator takes a series of temperature values and a lookup table with calibration data (per-temperature drifts) as its two inputs and models the behavior of a receiver node exposed to these environmental conditions.

The input temperature values are assumed to be the ground truth of the air temperature around the node, and the lookup table of drifts – the ground truth of the temperature-dependent clock frequency changes. In order to generate synchronization errors, four principal classes of errors are introduced in the simulator:

- Errors in temperature measurements. The HDC1000 temperature sensor has $\pm 0.2\,^o$C accuracy [9], therefore a random uniform error in this range is added to its readings.
- Delay in drift estimation and inertia in temperature. The calibration reports the average drift during the last 12 sec (Section IV-B). Furthermore, the crystal oscillator can be assumed to be heating up more slowly than the air around the node as measured by the sensor, and lagging a short time behind it. Therefore, for drift reported at time t, the simulations use T^o at $t - 10$ seconds as the operating T^o of the oscillator.
- Sub-degree differences in temperature that are not captured by the compensation table as the implementation uses a table with granularity of $1\,^o$C. The compensation algorithm rounds down the measured temperature T to the nearest integer T', which is then used to obtain the drift estimate $d(T')$. However, the simulations use linearly weighted combination of $d(T')$ and $d(T'+1)$ as the effective drift at temperature T.
- Errors in time measurements. According to analytic models [11], time is measured with accuracy of one timer tick ($0.25\,\mu$s in our setup), therefore a random uniform error in this range is added to the synchronization error calculated upon packet reception.

The results (using input data from Section IV) are shown in Fig. 7. Each simulation is re-run three times in order to show the small effects from randomness.

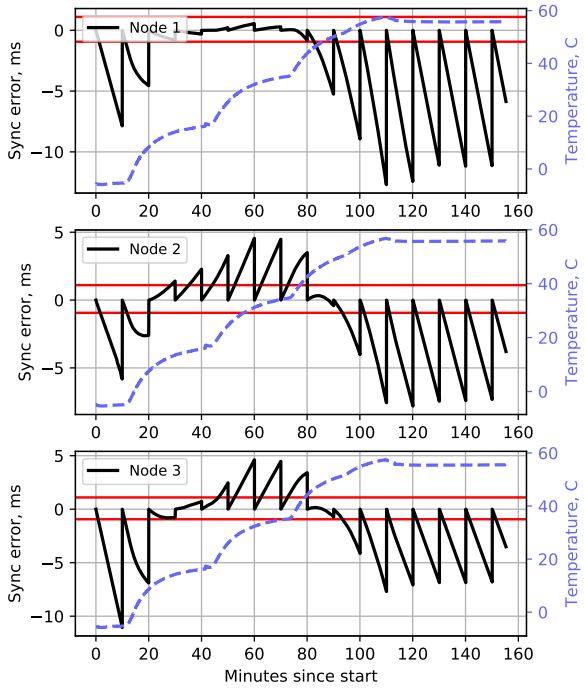
Figures 7c and 7d show an order-of-magnitude improvement in synchronization quality after applying Algorithm 1. The maximum synchronization error ($|err_{max}|$) is 12.7 ms in the baseline TSCH (Fig. 7a), but only 0.7 ms when temperature-based compensation is enabled (Fig. 7d): an difference of more than 10 times. Counterintuitively, disabling history-based compensation in Algorithm 1 gives the smallest average error (Fig. 7d). This happens because otherwise second-order errors arise from the interaction of the two drift compensation mechanisms. However, there are some practical reasons for enabling history-based compensation. In particular, aging processes of oscillator crystals change their drift, therefore the lookup tables will eventually become outdated unless the calibration is periodically repeated (*e.g.*, once per year).

Figure 8 shows that the algorithm is relatively robust to temperature dynamics as long as the rate of temperature change stays realistically small.

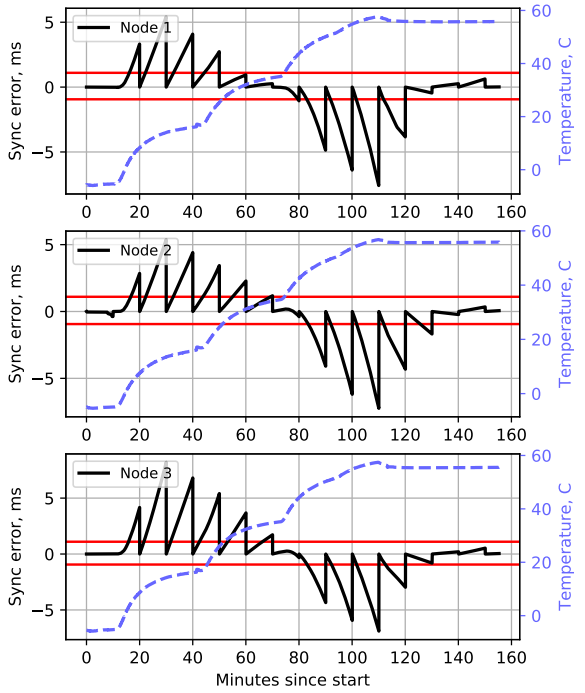
VII. EXPERIMENTAL EVALUATION

A. Synchronization accuracy: controlled experiment

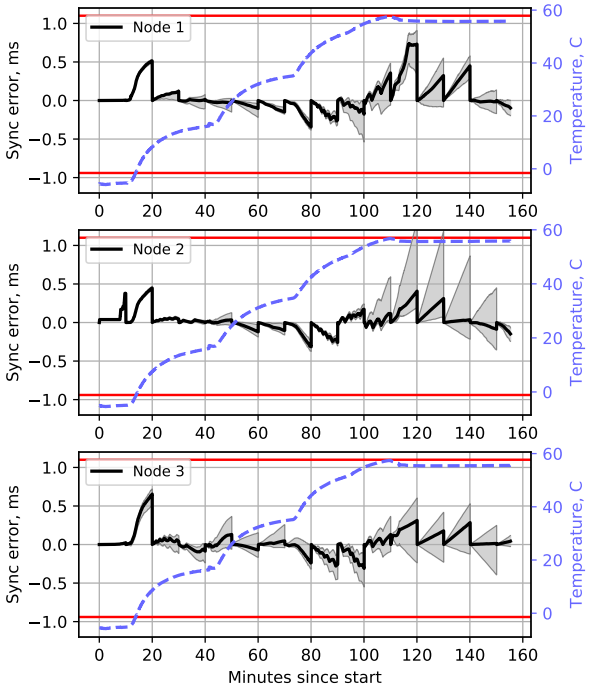
An experiment in the temperature chamber is performed in order to validate the simulation results. The setup in Fig. 2 is replicated. After cooling down the nodes to $-5\,^o$C the system is started up; then the chamber's temperature setting is incremented in three steps of $+20\,^o$C, stopping for 30 min between subsequent increments (at $+15$, $+35$ and $+55\,^o$C).



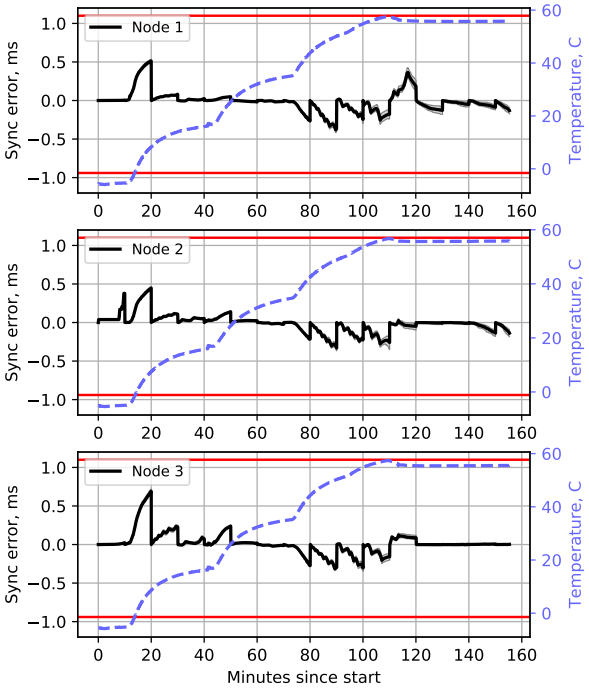
(a) Baseline TSCH, no compensation. $|err|_{max} = 12.7$ ms, $|err|_{mean} = 2.58$ ms



(b) With history-based compensation. $|err|_{max} = 8.2$ ms, $|err|_{mean} = 1.36$ ms



(c) With temperature-based compensation and history-based compensation (Algorithm 1). $|err|_{max} = 1.32$ ms, $|err|_{mean} = 0.11$ ms



(d) With temperature-based compensation only. $|err|_{max} = 0.72$ ms, $|err|_{mean} = 0.08$ ms

Fig. 7: Simulation performance; each simulation repeated 100 times, the average error plotted with thick black line, the distribution of errors shown as a colored area. **Here and in further figures:** synchronization period is 10 min; the horizontal lines mark the standard TSCH desynchronization thresholds. In **(a)**, the network would desynchronize in 51 seconds in the worst case, in **(b)**: in 106 seconds, in **(c)**: in 9 min 19 seconds. **(d)** always stay within the synchronization interval (its estimated desynchronization period is 15 minutes), as does **(c)** in the majority of simulations. Note the different y-axis scales.

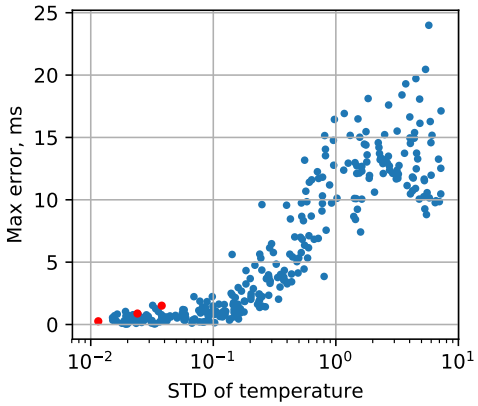


Fig. 8: Dependence of the maximal error on temperature dynamics. Results obtained on synthetically generated temperature curves where the temperature from second to second changes according to the normal distribution with standard deviation σ given on x-axis. For each simulation, starting temperature t_0 randomly selected in range $10 \leq t_0 \leq 40\,^o$C. **Empirical datapoints from the indoor, temperature chamber, and outdoor experiments plotted red.**

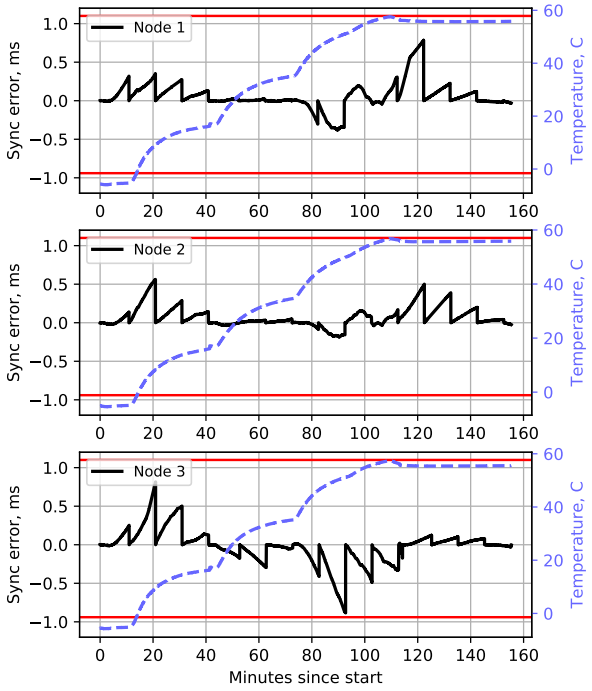


Fig. 9: Experimental results in the chamber. Settings as in Fig. 7c. The nodes stay within the interval at all times: $|err|_{max} = 0.88$ ms, $|err|_{mean} = 0.11$ ms.

The results (Fig. 9) closely repeat the simulations (Fig. 7c). Firstly, they both show a similar pattern of synchronization errors. Secondly, the disagreement in the average error observed in simulations and experiments is only 5 %. The system remains within the guard-time boundaries of the standard TSCH settings when synchronizing once every 10 minutes. Furthermore, it can be seen that once the temperature is stable at $+55\,^o$C, the synchronization accuracy iteratively improves because of the history-based drift compensation.

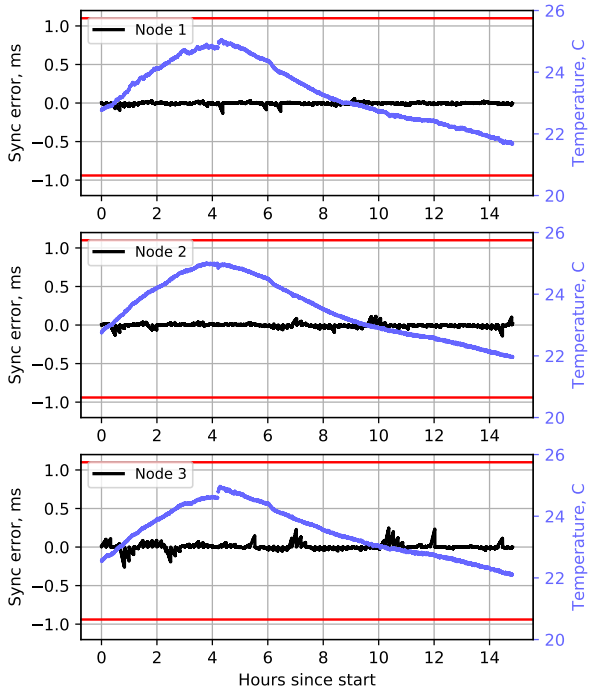


Fig. 10: Experimental results, 15 h indoors. Settings as in Fig. 7c. $|err|_{max} = 0.26$ ms, $|err|_{mean} = 0.017$ ms.

B. Synchronization accuracy: indoor and outdoor experiments

To evaluate longer-term operation of the system in realistic conditions, one 15 h indoor experiment and one 15 h outdoor experiment are performed with a similar setup.

The indoor experiment (Fig. 10) shows relatively stable temperature and low synchronization errors ($|err|_{max} = 0.26$ ms). Here, the resynchronization interval could be increased by 3.5 times to more than 30 min. It also shows that the maximum synchronization error in room temperature is heavily dependent on the node.

The outdoor experiment was undertaken on the 19th June 2017 in the UK. According to meteorological data, it was one of the warmest days of the year. The experiment included one period when the nodes, except the coordinator node, were exposed to direct sunlight. During the experiment the nodes were protected only by light off-the-shelf plastic casing (see [8]). The temperature curve shows stable increase during the morning heat-up phase, after which a plateau of $> 45\,^o$C is reached. However, during this plateau, significant rapid variations in temperature are present, caused by minor clouds and wind, and the synchronization errors are much larger than during the stable heat-up phase. Then, after being shadowed by a larger cloud, the temperature on the nodes drops sharply (more than $8\,^o$C change per 5 minutes) below $< 40\,^o$C. Only during this rapid change the guard time limit is breached; a synchronization frequency of 6 minutes would be sufficient to stay within it. Afterwards the nodes are shadowed by a building and the system remains stable until the end of the experiment.

The results stress the importance of real-world experiments, as the node outdoor temperature dynamics show unexpectedly

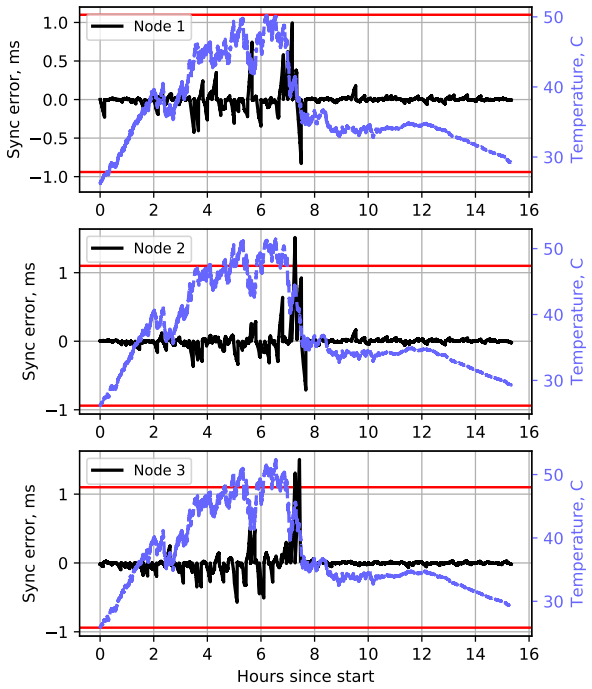


Fig. 11: Experimental results, 15 h outdoors. Settings as in Fig. 7c (except history buffer size: 1 sample). $|err|_{max} = 1.51$ ms, $|err|_{mean} = 0.055$ ms.

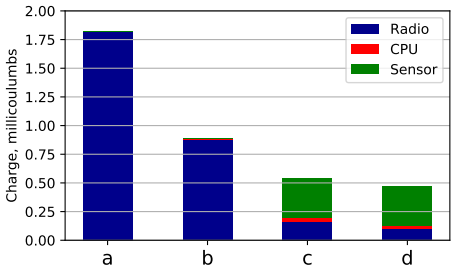


Fig. 12: Charge consumption breakdown in a 600-second period. The options *a*, *b*, *c*, and *d* correspond to the four methods in Fig. 7: (a) default TSCH, (b) adaptive time synchronization [6], (c) adaptive time synchronization combined with the proposed temperature-resilient synchronization, and (d) the proposed temperature-resilient synchronization. The application receives five packets in each desynchronization period. Radio and CPU results estimated from current level measurements with a RocketLogger [20] device; time is measured in software; sensor current consumption taken from datasheet [9].

high randomness, and confirm direct sunlight as a particularly difficult environment for time-synchronized networks.

C. Energy efficiency

Figure 12 shows the charge required for synchronization in a 600-second period on the SPES-2 platform depending on the synchronization method. "Sensor" refers to the charge requirements of the HDC1000 temperature sensor, sampled once per second; "CPU" to the cost of running the adaptive and temperature compensation algorithms, also once per second; "Radio" to the energy costs for receiving TSCH EB synchronization packets (36 bytes each).

The number of packets per 600 seconds is dependent on the desynchronization period, which in turn is dependent on the method. Note that assuming a realistic 80 % link-layer packet reception rate, the application needs to try to receive more than one packet per each desynchronization period in order to receive at least one successfully with high probability. These experiments assume that $> 99.9\,\%$ probability of remaining synchronized in a single desynchronization period is required, therefore five packet reception attempts must be scheduled. Consequently, within the 10-min period, method (a) requires receiving 59 packets, while (d): just 3.3 packets on the average.

The results show that even though reading the temperature sensor adds a constant overhead to the system, the temperature compensation helps to increase the overall system efficiency in this scenario. The relative efficiency would be further increased in a multihop network formed by energy-constrained nodes, as in this case not only the costs for reception of synchronization packets, but also the costs for retransmission would have to be taken into account. Furthermore, the relative efficiency would be further increased if the overlaying application makes use of the temperature sensor. In that case, the proposed algorithm could leverage these temperature measurements, not requiring any additional sensing.

VIII. Conclusion

This paper proposes *adaptive temperature-resilient time synchronization*, a method for improving time synchronization accuracy in TSCH networks exposed to large dynamic changes in temperature. The results show that this method reduces synchronization message frequency by up to 10 times compared to baseline TSCH. With one synchronization message per ten minutes, the system shows a 0.26 ms maximal synchronization error indoors and a 0.88 ms error in a temperature chamber where the network undergoes a 60 ^{o}C change in temperature. In both experimental settings, all nodes stay within the standard TSCH guard time that bounds the maximum error to 0.94 ms. In outdoor settings and direct sunlight where the rate of change exceeds 8 ^{o}C per five minutes, a 6-minute synchronization period is sufficient to achieve the same result. Finally, both the calibration stage and the continuous operation stage of the method has low computational and sensing requirements, therefore can be easily implemented on low-power Internet of Things nodes equipped with common-off-the-shelf temperature sensors.

References

[1] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, 2014.

[2] "IEEE Standard for Local and metropolitan area networks—Part 15.4," IEEE Std 802.15.42015, 2015.

[3] K. Brun-Laguna, A. L. Diedrichs, D. Dujovne, R. Léone, X. Vilajosana, and T. Watteyne, "(Not so) intuitive results from a smart agriculture low-power wireless mesh deployment," in *Proc. 11th ACM Workshop on Challenged Networks*. ACM, 2016, pp. 25–30.

[4] K. S. Low, W. N. N. Win, and M. J. Er, "Wireless sensor networks for industrial environments," in *Proc. 2005 International Conference on Computational Intelligence for Modelling, Control and Automation*, vol. 2. IEEE, 2005, pp. 271–276.

[5] A. Mavromatis, G. Z. Papadopoulos *et al.*, "Impact of guard time length on IEEE 802.15.4e TSCH energy consumption," in *Proc. 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2016, pp. 1–3.

[6] D. Stanislowski, X. Vilajosana, Q. Wang, T. Watteyne, and K. S. Pister, "Adaptive synchronization in IEEE802.15.4e networks," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 795–802, 2014.

[7] T. Chang, T. Watteyne, K. Pister, and Q. Wang, "Adaptive synchronization in multi-hop TSCH networks," *Elsevier Computer Networks*, vol. 76, pp. 165–176, 2015.

[8] X. Fafoutis, A. Elsts, A. Vafeas, G. Oikonomou, and R. Piechocki, "Demo: SPES-2 A Sensing Platform for Maintenance-Free Residential Monitoring," in *Proc. 2017 International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2017, pp. 240–241.

[9] "HDC1000: Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor," www.ti.com/lit/ds/symlink/hdc1000.pdf.

[10] S. Duquennoy, A. Elsts, B. Nahas, and G. Oikonomou, "TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation," in *Proc. 2017 International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2017.

[11] A. Elsts, S. Duquennoy, X. Fafoutis *et al.*, "Microsecond-accuracy time synchronization using the IEEE 802.15.4 TSCH protocol," in *Proc. IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*. IEEE, 2016, pp. 156–164.

[12] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proc. 2004 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 39–49.

[13] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *Proc. 2011 10th International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2011, pp. 73–84.

[14] T. Chang and Q. Wang, "Adaptive compensation for time-slotted synchronization in wireless sensor network," *International Journal of Distributed Sensor Networks*, 2014.

[15] W. Masood, J. F. Schmidt, G. Brandner, and C. Bettstetter, "DISTY: Dynamic Stochastic Time Synchronization for Wireless Sensor Networks," *IEEE Trans. Ind. Informat.*, 2016.

[16] J. A. Boyle, J. S. Reeve, and A. S. Weddell, "DiStiNCT: Synchronizing Nodes With Imprecise Timers in Distributed Wireless Sensor Networks," *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 938–946, 2017.

[17] T. Qiu, Y. Zhang *et al.*, "A Robust Time Synchronization Scheme for Industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. PP, no. 99, pp. 1–1, 2017.

[18] W. Dong and X. Liu, "Robust and Secure Time-Synchronization Against Sybil Attacks for Sensor Networks," *IEEE Trans. Ind. Informat.*, vol. 11, no. 6, pp. 1482–1491, Dec 2015.

[19] "FC-135R / FC-135 kHz range crystal unit," https://support.epson.biz/td/api/doc_check.php?dl=brief_FC-135R_en.pdf.

[20] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele, "Measurement and Validation of Energy Harvesting IoT Devices," in *Proc. 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, 2017.

Atis Elsts (M'17) received his doctoral degree in computer science from the University of Latvia, in 2014, based on research done at the Institute of Electronics and Computer Science (IECS/EDI, Riga, Latvia). Since 2016 he is a researcher in the SPHERE interdisciplinary research collaboration at University of Bristol. Prior to that, he was a researcher at Uppsala University (2014-2015) and at the Swedish Institute of Computer Science (SICS, 2015). His scientific interests focus on experimental research in Networked Embedded Systems, including network protocols, time synchronization, operating systems and tools. He is a maintainer of the Contiki-NG operating system for the Internet of Things (IoT).

Xenofon Fafoutis (S'09-M'14) received a PhD degree in Embedded Systems Engineering from the Technical University of Denmark in 2014; an MSc degree in Computer Science from the University of Crete in 2010; and a BSc in Informatics and Telecommunications from the University of Athens in 2007. Since 2014, he is a researcher with the University of Bristol and a core member of the EPSRC SPHERE interdisciplinary research collaboration. His research interests lie in Networked Embedded Systems as an enabling technology for Digital Health, Smart and Healthy Cities, and the Internet of Things (IoT).

Simon Duquennoy is a research scientist at RISE SICS (Sweden). He obtained his PhD from Universit de Lille 1 (France) in 2010, and was then granted an ERCIM Alain Bensoussan fellowship for a post-doc at SICS. He then obtained then a permanent researcher position at SICS, in addition to which he worked at Inria (France) in 2016-2017. His research focuses on communication in the IoT, MAC, routing, security, dependability, and embedded systems design. He is a TPC member of internationally recognized conferences (ACM SenSys, ICDCS, EWSN, and MASS). He has over 35 peer-reviewed papers including many at flagship venues (ACM SenSys, ACM/IEEE IPSN, ACM TOSN and ACM Emsoft). Simon has experience as a principal investigator in several European and national projects. He is a co-founder of Contiki-NG, an open-source operating system for the IoT. Further, he is active in standardization through the IETF 6TiSCH, 6LO and LWIG Working Groups.

George Oikonomou received the MSc Degree in Information Systems and the PhD degree in computer science from the Athens University of Economics and Business, Athens, Greece, in 2002 and 2009 respectively. He is currently a Lecturer with the Department of Electrical and Electronic Engineering at the University of Bristol. He has previously worked as a Research Associate at the Faculty of Engineering, University of Bristol and at the Computer Science department, Loughborough University, UK. His current research focuses on energy-efficient networking for networks of severely constrained wireless embedded devices and the Internet of Things. Dr Oikonomou is a co-founder, steering group member and maintainer of Contiki-NG, and an active developer of the Contiki open source embedded operating system for the Internet of Things.

Robert Piechocki received an MSc degree from Technical University of Wroclaw (Poland) in 1997 and a PhD degree from the University of Bristol in 2002. He is currently a reader in Advanced Wireless Access and a member of Communications Systems and Networks group. His research interests span the areas of Statistical Signal Processing, Information and Communication Theory, Wireless Networking, Body and ad-hoc networks, Ultra Low Power Communications and Vehicular Communications. He has published over 100 papers in international journals and conferences and holds 13 patents in these areas. For the SPHERE project Robert is leading the development of wearable and wireless technologies.

Ian Craddock (M'09-SM'10-F'16) is currently a full Professor with the University of Bristol (UK) and Director of the flagship "SPHERE" centre (www.irc-sphere.ac.uk) comprising approximately 100 researchers and clinicians working on IoT technology for health. He has been working in healthcare technology for 15 years and founded a company that is currently completing trials of a CE-marked breast imaging device based on his research. He has published over 150 papers and serves on the healthcare strategy board for the UKs largest engineering funder. He is also separately employed by Toshiba as Managing Director of their Telecommunications Research Lab in Bristol, responsible for a portfolio of both internal and collaborative communications, IoT and smart city research.