# Clock Drift Prediction for Fast Rejoin in 802.15.4e TSCH Networks

Timothy Claeys, Franck Rousseau, Bernard Tourancheau, Andrzej Duda

# Clock Drift Prediction for Fast Rejoin in 802.15.4e TSCH Networks

Timothy Claeys, Franck Rousseau, Bernard Tourancheau, Andrzej Duda
University Grenoble Alps,
Grenoble Institute of Technology, CNRS Grenoble Informatics Laboratory, Grenoble, France.
Email: firstname.lastname@imag.fr

***Abstract*—In this paper, we propose a fast predictive resynchronization scheme that allows nodes to rejoin a 802.15.4e TSCH network with which they were previously synchronized. The idea of the scheme comes from the investigation of the internal and external root causes of the clock drift between two nodes—we have identified the causes and proposed a means for its estimation. Based on the prediction of the drift between any pair of nodes, we are able to determine the instant at which a node needs to wake up after desynchronization, thus reducing its energy consumption and the rejoin latency.**

**We compare the proposed scheme with other state-of-the-art proposals through experiments on different hardware platforms. The experimental results show that desynchronized nodes rejoin up to 34 times faster and consume less energy by a factor of 1000 compared to the other proposals.**

## I. Introduction

There is a growing need for network protocols that meet the requirements for supporting the Internet of Things (IoT). The IEEE802.15.4e standard defined three new MAC-layer protocols [1]: i) the Deterministic and Synchronous Multichannel Extension (DSME) that uses the same beacon format as IEEE802.15.4 and adds channel hopping, ii) Low Latency Deterministic Networks (LLDN), and iii) Time Slotted Channel Hopping (TSCH). TSCH defines a time synchronized network with dedicated timeslots to avoid collisions, and channel hopping, with maximum 16 frequency channels, for high reliability. Timeslot synchronization is important because the operation of the MAC layer relies on the precise time alignment of the timeslots. Clock imperfections may therefore lead to synchronization issues. To mitigate such problems, TSCH specifies a default scheme to maintain synchronization: nodes take advantage of the timing information on frame arrivals to calibrate their clocks. Stanislowski et al. [2] extended this default scheme by introducing the concept of *drift awareness* in which nodes learn how their clocks drift with respect to their neighbors to dynamically change the amount of traffic required for a network to stay synchronized.

Synchronized operation and channel hopping have made TSCH one of the most promising solutions for a wide range of IoT applications (e.g. industrial, automotive, e-health applications) [3]. A fast network rejoin procedure lets TSCH support truly dynamic networks with high churn when nodes often leave and rejoin the network, e.g. the Floating Sensor Network (FSN) project built at UC Berkeley [4] or energy scavenging networks where nodes need to rejoin after the batteries are recharged. Currently, the only way to realign the timeslots of a node that wants to rejoin the network is to capture a specific frame, called an Enhanced Beacon (EB), containing synchronization information. If the network traffic is sparse and spread over multiple frequency channels, the rejoining node may have to scan actively for long time periods, which may be unfeasible for nodes with small batteries or energy harvesting. The current state of the art in this domain [5], [6] has mainly focused on optimizing the bootstrap synchronization phase in which uninitialized nodes want to join a network.

In this paper, we first study in detail the internal and external root causes of the clock drift between two nodes. We begin with an analysis of the internal root causes by measuring the clock frequency variation on three hardware platforms, namely TelosB Crossbow [7], OpenMote-CC2538 [8], and GreenNet [9]. We investigate its effect on the clock drift and model the internal as well as external root causes of the drift. The investigation shows that a pair of nodes has a stable relative drift that depends only on the stability of the clock frequency. Moreover, platforms with a high frequency (HF) clock stay synchronized longer because the fluctuations in the clock frequency caused by the *production spread* (small imperfections in the crystal and the surrounding electrical components [3]) have a smaller impact on the relative drift. Furthermore, we study the impact of external drift root causes such as temperature and input voltage to show that we can use the relationship between the external causes and the frequency of the crystal to compensate for an unstable relative drift.

Based on these investigations, we propose a new three phase scheme for rejoining the network with a minimal latency and energy consumption. In the first phase, when a node is still synchronized, it predicts the relative drift caused by internal and external sources with respect to its *time source neighbors*, the nodes whose timing information is used to calibrate the clock. The node also tracks the number of elapsing slots by using a low power timer. In the second phase, when the node becomes desynchronized, it can find the current network timeslot in an accurate manner by combining the drift prediction and the low power timer value. Finally, the node computes the right frequency channel based on the timeslot parameters stored in the persistent memory. The node uses a radio reception (Rx) window to capture passing frames. The length of the Rx window depends on the time during which the node was desynchronized and on the different types of drift

sources. The window grows steadily to account for unforeseen drift factors.

We have implemented the proposed scheme in the OpenWSN stack[1] and evaluated its performance on three different hardware platforms. The experimental results show that our scheme outperforms the existing joining schemes. It achieves a faster rejoin latency and consumes much less energy because it accurately predicts the time location of frames transmitted by the time source neighbor and thus spends less time in radio active mode.

We start the paper with the background on IEEE802.15.4e TSCH in Section II. Section III presents the state of the art of the research relative to our work. Section IV shows the investigation results on modeling the sources of the clock drift and Section V introduces the proposed scheme. Section VI reports on the experimental results of the performance evaluation and Section VII concludes the paper.

## II. IEEE802.15.4e TSCH: An Overview

This section gives an overview of the IEEE802.15.4e TSCH protocol and discusses in detail its synchronization methods.

### A. Slotted Schedule

TSCH uses a *schedule*, which repeats over time, to synchronize its communication in time and frequency. The schedule defines a *slotframe* structure consisting of a group of *timeslots*. Nodes can use different *channel offsets* during a timeslot so a timeslot is subdivided into individual slots (see Fig. 1). The schedule specifies the channel offset and the timeslots during which a node should wake up to communicate with its neighbors [10]. The node can go to sleep to save energy in other slots. Each timeslot has an unique Absolute Slot Number (ASN): the total number of timeslots elapsed since the start of the network or an arbitrary start time determined by the PAN coordinator. The ASN counter is incremented for every timeslot and its value is tracked by all the devices in the network. The ASN and the channel offset uniquely identify a slot.

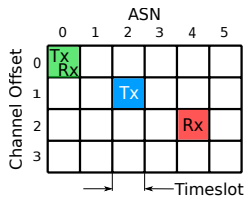



Fig. 1. Example TSCH schedule and slotframe with 6 timeslots, 4 channel offsets, and slots dedicated to Tx or Rx operations.

A single slot is long enough to send a maximum length frame and receive an acknowledgement frame. The duration of a slot is implementation specific with 10 ms or 15 ms commonly used values. When a node wakes up for a transmission slot (Tx) in its schedule, it checks the transmission buffer for a frame to send. If the buffer is empty, the node goes back to sleep. If there is a frame in the buffer, the node turns the radio on, sends the frame, and waits for the acknowledgement if expected. For a reception slot (Rx), the node turns on its radio to receive a frame, sends back an acknowledgement if required, and goes back to sleep. When it does not receive anything within a specified time interval, it goes back to sleep. During the TxRx slot, the node first checks the buffer for a frame to transmit. If there is one, it proceeds as in the Tx slot, otherwise it acts as an Rx slot. No frame reception in Rx mode means that either the sender had nothing to send or the frame sent by a neighboring node was lost.

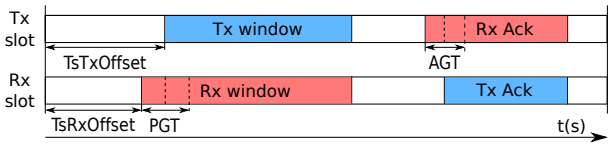



Fig. 2. Timeslot template indicates the time offsets within the timeslot. PGT and AGT stand for Packet and Acknowledgment Guard Time, respectively.

One of the main advantages of TSCH is frequency channel hopping that results in frequency diversity to mitigate the effects of interference and multipath fading [11]. Moreover, the use of several frequencies increases network capacity because more nodes can transmit at the same time. Nodes periodically switch the frequency channel using a *hopping sequence*, an ordered list of frequency channels (by default, it is a pseudo-randomly sorted list of all available channels) known to both sending and receiving nodes. Two hopping nodes sharing a hopping sequence compute the frequency channel for a given slot with [1]:

$$CH = HS[counter \mod sequence\_length], \qquad (1)$$

where $HS$ is a hopping sequence, $counter$ is a shared counter for a pair of communicating nodes, and $sequence\_length$ is the length of the channel hopping sequence. In most TSCH implementations, the counter is constructed as the sum of the ASN and a channel offset for a given slot [11], [12]. Nodes can use 16 available frequency channels. If the hopping sequence length and slotframe size are relatively prime, nodes will successively hop through all available channels [11].

### B. TSCH PAN Formation and Maintenance

*1) Building a TSCH Network:* Nodes in the network advertise the network presence by periodically sending out Enhanced Beacons (EB) containing a payload with information elements (IE). IEs contain all the information needed by a new or desynchronized node to join the network. The following payloads are mandatory to ensure the successful synchronization of a new node:

- *Synchronization IE:* contains the current ASN.
- *Timeslot IE:* contains the timeslot template that specifies when the radio of the receiver and sender should be

[1]The OpenWSN project provides open-source implementations of IoT protocol stacks (https://github.com/openwsn-berkeley).

turned on, when they should expect an incoming data or acknowledgement frame. Fig. 2 presents a template.
- *Channel Hopping IE:* contains the hopping sequence used by the network.
- *TSCH Slotframe and Link IE:* contains one or more slotframes and their active slots. The joining nodes can use the slots to communicate with the advertising device.

A node willing to join the network turns its radio on and listens to EBs on a given channel [13]. The information delivered by an EB allows the joining node to construct an initial local schedule and to negotiate with the advertising node some dedicated slots in which only one pair of nodes can communicate. After building the initial schedule, the node has to select a time source neighbor for clock calibration and eventually construct its own EB for network advertisement.

*2) Synchronization:* As described above, a node initially synchronizes with the network by means of EBs. This way of operation is called *advertisement based synchronization* [14]. Every network has an advertisement policy that defines when nodes transmit EBs. After the bootstrap phase synchronization, nodes maintain time synchronization through regular communication with their time source neighbor. The need for synchronization comes from the fact that nodes have independent clocks, the frequency of which may diverge from the nominal frequency because of the production spread, small imperfections in the crystal and the surrounding electrical components. External factors can also influence the actual frequency of clocks. All these phenomena cause a clock drift: we assume that nodes have imperfect clocks $C(t)$ with a given *drift* $D$ such that $\left|\frac{dC(t)}{dt} - 1\right| \leq D$ [15]. A typical value of $D$ is $40\,ppm$ for crystal clocks. The clock *skew* is the difference between a given clock with respect to the perfect clock after a certain time: $\delta = [C(t) - t] \cdot \Delta t$.

TSCH defines an additional synchronization method based on exchanging *keep-alive frames*. A higher layer such as 6top can create a keep-alive (KA) frame when there has been no clock calibration for a long time. When a node does not communicate with its time source neighbor for a time longer than the *KA interval*, it is considered desynchronized. KA frames contain no data, they are sent just to calibrate the clock.

When a node pair exchanges any frame with its time source neighbor, it uses either *frame-based* or *acknowledgement-based synchronization*. Both methods require the receiver to measure the clock skew ($\delta$) between the expected and actual time of the frame arrival. Because of the clock skew, nodes have to use a *guard time* when turning on the radio. TSCH defines the *Packet Guard Time* (PGT) with typical value of $2600\,\mu s$ [12] [14]. A receiver will turn its radio on PGT/2 earlier than it expects the arrival of a frame (see Fig. 3). If at least one frame is sent every KA interval, then $|\delta| < \frac{PGT}{2}$ and the reception of the frame is guaranteed. When the node does not receive a frame preamble after a full PGT, it will turn its radio off and go back to sleep. The same procedure is used at the transmitter side when it expects an acknowledgement.

The frame-based and acknowledgement-based synchronization schemes are defined as follows:

1) The receiver records timestamp $t_s$ of the frame start symbol.
2) It measures the following clock skew:
$$\delta = Radio\_offset + \frac{PGT}{2} - t_s, \tag{2}$$
where $Radio\_offset$ represents the time between the beginning of the timeslot and the activation of the radio. For Tx and Rx slots these offsets are called TsTxOffset and TsRxOffset, respectively, see Fig. 3.
3) In case of frame-based synchronization, the receiver will use the measured value of $\delta$ to calibrate its clock. When using acknowledgement-based synchronization, the receiver will add $\delta$ to the acknowledgement frame, which is sent back to the transmitter. In this scenario, the transmitter calibrates its clock.

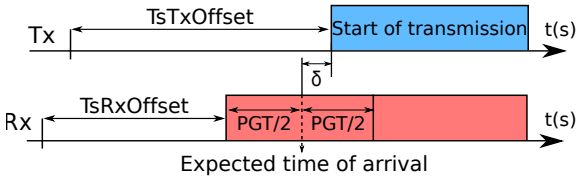



Fig. 3. Node pair timeline: the receiver activates radio $\frac{PGT}{2}$ earlier than the instant at which it expects a frame, $|\delta|$ must be smaller than $\frac{PGT}{2}$.

## III. Related Work

In this section, we give a brief overview of the main proposals related to our work and with which we compare in the evaluation. The vast amount of related research has mainly focused on optimizing the bootstrap synchronization phase for single channel protocols. Nodes have no prior knowledge of the network but use only one radio frequency for discovery [16] [17] [18]. We do not assess these proposals as our scheme focuses on a multichannel rejoining scenario, where we can exploit previously gained network information. In this paper, we compare our scheme to the proposals of Vogli et al. [5] and Duy et al. [6] as their join schemes also specifically focus on the TSCH protocol. Both papers impose network advertisement policies, while our scheme does not require any specific scheduling of EBs nor data frames.

Vogli et al. [5] proposed two advertisement policies: Random Vertical filling (RV) and Random Horizontal filling (RH). The RV policy fixes a timeslot. Each new node that joins the network sends one EB per slotframe. The location of the EB in the schedule is defined by the fixed timeslot and a randomly picked channel offset. In contrast, the RH policy fixes the channel offset and randomly chooses a timeslot. As timeslots or channel offsets are randomly picked, there is no guarantee that a slot will not be taken twice by neighboring nodes. If this happens, EBs are lost due to collisions. The chance that a node captures an EB will also reduce when the slotframe size and the amount of used channel offsets increases [13].

Duy et al. [6] took a similar approach, but they let every node send multiple EBs per slotframe. Their scheme divides a slotframe into an advertisement plane and a data plane. Each

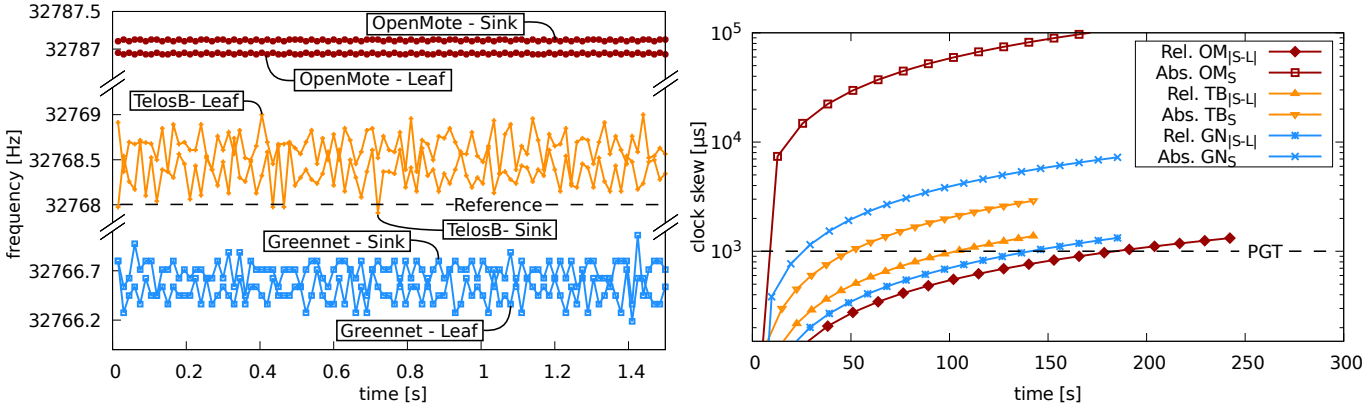



Fig. 4. Clock measurements on TelosB Crossbow, OpenMote-CC2538, and GREENNET platforms (at 25 °C): a) frequency variations in time, b) clock skew.

synchronizer, a node that advertises the network with EBs, uses the advertisement plane to send a EBs on random consecutive channels. The number of EBs sent by each synchronizer depends on the output of a fuzzy system that takes into account the number of neighbors in the network and optimizes the number of EBs sent by each node.

## IV. IDENTIFYING ROOT CAUSES OF THE CLOCK DRIFT, CLOCK SKEW AND ITS MODELING

In subsection IV-A and IV-B, we respectively investigate the internal and external sources, i.e., voltage and temperature, of the clock drift and the clock skew on different hardware platforms running TSCH. The methodology of the study is to have a one-hop network with a sink and a leaf. We deactivate all drift mitigation techniques of TSCH in OpenWSN and measure the clock frequencies on the nodes with a logic analyzer [19] that samples at 50 MHz.

OpenWSN expects each node to have a clock frequency of exactly 32 768 Hz. Therefore, a perfect clock with a frequency of 32 768 Hz is our reference clock to measure the drift and the skew of a given clock. Fig. 4a shows the frequencies of the crystal oscillator for both the sink and the leaf for each tested platform, the reference clock is shown by the dashed line. We call the difference between the real frequency and the reference the *absolute drift* ($D_{abs}$). The difference between the absolute drift of the sink and the leaf is the *relative drift* ($D_{rel}$). Fig. 4b shows how fast the clocks diverge with respect to a perfect clock by showing the *absolute clock skew*, and with respect to each other, by presenting the *relative clock skew*. Every time the node uses Eq. (2), it actually measures the relative clock skew with respect to its time source neighbor. We only show the absolute clock skew for the sink and the relative clock skew to keep the figure readable. The leaf absolute skew is not shown in Fig. 4b.

### A. Internal Drift Causes

The TelosB platform has a 32 768 Hz crystal oscillator [20] and we observe that it has the smallest absolute drift (see Table I), but also important fluctuations, caused by the production spread. The GREENNET motes use a 64 kHz clock [9] with a scaling factor to approximate a 32 768 Hz clock. As the oscillator frequency is not a multiple of 32 768 Hz, the resulting frequency deviates with an extra 2 Hz from the reference frequency, resulting in a greater absolute drift. The OpenMote nodes have a 32 MHz internal oscillator [8], so its output is divided by an integer to obtain the frequency of 32 768 Hz. Similarly to the GREENNET platform, the 32 MHz frequency of OpenMotes is not a multiple of 32 768 Hz and the obtained frequency is shifted with roughly 18 Hz on top of the production spread, giving them very high absolute drift.

TABLE I
MEASURED FREQUENCY (HZ) AND DRIFT (PPM) ON CHOSEN PLATFORMS.

| | **Avg. Frequency** | | $D_{abs}$ | | $D_{rel}$ |
|---|---|---|---|---|---|
| | Sink | Leaf | Sink | Leaf | $\lvert S - L \rvert$ |
| TelosB | 32768.65 | 32768.34 | 20.1 | 10.6 | 9.5 |
| GREENNET | 32766.47 | 32766.47 | 46.3 | 39.5 | 6.8 |
| OpenMote | 32786.93 | 32786.93 | 578.5 | 584.0 | 5.5 |

The absolute drift shown in Table I generates an increasing absolute skew, $\delta_{abs}$, with respect to a perfect clock of 32 768 Hz (see Fig. 4b) represented after time $\Delta t$ by the following relation:

$$\delta_{abs} = \Delta t \left( \frac{1}{1 \pm e_f} + \epsilon \right), \tag{3}$$

where $\pm e_f$ [ppm] is the production spread and $\epsilon$ represents the frequency shift due to integer divisions.

The absolute skew for the OpenMote nodes is the largest due to the large shift of 18 Hz. At the same time, the OpenMote nodes present small frequency fluctuations while they are fairly large on TelosB and GREENNET, which explains the smaller relative drift ($D_{rel}$) for OpenMotes. The clock of OpenMotes runs at 32 MHz and dividing its frequency by almost 1000 renders the clock fluctuations negligible. For the TelosB and GREENNET nodes, the frequency fluctuations result in a greater relative drift. When the resulting relative skew exceeds $\frac{\text{PGT}}{2}$, i.e. 1300 µs, indicated by the dashed line in Fig. 4b, the leaf desynchronizes.

We can observe that combining nodes of different hardware platforms leads to an increased relative skew due to a large $\Delta\epsilon$ between platforms. Synchronization in heterogeneous hardware networks falls out of the scope of this paper, but the authors are actively working on this issue.

### B. External Drift Sources

*1) Temperature:* The main external factor that influences the clock drift is temperature. Other causes, i.e. supply voltage and crystal aging, change more slowly [2]. Datasheets of clock crystals often describe the relationship between the temperature and the crystal frequency. We can use this information to compensate the varying drift when a node can measure the temperature changes, see subsection V-B.

*2) Voltage:* We have tested the impact of the battery voltage on the TelosB crystal oscillator. We only tested the TelosB platform as it is the only platform at our disposal that does not have any voltage regulator. The input voltage is directly applied to the clock crystal. In an experiment, we have successively used the voltage of $1.8\,\mathrm{V}$, $3\,\mathrm{V}$, and $4\,\mathrm{V}$ for periods of ten minutes. We have measured the following corresponding frequencies: $32\,768.42\,\mathrm{Hz}$, $32\,768.45\,\mathrm{Hz}$, and $32\,768.49\,\mathrm{Hz}$. Although we can notice a slight increase in frequency with the growing input voltage, the values are too small to conclude that the changing voltage would noticeably influence the clock drift for this platform. The TelosB nodes include a battery voltage monitor that can track the voltage levels of the battery. Platforms whose clock drift depends on a changing input voltage can use such a monitor to compensate for this external form of the varying drift.

### C. Clock Drift and Clock Skew Model

By accounting for all the previous causes of the clock drift, we can use the following relation to estimate the relative skew between a node and its time source neighbor after time $\Delta t$:

$$\delta_{rel} = \Delta t\left(\frac{1}{1-e_f} - \frac{1}{1+e_f} + E(t) + \Delta\epsilon\right), \qquad (4)$$

where $E(t)$, $\Delta\epsilon$ and $\pm e_f$ represent the varying drift from external causes, the difference in the frequency shift between the nodes and the production spreads of the nodes, respectively.

## V. Predictive Rejoining Scheme for TSCH

Building on the model presented in the last section, we propose a fast rejoining scheme based on the prediction of the clock drift and the resulting skew. The scheme minimizes the join latency and energy consumption. It is backward compatible with the TSCH protocol. Nodes implementing the scheme do not rely on any actions taken by other nodes and can thus coexist with "standard" nodes. As we assume that the radio is the main energy consumer, the scheme needs to minimize the time during which a node actively scans for EB frames and tries to rejoin the network. To decide when the node needs to start receiving, the scheme computes the relative clock drift to predict the instant of the next frame, sent by a time source neighbor. The duration of the *Rx window* depends on the predicted skew, the minimal duration being default PGT ($2600\,\mu\mathrm{s}$).

### A. Principles of the Rejoining Scheme

The scheme consists of three phases:

**Phase 1.** The first stage occurs when nodes are still synchronized. Every node periodically measures its clock skew with respect to its time source neighbor, $\delta_{rel}$, by using the frame-based synchronization method and calibrates its clock accordingly, see Fig. 3. The rejoining scheme sums up the measured skews over a time interval. The relative drift can be calculated by dividing the skew sum by the time interval. This estimation is constantly updated while the node stays synchronized. A node also stores the ASN value each time it measures $\delta_{rel}$. After each clock calibration, the node restarts a low power timer that counts the elapsed slots until a new clock calibration occurs. The timer keeps running even when the node is in the low power sleeping mode to save energy.

A node becomes desynchronized when the time elapsed since the last clock calibration exceeds the KA interval. For example solar-energy powered nodes desynchronize during the night. At this instant, we cannot guarantee that frames will be captured as the relative clock skew, $\delta_{rel}$, might be larger than PGT/2. When the desynchronized node does no longer exchanges frames with its time source neighbor, the clock drift causes the skew to grow further (see Fig. 5). The ASN of the desynchronized node for a given slot does not longer correspond to the ASN used by its time source neighbor and Eq. (1) will not give the right frequency channel.

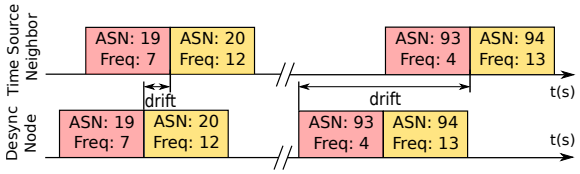



Fig. 5. Node pair timeline: the skew grows larger than the entire timeslot.

We can notice that all other information delivered in the EB frame (slotframe size, channel hopping sequence, and timeslot template) during the bootstrap phase of the network stays valid after desynchronization. We can make this assumption because changing these parameters would require a full update of the whole network.

**Phase 2.** The second phase starts when the node detects its desynchronization, which may happen either directly after the KA interval overflow or after an extended period of time when the node wakes up from low power mode. The node adds the value of the low power timer to the most recently stored ASN. The resulting value gives the node the first estimation of the ASN currently used by the network. Then, the node calculates the additional skew by multiplying the most recent relative drift estimation with the total time the node was desynchronized. If the additional relative skew since the last clock calibration is larger than an entire timeslot, the node

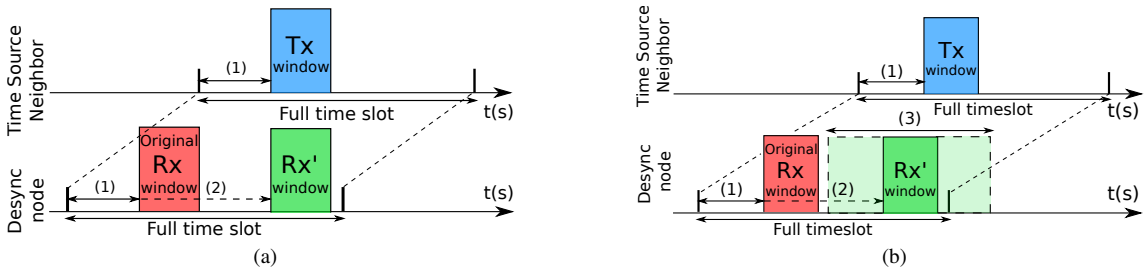



Fig. 6. Node pair timeline illustrating rejoin. (1) TsRxOffset and TsTxOffset before the respective Tx and Rx radio windows. (2) The predicted relative skew allows for rescheduling of the Rx window (shown as the RX' window). (3) Rx window expands for the unpredictable skew after desynchronization.

updates its estimated ASN value. For example, on the right side of Fig. 5, the desynchronized node estimates an ASN of 94, while due to the clock drift, the actual ASN used by the network, and the time source neighbor, is 93. By accounting for the additional skew, a node detects that the estimated ASN is off by 1 because the additional skew is larger than a timeslot.

**Phase 3.** In the final step, the desynchronized node looks at its stored schedule for the first Rx slot for communication with its time source neighbor and it goes to sleep up to the Rx slot. When the node wakes up, it reestimates its additional skew as if it had to sleep for a long time until the Rx slot. The schedule contains the channel offset of the respective Rx slot and combined with the estimated ASN, the node can tune its radio to the right frequency channel. The node uses the estimated skew to reschedule the Rx window such that it aligns with the Tx window of its time source neighbor (see Fig. 6a). The node activates its radio for the PGT interval. If the time source neighbor does not send a frame in the Rx slot, the node repeats the final step and locates the next Rx slots.

When the node successfully captures an incoming frame from its time source neighbor, it recalibrates its clock with the KA based synchronization method. The low power timer is reset and the node becomes again synchronized with the network. The captured frame does not have to be necessarily addressed to the desynchronized node, but the transmitter must be its time source neighbor. The node can use any frame coming from the time source neighbor to synchronize (so its content is not important), which means that an encrypted frame can also be used to resynchronize—only the arrival instant is needed to recalibrate the clock.

### *B. Compensation for Unpredictable Drift Causes*

Unforeseen drift fluctuations after desynchronization and finite arithmetic precision during the skew estimation may create an undetected skew between the Tx window of the time source neighbor and the Rx window of the desynchronized node, see Fig. 6b. This skew comes from the fact that the relative drift the node established during phase 1 of the rejoining scheme is only an ephemeral estimation. The relative drift can change any moment after desynchronization due to external causes, e.g. temperature fluctuations. Once the node is desynchronized, it is no longer aware that the relative drift has changed as it cannot communicate with its time source neighbor. The second cause of scheduling errors is the finite precision when the node estimates the relative skew because all platforms only use 16-bit integer arithmetic. Rounding errors become more relevant when a node is desynchronized for longer periods.

To compensate for the skew caused by unforeseen drift causes, the Rx radio window grows steadily in time (see Fig. 6b)—it depends on the total time the node was desynchronized, the arithmetic precision, and the worst case external drift change, e.g. the largest temperature change, expected in the network. The window expands in both directions as the desynchronized node does not know in which direction the clock will drift. Nodes with a temperature sensor and having sufficient energy can apply drift compensation in real-time. Its goal is to keep the actual relative drift as close as possible to the previously established relative drift estimation. If the node obtains a stable relative drift by compensating the drift changes that occur after desynchronization, our scheme gives a more precise prediction of the relative skew and the instant of the Tx window used by its time source neighbor, which results in an smaller Rx window, thus lower energy consumption.

## VI. Experimental Evaluation

To validate our prediction-based rejoining scheme, we have implemented it on the hardware platforms [7] [8] [9], using the OpenWSN networking stack. We use the TSCH-minimal schedule with 16 frequency channels and a slotframe of 101 timeslots, $15\,\mathrm{ms}$ each to obtain a $1\%$ duty cycle [21]. The EB interval is the sum of a fixed value of $30\,\mathrm{s}$ and a random value chosen in $[-3.48; +3.48]$, which lowers the probability of an EB collision when nodes use the same EB interval and the same timeslot. We compare our scheme with the proposals by Vogli et al. [5] and Duy et al. [6].

### *A. Join Latency*

In the first experiment, we have evaluated the join latency of our scheme on the three platforms. The first node is configured as the network sink, while a second node is set up as a network leaf. At the beginning of the experiment, we wait until the leaf is synchronized with the sink. Once the initial synchronization occurs, we wait an additional $5\,\mathrm{min}$ so that the leaf node learns its relative drift (see Phase 1 of Section V) with respect to the sink. Once the leaf node establishes an initial relative drift estimation, the sink node starts sending frames on an unreachable frequency channel. After a period

of the KA interval, the leaf node enters the desynchronized state and the resynchronization procedure starts. The sink node keeps transmitting its frames on the unreachable frequency channel for $5\,\mathrm{min}$. After this delay, we instruct the sink to return to the same channel hopping sequence as the leaf node. Starting from this instant, we start measuring the time for the leaf node to resynchronize with the sink. In this experiment, the rejoin procedure becomes active directly after the node desynchronizes as the node has sufficient energy.

Table II shows the average join latency for three different scenarios. We obtain the experimental results for our scheme after 35 runs on the OpenMote platform and we compute the average join latency for the other schemes based on the theoretical expressions experimentally validated by the authors in their respective papers [5], [6]. Below the results for the other proposals, we show the ratio with respect to our scheme.

TABLE II
AVERAGE REJOIN LATENCY (s)

| **Join Scheme** | **EB** (30 s) | **EB** (15 s) | **EB** (6 s) |
|---|---|---|---|
| Predictive Rejoin | 7.51 | 5.9 | 5.25 |
| Fast\|Rapid Join [5][6] | 255<br>× 34 | 127.5<br>× 22 | 51<br>× 10 |
| Rapid Join [6] with 4 EBs | 63.75<br>× 8 | 31.88<br>× 5 | 12.75<br>× 2 |

We can see that our scheme significantly outperforms the other proposals in each scenario. The upper bound join latency for our scheme corresponds to the interval between EBs as we will always capture every frame sent by the time source neighbor for a Packet Delivery Ratio (PDR) of 100% and the correct drift estimation. However, other frames such as Routing Protocol for Low-Power and Lossy Networks (RPL), KA, or data frames will very often speed up the resynchronization process, because they are also sent by the network and they do not cost any additional energy. Adding extra EB frames will therefore only slightly improve the speed of our scheme. The other proposals only use EBs for resynchronization and do not predict the instant of the transmitted EBs. Fast Join by Vogli [5] sends one EB per node. Rapid Join Scheme by Duy [6] has two modes: the first mode is the same as Fast Join, and the second mode lets every node send multiple EBs (it is the scheme in the last row of Table II). The latter mode obtains faster rejoin, but consumes more energy from the network perspective as every node sends extra EBs. A typical energy cost for an EB, depending on its size and hardware platform, is $72.4\,\mu\mathrm{J}$ with a battery at $2.2\,\mathrm{V}$ [6].

We have further tested the influence of the application traffic on the rejoin scheme performance on the other hardware platforms. Fig. 7 shows the synchronization speed for three different traffic loads. In the first setup, there is no application compiled on top of the stack and nodes only rely on EBs, RPL, and KA frames to resynchronize. In the second setup, the sink node injects one data frame in the network every EB period and in the last scenario, the sink sends four extra data packets per EB period. We notice that the resynchronization speed for our scheme is independent from the platform and shows the same behavior as in Table II. We can also see that there is one outlier for the TelosB experiment with no application data. We cannot be sure what caused the desynchronized node to miss earlier frames, but we suspect interference issues. Our scheme reschedules its Rx window and resynchronizes to the next frame sent by its time source neighbor.

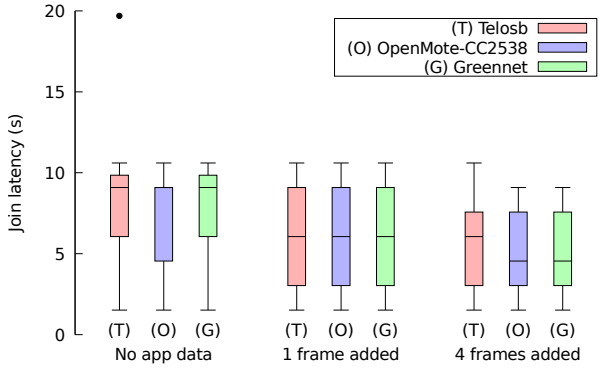



Fig. 7. Join latency after 35 experimental runs for three application traffic loads on three different platforms.

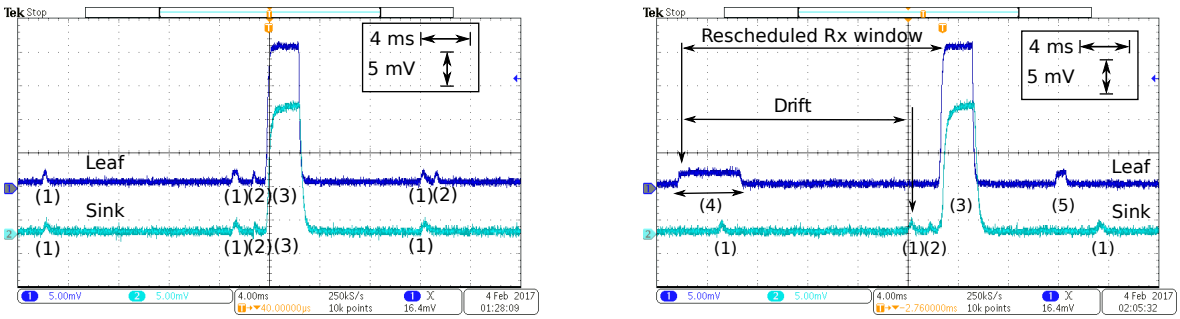
### B. Expanded Radio Reception Windows

*1) Impact of the Arithmetic Precision:* The OpenWSN code is written in a portable way so it can be run on a wide range of different platforms. We have used integer arithmetic for computing the relative skew because either the usage of floating point arithmetic is not supported or the memory and speed costs are too high. We have optimized our code to obtain a relative drift precision of $0.5\,ppm$. To get the experimental results in Table III, we have used the same setup as in the first experiment, but we have desynchronized the leaf for $1\,\mathrm{h}$. At the end, we have measured with the logic analyzer [19] at $25\,\mathrm{MHz}$ the difference between the skew prediction and the real skew. The desynchronized node uses a standard Rx window size of PGT. If the unpredicted skew is smaller than the half PGT ($1300\,\mu\mathrm{s}$), a successful resynchronization is possible without an expanded Rx window (see Fig. 6).

TABLE III
ARITHMETIC PRECISION IMPACT ON DRIFT PREDICTION (ms)

| **Platform** | **Predicted** | **Real** | **Abs. Err.** | **Rel. Err.** |
|---|---|---|---|---|
| TelosB | 39.26 | 39.94 | $0.68 \pm 0.08\mathrm{e}{-3}$ | 1.2% |
| OpenMote | 2.13 | 2.39 | $0.26 \pm 0.08\mathrm{e}{-3}$ | 3.1% |
| GREENNET | 21.67 | 23.65 | $1.98 \pm 0.08\mathrm{e}{-3}$ | 0.4% |

We notice that after $1\,\mathrm{h}$ both the OpenMote and TelosB platforms are still able to resynchronize without an expanded Rx window. The bad results on the GREENNET platform can be explained by looking at the relative drift prediction and the actual drift between a GREENNET pair of nodes: the prediction error on the GREENNET platform is significantly larger because the relative drift for the GREENNET nodes amounts to




(a) (b)

Fig. 8. Current measurements with a Tektronix MD03012 oscilloscope for energy consumption evaluation on the TelosB platform (battery at 2.2 V).

$6.8\,ppm$ (see Table I), which is not a multiple of $0.5\,ppm$, the smallest step we can apply in the Rx rescheduling process with our scheme. Therefore, the drift estimation introduces an extra error of $0.2\,ppm$. The relative drifts for the other platforms are multiples of $0.5\,ppm$, which results in a smaller error.

*2) Temperature Impact:* In this experiment, we have evaluated the growth of the Rx windows when after desynchronization, a node is exposed to a fluctuating relative drift caused by temperature changes. We have considered two cases: i) a node without a temperature sensor and therefore it could not compensate the additional drift, ii) a node with a temperature sensor, compensating the measured drift in real-time. In both experiments, the TelosB sink was in a room with a temperature of approximately $25\,°\mathrm{C}$. The leaf was put in an environment where the temperature ranged between $10\,°\mathrm{C}$ and $35\,°\mathrm{C}$. The leaf measured its temperature and relative clock drift every minute, and sent the results to the sink as plotted in Fig. 9. The drift compensation in the second scenario was computed using the relation $-0.035\ \mathrm{ppm}/°\mathrm{C}^2 \times (T - T_0)^2$ [20]. When the nodes start communicating, the temperature is $25\,°\mathrm{C}$ everywhere and the clocks operate at their nominal temperature. The relative drift measured during the first $10\,\mathrm{min}$ of the experiment, at a constant temperature, is around $9.5\,ppm$ for node pair (1) and (2). After $10\,\mathrm{min}$, we have gradually changed the temperature in the environment of the leaf node. The node, without temperature sensor, is not aware of the temperature fluctuation. It cannot compensate and the relative drift for node pair (1) changes drastically. Notice that Phase 1 of our scheme accounts for the observed $9.5\,ppm$ and the expanded Rx window must compensate for the additional drift.

Node pair (1) has an additional $5\,ppm$ drift caused by the temperature changes (see Fig. 9), which amounts to $18\,\mathrm{ms}$ after $1\,\mathrm{h}$. Because we cannot predict how the drift changes, we would have to expand the Rx window at both sides to: $(2 \cdot 18\,\mathrm{ms}) + \mathrm{PGT} = 38.6\,\mathrm{ms}$. Node pair (2) only needs to compensate an additional $1.5\,ppm$, which amounts to a smaller Rx window of $13.4\,\mathrm{ms}$.

### C. Predictive Rejoin Scheme Footprint

Fig. 8 presents the measured current consumption: a) by a synchronized pair of TelosB nodes and b) when the predictive rejoin scheme reschedules the Rx window.

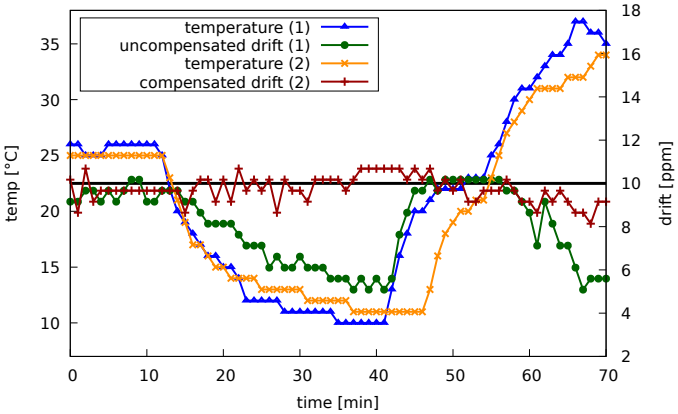



Fig. 9. Temperature and drift in time for compensated and uncompensated TelosB pairs shows that temperature changes cause an additional drift.

TABLE IV
MEASURED ENERGY CONSUMPTION OF TELOSB (mJ)

| TelosB | CPU Idle | CPU Active | Radio Tx/Rx |
|---|---|---|---|
| Energy | 0.66 | 2.86 | 39.6 |

We use a simplified energy model by Vilajosana et al. [14] to obtain energy consumption from the current measurements shown in Fig. 8. Table IV presents the measured energy (mJ) on a TelosB node for the different states of a timeslot. Fig. 8 shows (1) the CPU waking up to check if a frame needs to be sent at this timeslot, (2) the preparation of the data that needs to be sent, (3) activation of the radio in Rx or Tx mode, (4) the CPU waking up, noticing there is an incoming frame, calculating the skew and rescheduling the Rx window, and (5) the CPU verifying there is no incoming frame from the time source neighbor. In Fig. 8, (4) shows the computational overhead of the scheme in phase 2 and 3. The code is not optimized but only uses basic arithmetic operations. The non-optimized code size is $2.5\,\mathrm{kB}$. The OpenWSN stack code size is currently more than $50\,\mathrm{kB}$. The overhead of phase 1 is negligible as it only increments a counter, uses additions and a simple division to estimate the clock drift. We can compute the total used energy in a timeslot as the sum of the energy spent in each state (Idle, Active, Radio), where $T$, $I$ and $V_b$

are the time spent, the current drawn in a state, and the battery voltage, respectively:

$$E_{tot} = (T_{Id} \cdot I_{Id} + T_{Act} \cdot I_{Act} + T_{Tx|Rx} \cdot I_{Tx|Rx}) \cdot V_b. \quad (5)$$

An "idle slot" is a slot during which only the CPU is responsible for energy consumption. Slots during which the radio is active are "radio slots". Fast and Rapid Rejoin Schemes [5], [6] cannot use prediction to localize the incoming EB. Therefore, they need to keep their radio active all the time. The multi-beacon option of Rapid Join also requires additional network energy because it sends multiple EBs in the same EB period. Because the sink uses a duty cycle of $1\%$, our scheme also approaches this duty cycle. It uses one double sized active slot to reschedule the Rx window and 99 idle slots per slotframe (see Fig. 8b). This approach facilitates rescheduling of the Rx window when it is located in between two timeslots. The energy drawn for 99 idle slots and 1 double sized active slot is $1.29\,\mathrm{mJ}$. While the energy for 101 active slots is $59.99\,\mathrm{mJ}$. Table V shows the computed energy for the measured join latency presented in Table II. Below the used energy for the other proposals, we show the ratio with respect to our scheme.

TABLE V
ENERGY USED FOR RESYNCHRONIZATION (mJ)

| **Join scheme** | **EB** (30 s) | **EB** (15 s) | **EB** (6 s) |
|---|---|---|---|
| Predictive Rejoin | 6.45 | 5.15 | 4.51 |
| Fast\|Rapid Join [5][6] | 15.3e3<br>× 2372 | 7.65e3<br>× 1485 | 3.06e3<br>× 678 |
| Rapid Join [6] (4 EBs) | 3.82e3<br>× 592 | 1.91e3<br>× 370 | 0.76e3<br>× 168 |

## VII. CONCLUSION

This paper proposes a novel predictive resynchronization scheme that allows nodes to rejoin quickly the TSCH network with which they were previously synchronized. The idea of the scheme comes from the modeling of the clock drift root causes. Based on the prediction of the relative drift, we are able to determine the instant at which a node needs to wake up after desynchronization thus reducing its energy consumption.

The predictive rejoining scheme is composed of three phases. In phase 1, while nodes are still synchronized, they learn their relative drift with respect to their time source neighbors. Phase 2 happens when the node desynchronizes from the network. It estimates the correct ASN based on a low power timer and skew prediction. Phase 3 applies a rescheduled Rx window that has an adapted length to compensate unforeseen drift sources.

We have compared the proposed scheme with other state-of-the-art proposals. The experimental results show that our scheme significantly outperforms other proposals as we can exploit the previously obtained network information. The scheme results in a join latency that is up to 34 times faster and the nodes consume a factor of 1000 less energy.

## VIII. ACKNOWLEDGMENTS

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program Investissement d'avenir, the FUI IoTize project funded by Région Auvergne-Rhône-Alpes, and the DataTweet project under contract ANR-13-INFR-0008-01.

## REFERENCES

[1] *IEEE 802.15.4e Low-Rate Wireless Personal Area Networks (Amendment to IEEE Std 802.15.4-2011)*. New York, NY, USA: IEEE Standards Office, 2012.
[2] D. Stanislowski, X. Vilajosana, Q. Wang, T. Watteyne, and K. S. Pister, "Adaptive Synchronization in IEEE 802.15.4e Networks," *IEEE Trans. on Industrial Informatics*, vol. 10, no. 1, pp. 795–802, 2014.
[3] A. Elsts, S. Duquennoy, X. Fafoutis, G. Oikonomou, R. Piechocki, and I. Craddock, "Microsecond-Accuracy Time Synchronization Using the IEEE 802.15. 4 TSCH Protocol," *2016 SenseApp*, 2016.
[4] A. Tinka, T. Watteyne, and K. Pister, "A Decentralized Scheduling Algorithm for Time Synchronized Channel Hopping," in *Ad Hoc Networks*. Springer, 2010, pp. 201–216.
[5] E. Vogli, G. Ribezzo, L. A. Grieco, and G. Boggia, "Fast Join and Synchronization Schema in the IEEE 802.15.4e MAC," in *2015 IEEE WCNC Workshops*, 2015, pp. 85–90.
[6] T. P. Duy, T. Dinh, and Y. Kim, "A Rapid Joining Scheme based on Fuzzy Logic for Highly Dynamic IEEE 802.15.4e TSCH Networks," *IJDSN*, vol. 12, no. 8, 2016.
[7] Telosb. Memsic, Inc. San Jose, California. [Online]. Available: https://openwsn.atlassian.net/wiki/display/OW/TelosB
[8] *A Powerful System-On-Chip for 2.4-GHz IEEE 802.15.4-2006 and ZigBee Applications*, Texas Instruments, 12 2012.
[9] L.-O. Varga, G. Romaniello, M. Vučinić, M. Favre, A. Banciu, R. Guizzetti, C. Planat, P. Urard, M. Heusse, F. Rousseau, O. Alphand, E. Dublé, and A. Duda, "GreenNet: an Energy Harvesting IP-enabled Wireless Sensor Network," *IEEE Internet of Things Journal*, vol. 2, 2015.
[10] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized Protocol Stack for the Internet of (Important) Things," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.
[11] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic Aware Scheduling Algorithm for Reliable Low-Power Multi-Hop IEEE 802.15. 4e Networks," in *2012 IEEE PIMRC*. IEEE, 2012, pp. 327–332.
[12] OpenWSN: Open-Source Implementations of Protocol Stacks Based on IoT Standards. [Online]. Available: http://www.openwsn.org
[13] D. De Guglielmo, A. Seghetti, G. Anastasi, and M. Conti, "A Performance Analysis of the Network Formation Process in IEEE 802.15. 4e TSCH Wireless Sensor/Actuator Networks," in *2014 IEEE ISCC*. IEEE, 2014, pp. 1–6.
[14] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. Pister, "A Realistic Energy Consumption Model for TSCH Networks," *IEEE Sensors Journal*, vol. 14, no. 2, pp. 482–489, 2014.
[15] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu, "Tiny-Sync: Tight Time Synchronization for Wireless Sensor Networks," *ACM TOSN*, vol. 3, no. 2, p. 8, 2007.
[16] P. Dutta and D. Culler, "Practical Asynchronous Neighbor Discovery and Rendezvous for Mobile Sensing Applications," in *Proc. 6th ACM SenSys*. ACM, 2008, pp. 71–84.
[17] M. Bakht, M. Trower, and R. H. Kravets, "Searchlight: Won't You Be My Neighbor?" in *Proc. 18th MOBICOM*. ACM, 2012, pp. 185–196.
[18] Y. Qiu, S. Li, X. Xu, and Z. Li, "Talk More Listen Less: Energy-Efficient Neighbor Discovery in Wireless Sensor Networks," in *IEEE INFOCOM*. IEEE, 2016, pp. 1–9.
[19] (2016) Saleae. Saleae, Inc. 408 N Canal St, Unit A, South San Francisco, CA 94080. [Online]. Available: https://www.saleae.com/
[20] *MSP430 32-kHz Crystal Oscillators*, Texas Instruments, 8 2006, rev. 2.
[21] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal 6TiSCH Configuration," IETF, Internet-Draft draft-ietf-6tisch-minimal-21, Feb. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-minimal-21.