On the Synchronization of Computational RFIDs

Kasım Sinan Yıldırım, Henko Aantjes, Przemysław Pawełczak *Member, IEEE,* and Amjad Yousef Majid

Abstract—Battery-free computational RFID platforms, such as WISP (Wireless Identification and Sensing Platform), are intermittently-powered devices designed for replacing existing sensor networks. Accordingly, synchronization appears as one of the crucial building blocks for collaborative and coordinated actions in these platforms. However, intermittent power leads to frequent loss of computational state and short-term clock frequency instability that makes synchronization challenging. In this article, we introduce WISP-Sync protocol that provides synchronization among WISP tags in the communication range of an RFID reader. WISP-Sync overcomes the aforementioned challenges by employing a Proportional-Integral (PI) controller-inspired algorithm which (i) is adaptive—reactive to short-term clock instabilities; (ii) requires only a few computation steps—suitable for limited harvested energy; and (iii) keeps a few variables to hold the synchronization state—minimum overhead to recover from power interrupts. Evaluations in our testbed showed that WISP-Sync ensured an average synchronization error of approximately 1 ms among the tags with an average energy overhead of 1.85 μJ per synchronization round.

Index Terms—Time Synchronization, Computational RFIDs, Wireless Identification and Sensing Platform (WISP).

1 Introduction

Low-power wireless embedded systems consist of tiny, low-cost and spatially separated computers that communicate with each other by exchanging radio packets. Powering these small-scale embedded systems, e.g. wireless sensor networks (WSNs), is still a crucial problem [2]. Replacing or recharging their batteries is impractical and inhibits long-term operation. Moreover, batteries increase the size and cost of the hardware. Fortunately, the energy efficiency of these systems has improved considerably such that their power requirements are in the order of a few μW [3]. Furthermore, recent advancements in microelectronics technology enabled harvesting power from radio frequency (RF) sources that is sufficient to operate low-power embedded systems in practice [2], [4], [5], [6], [7]. Nowadays, the growth of the RF-powered computing paradigm is bringing new research opportunities and challenges [2], leading to a promising class of low-power embedded systems, the so-called Intermittently Powered Devices (IPDs).

By taking existing RFID (Radio Frequency IDentification) technology as a foundation, computational RFIDs (CRFIDs) are emerging IPDs that allow sensing, computation and communication without batteries—replacing existing battery-powered sensor networks [3]. CRFIDs are equipped with a backscatter radio [8, Chapter 4] composed of a simple circuitry that modulates the carrier wave generated by a reader to transmit information. This allows communication to come almost for free. This is a fundamental difference from sensor networks where the transceiver circuit is the most energy-hungry component. In the CRFID domain, the bottleneck in terms of power consumption has shifted from communication to computation and sensing [9]. A typical example of CRFIDs is the WISP (Wireless Identification and Sensing Platform) [10]. Commercial RFID readers implementing the EPC Gen2 standard [11] are used to power WISP tags. Apart from low data rate sensing [12], these maintenance-free devices are evolving to support high data rate and more complex sensing applications such as continuous sensor-data streaming, e.g. capture and transfer of images via battery-free cameras, i.e. WISPCam [13], [14].

1.1 The Need for CRFID Network Synchronization

As of now, battery-less networks are in the form of one-hop architecture in which the CRFID tags communicate only with the RFID reader [3], [10], [15]. Therefore, CRFID tags cannot operate in a multi-hop manner. However, recent advancements in the wireless communication enabled tag-to-tag communication, e.g. [16]. We anticipate that this communication capability will lead up multi-hop CRFID networks that will demand their own implementation of basic sensor network building blocks [17] with the aim of replacing the existing battery-powered sensing and data collection applications with their battery-less counterparts.

Since we are not aware of any network infrastructure demonstrating neither multi-hop communication nor multi-hop synchronization, our focus in this article are the CRFID networks where several readers cover a tag population simultaneously in a single hop. A representative application is the greenhouse monitoring where several tags are powered by multiple readers, see Figure 1. In this application, the tags sense the environment and timestamp their temperature readings. When the readers request their temperature readings, the tags transmit their timestamps together with the collected temperature data. In order to make inferences on the collected temperature readings, timestamps should be consistent so that the readers can provide a temporal ordering of the readings. Therefore, tags require a common time notion to timestamp their readers—creating a demand for

- *A preliminary version of this article was presented at the HLPC 2016 Workshop, Atlanta, GA, USA, April 2, 2016 [1].*
- *Kasım Sinan Yıldırım, Przemysław Pawełczak and Amjad Yousef Majid are with the Embedded Software Group, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands (e-mail: {k.s.yildirim, p.pawelczak, a.y.majid}@tudelft.nl).*
- *Henko Aantjes is with the Brighsight BV, Brassersplein 2, 2612 CT, Delft, The Netherlands (e-mail: aantjes@brightsight.com)*
- *Kasım Sinan Yıldırım is also affilated with the Computer Engineering Department of Ege University, Izmir, Turkey.*

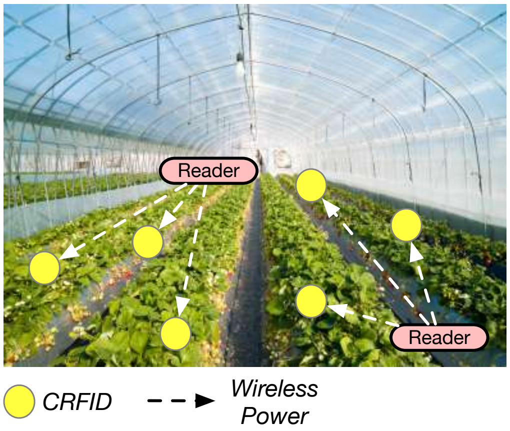

Fig. 1. A representative greenhouse monitoring application using CRFIDs. Two readers are deployed to provide power to the individual tags and to collect their periodic sensor readings.

explicit time synchronization. In order to achieve synchronization, readers should first acquire the global time, e.g. by accessing time servers using the Internet; then individual tags should be synchronized to the individual readers.

1.2 Differences From Battery-Powered WSNs

The EPC Gen2 protocol uses slotted Aloha that performs a form of synchronization between the reader and the tags: by adjusting clock offset differences and introducing guard times, tags are ensured to communicate at the allocated slot boundaries. However, this method does not compensate for clock drifts and output a synchronized clock value that can be accessed by the tags to timestamp (and make inference about) the collected information, e.g. from the sensors attached to tags. In consequence, applications requiring time notion cannot be enabled, since a clock should be maintained at each tag—making synchronization service mandatory. However, the characteristics of CRFID systems expose fundamentally different challenges than battery-powered sensor networks to implement the synchronization service. The reason is mainly twofold:

- **Challenge I:** CRFID systems should perform computations efficiently using *marginal* and intermittent RF power that leads to frequent loss of computational state, e.g. when the RFID reader moves away from the CRFID tag [18]. Therefore, computations pertaining to time synchronization have an extremely short time budget, e.g. should be completed with the other computation tasks within 10–100 ms in WISP [19]. On the contrary, the synchronization approaches in WSNs, e.g. the de facto protocol [20], keep communication infrequent by pushing computation instead, so that the battery power is used efficiently in order to extend the life-time of the network up to several years.
- **Challenge II:** The continuously varying voltage supply introduces severe hardware instability, e.g. varying oscillator frequencies in short-term affects the stability of the clocks and degrades the accuracy sensing. Therefore, computations pertaining to time synchronization should be reactive against very-frequent voltage effects; e.g. during super-capacitor discharge. On the contrary, in WSNs the input voltage is constant and the temperature is the main factor affecting the clock frequencies [21]. Temperature compensation techniques, e.g. [22], are not applicable to compensate for the voltage instabilities in CRFIDs due to their computational complexity.

1.3 Contributions

The focus of this article is to answer the question of *how to design a building block that synchronizes intermittently-powered CRFID systems?* To this end, we investigate the WISP platform and provide initial observations and limitations pertaining to the synchronization of these devices. In particular, the main contributions and findings of this article are:

- To the best of our knowledge, this is the first study that focuses on the synchronization of tags in CRFID systems. Even though there are studies on the synchronization of multiple RFID readers, e.g. [23], we are unaware of any study that provides synchronization of individual tags in the communication range of an RFID reader.
- By addressing the aforementioned challenges, we design and implement *the first synchronization primitive* for the tags, namely *WISP-Sync*, inspired by the Proportional-Integral (PI) controllers [24], [25]. WISP-Sync (i) requires only a few computation steps to run efficiently under limited harvested energy and keeps a few variables to hold the synchronization state to recover from power interruptions with minimum overhead—*addressing Challenge I*; (ii) is adaptive to react to clock instabilities in a fast manner—*addressing Challenge II*.
- Our time synchronization algorithms compensate for the clock drift and enable the first step to enable an *explicit clock* for CRFID applications. Moreover, our implementations are independent of how EPC Gen2 is implemented—providing a level of portability across different platforms.
- We provide theoretical analysis to prove that this primitive establishes synchronization and to reveal the factors affecting its synchronization performance.
- Evaluations in our testbed show that an average synchronization error of approximately 1 ms can be ensured among tags with an energy overhead of 1.85 μJ per synchronization round using this primitive.

We note that WISP-side implementations throughout the article are done using C and assembly language. All source codes and scripts used to generate the results in the article (including parsing, post-processing and measurement results) are available via https://goo.gl/RuWIZ2.

The remainder of this article is organized as follows. In Section 2 we present the fundamental challenges of synchronizing IPDs. Section 3 presents the hardware, in particular clock system of the WISP platform briefly, while Section 4 provides the mathematical notation to be used throughout the article. In Section 5 we propose a sender-receiver synchronization approach between an RFID reader and a single tag. Sections 6 and 7 present the WISP-Sync protocol and its evaluation in our testbed, respectively. We also present the synchronization of multiple tags in Section 8. Finally, we provide our conclusions and future research directions in Section 9.

2 Synchronizing Battery-Free CRFIDs

In this section, we present a brief state of the art on synchronization issues in conventional WSNs. Then, we emphasize the main challenges of synchronizing wireless battery-free embedded systems by focusing on a particular IPD platform: computational RFIDs.

2.1 Synchronization in Conventional WSNs

The instability of the clock hardware, delays during communication among sensor nodes, and software methods to establish synchronization are the main factors affecting the synchronization in conventional WSNs.

2.1.1 Clock Hardware

In WSNs, each sensor node is equipped with a built-in clock that is implemented as a counter register clocked by a low-cost external crystal oscillator. At each oscillator pulse, i.e. *tick* of the clock, the counter register is incremented. The duration between two consecutive ticks is the *rate* of the built-in clock. Environmental factors such as temperature, supply voltage and aging of the crystal prevent built-in clocks to generate ticks at the exact speed of real-time, leading to bounded *clock drift*. The prominent environmental factor affecting the frequency of the built-in clocks is the temperature [22], [26]. Moreover, *quantization errors* occur with low-frequency built-in clocks, which prevents precise timing measurements.

2.1.2 Synchronization Error

Sensor nodes exchange their clock information periodically to compute a *software clock* that represents the synchronized notion of time. A software clock is composed of an offset and a frequency that hold the value and speed difference between the corresponding built-in clock and the reference time, respectively. The difference between the reference time and the software clock is the *synchronization error*. In WSNs, the synchronization error is affected by several sources of errors. The *transmission delay*, defined as the time that passes between the start of the broadcast and the receipt by the receiver node, is the major error source and composed of deterministic and non-deterministic components [20]. Assigning timestamps at the MAC layer removes the deterministic delay components and improves synchronization accuracy. This obligates the use of transceivers that allow assignment of time information to a radio packet just before transmission and reception, e.g. Chipcon CC2420 [27]. Since the transceiver is the most power-hungry circuit, nodes in WSNs should reduce re-synchronization frequency to decrease communication overhead and save power.

2.1.3 State-of-the-art Synchronization Techniques

The most common synchronization mechanism is to propagate the time information of a particular *reference node* to let receiver nodes synchronize themselves to the received reference time information. Commonly, *least-squares regression* is employed by several practical synchronization protocols to adjust the offset and the frequency of the software clocks [20], [28], [29]. On the other hand *iterative* computation methods, e.g. in [24], [25], [30], [31], perform identical computation steps at each re-synchronization round. With iterative methods, sensor nodes establish synchronization after a finite number of rounds in a lightweight manner in terms of computation and memory requirements. There are also *fully distributed* approaches in which sensor nodes interact only with and synchronize to their direct neighbors in a peer-to-peer fashion without the requirement of a dedicated reference node. In these approaches, nodes employ computation methods based on *distributed consensus* [32], [33], [34]. Since the prominent factor affecting the clock frequencies in WSNs is the temperature [35], there are also temperature compensated time synchronization techniques, e.g. [22]. However, such techniques require a calibration phase in order to build a temperature–clock frequency table.

2.2 Fundamental Challenges of Synchronizing CRFIDs

We now delve into the synchronization characteristics of WISP, the de facto CRFID platform. The main aspects pertaining to synchronization in the WISP platform can be stated as follows:

2.2.1 Single-hop reader-tag architecture

WISP tags are deployed inside the communication range of an RFID reader and they can communicate only with the reader using backscatter communication. Therefore, the RFID reader itself is the natural reference device to establish synchronization among the WISPs, promoting *reader-tag synchronization*. Since WISP tags are unable to communicate with their neighboring nodes directly using backscatter communication as of now, *tag-tag synchronization* is not feasible.

2.2.2 Slot and bit-level synchronization

EPC Gen2 requires a slot-level synchronization to implement slotted Aloha protocol. This is achieved by adjusting clock offsets upon reception from the reader and introducing guard times to ensure that tags communicate at the allocated frame boundaries. It is also possible to exploit modulation properties of signals from the reader to the tags (such as bit timing in FM0 modulation) in order to get the timing measurements of the regular patterns in the signal, i.e. bit-level synchronization [36], [37], [38]. However, neither slot nor bit level synchronization compensate for clock drifts and output a synchronized clock value that can be accessed by the tags.

2.2.3 Continuously varying voltage level

In WSNs, the battery level decreases gradually that allows stable voltage levels in the short term. On the contrary,

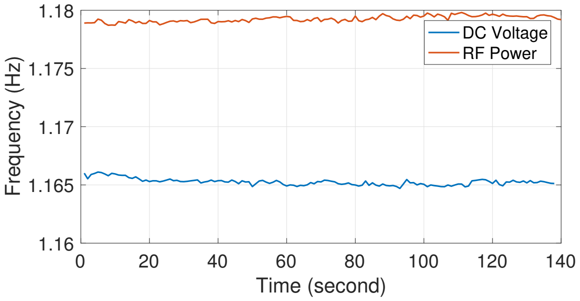

Fig. 2. In our testbed, see Section 3.2 for the details, we programmed a single WISP tag so that it toggles one of its output port whenever its 32 KHz built-in timer progresses 32000 clock ticks. In the ideal case where the crystal oscillator of the tag does not drift, the tag would toggle its output port at each 1 second, i.e. with a frequency of 1 Hz. During our experiment under a stable environmental temperature, we connected the output port of the tag to a logic analyzer and measured the frequency of the toggle event. We plotted our measurements when the tag is powered by the Flash Emulation Tool (FET) with a stable DC power and when the tag is powered by the RFID Reader at a distance of 5 cm. We measured the mean and standard deviation of the frequency in the DC case as 1.1653 and 0.00028553, and these values for the RF case were 1.1793 and 0.00026049, respectively.

on the WISP platform, the fluctuating input voltage prevents short-term stability of the clock hardware and introduces significant drift. Even though embedded platforms are equipped with voltage regulators, the output of these regulators also exhibit instability and inaccuracy with respect to the input voltage; i.e. they are not perfect and they have an input voltage transient response. Therefore, they have an output voltage variation with respect to the input voltage—still leading to clock frequency instabilities. Hence, the prominent factor affecting the frequency of the crystal oscillator is the varying voltage level, see Fig. 2.

2.2.4 Frequent power failures

Contrary to sensor nodes, WISP tags frequently "die" due to power loss and they need to save the synchronization state, e.g. the clock offset and relative frequency, into the non-volatile memory to recover when they harvested sufficient energy to start up. However, saving computational state is also an energy consuming task [39].

2.2.5 Energy and resource constraints

The classical motto of WSNs, "compute instead of communicate whenever possible" [40, p. 44], is no longer valid for the WISP platform since backscatter communication comes almost for free [9]. Due to the intermittent power, lightweight methods in terms of computation and memory are required for the synchronization. Since methods like least-squares regression are computationally heavy and require considerable amount of memory [24] they should be avoided.

2.2.6 Limitations of the EPC Gen 2 standard

The WISP firmware implements the EPC Gen 2 standard [11] which increases the compatibility with the existing RFID systems. However, the standard introduces limitations, e.g. currently it does not assign timestamps to the radio packets, which is a fundamental requirement to establish synchronization. Moreover, communication delays between the reader and tag are quite dependent on the implementation of this standard by RFID readers. Unfortunately, these issues lead to less accurate synchronization as compared to existing WSN solutions, as shown by our measurements presented in the following sections. Consequently, WSN synchronization protocols that can guarantee μs-level synchronization accuracy, e.g. [20], are hard to achieve in CRFID domain.

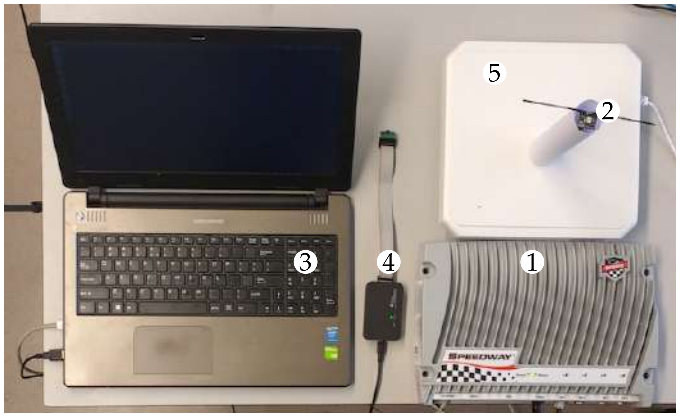

Fig. 3. A single-hop CRFID testbed: (1) an Impinj Speedway R1000 RFID reader, (2) a single WISP 5.1 tag, (3) a host computer, (4) FET programmer and (5) a Laird S9028PCR antenna. The same testbed setup has been used previously in [15] by us to evaluate downstream data transmission in CRFIDs.

3 WISP Hardware and CRFID Networks: A Brief Look

In this section we present brief information about the WISP hardware, in particular its clock system. For the sake of clarity, we also give details about a typical CRFID network, which we used to implement and evaluate our synchronization methods in this article.

3.1 Low Power Operation Modes and The Clock System

The WISP 5.0 platform comes with the MSP430FR5969 [41] microcontroller (later called as MSP430 in this article) with FRAM non-volatile memory. The MSP430 clock system includes (i) a 32 kHz external crystal oscillator, (ii) an internal very-low-power low-frequency oscillator, and (iii) an integrated internal digitally controlled oscillator (DCO). The five built-in 16-bit timers (TA0–TA3, TB0) in this system can be clocked with the auxiliary clock (ACLK) signal that is sourced from the relatively stable external 32 kHz oscillator. The MSP430 has one active mode and seven software selectable low-power operation modes. Low Power operation Mode 3 (LPM3) is the standby mode where CPU, FRAM and high-frequency peripherals are off; the external 32 kHz oscillator and the ACKL signal are active. In our target implementations, we configured TB0 to be clocked with ACKL so that it has 16-bit 32 kHz precision and runs continuously in LPM3 mode.

3.2 An Experimental CRFID System

As presented in Fig. 3, a typical CRFID system is composed of an RFID reader, a single WISP tag placed inside the communication range of this reader and a host computer to control this reader. In our testbed, we used a 915 MHz Impinj Speedway R1000 RFID reader with firmware version 3.2.4 connected to a Laird S9028PCR 8.5 dBic gain antenna. We placed the WISP tag at the line-of-sight from the reader antenna. For the host-reader control operations, we used sllurp [42], a LLRP (Low-Level Reader Protocol) [43] control library written in Python. To program the WISP tag, we used MSP430 Flash Emulation Tool (FET), in combination with TI Code Composer Studio (CCS), attached to the host. All of our experimental evaluation presented in the next sections were conducted using this setup at a university office with human presence.

3.3 A Sniffer for the CRFID System

Communication delays are one of the main sources of errors in synchronization [20] and they should be kept as small as possible. In order to characterize the communication delays between the RFID reader and the WISP tags during backscatter communication and understand the effect of these delays on the synchronization error, we used USRP 210 software-defined radio, another Laird antenna placed at 50 cm from the tag and the GNU Radio toolkit [44] with the custom software to sniff the radio packets. The sniffer was presented first in [45] and we refer the reader to this study for more details. Implementing sniffing mechanism allowed us to store the timings of the transmission and reception of the EPC Gen 2 packets, which we later processed offline to measure the actual transmission delays. We will present these measurements in the next sections.

4 System Model, Notation and Preliminaries

In this section, we provide a mathematical framework that models the built-in clocks and in particular the clock drifts of the WISP tags and the RFID reader. We will use these models to prove the correctness of our synchronization algorithms and to derive mathematical expressions that represent their synchronization performance. In this framework, we denote a single WISP tag by w and the RFID reader by r. The Timer B0 mentioned in Section 3.1 can be considered as the built-in clock, i.e. *local clock*, of the tag w, which we denote by C_w. The value of C_w at time t can be modeled as

$$C_w(t) \quad \triangleq \quad C_w(t_0) + \int_{t_0}^{t} f_w(u)du, \tag{1}$$

where t_0 represents the time at which tag w is powered on and $f_w(u)$ represents the *instantaneous oscillator frequency* of C_w at time u. Since the frequency of the oscillators are affected by environmental factors such as temperature and voltage level, they are not stable [46] and f_w is not a constant but it is *time varying*, see Fig. 4. From (1), it can be noticed that C_w is *unitless*. Indeed, it can be seen as a counter of oscillation events that are produced at a frequency of f_w.

The RFID reader r can be considered as a natural time reference for the tag w. We denote the local clock of the RFID reader at time t by $C_r(t)$ and its frequency by $f_r(t)$. By collecting (time) information from the RFID reader, the tag w runs a *clock synchronization algorithm* to calculate a *software clock*, denoted by $S_w(t)$, whose value represents the synchronized notion of time. The objective of the synchronization algorithm is minimizing the *synchronization error* with respect to the reader r at any time t, which is defined as

$$\gamma(t) \quad \triangleq \quad S_w(t) - C_r(t). \tag{2}$$

Considering the individual clock frequencies f_w and f_r, we denote the *instantaneous relative clock frequency* of the WISP tag with respect to the RFID reader at time t as

$$f_w^r(t) \quad \triangleq \quad \frac{f_w(t)}{f_r(t)} - 1. \tag{3}$$

5 Sender-Receiver Based Reader-Tag Synchronization in CRFID Systems

In this section we provide initial observations, design and implementation of a *sender-receiver* based synchronization approach that synchronizes a single WISP tag to the RFID reader and we reveal its limitations in current CRFID networks. In sender-receiver based synchronization mechanisms, receiver devices synchronize to the clock of a reference sender device. In order to synchronize itself to the RFID reader with such a mechanism, the WISP tag should obtain several $(C_w(t), C_r(t))$ synchronization points to establish a *relationship* between its local clock C_w and the reader clock C_r, represented by its software clock S_w. The value $S_w(t)$ will provide an estimate of the reference clock $C_r(t)$ at any time instant t.

We explored the EPC Gen2 standard and the LLRP protocol and found out that LLRP assigns a *FirstSeenTimestamp* in UTC (Coordinated Universal Time), which is defined as *"The Reader SHALL set it to the time of the first observation amongst the tag reports that get accumulated in the TagReportData"* [43, p. 87]. From this definition, we assume that this timestamp is assigned by the reader when it receives the EPC during the handshake operation with the corresponding tag, shown in Fig. 5 as the timestamp assigned at time t_1. Therefore, the *FirstSeenTimestamp* can be considered as $C_r(t_1)$. In order to obtain the corresponding local time $C_w(t_1)$, one strategy is to force the reader to send a special "synchronization" command after the handshake so that the tag timestamps the command reception event using its local clock, shown in Fig. 5 as time t_2. As we emphasized in Section 2.1, transmission delays are the main source of errors in time synchronization and they should be kept as deterministic as possible. We denote the *transmission delay* in this case by $\Delta t = t_2 - t_1$ and it is desirable to keep the *variation* of Δt, i.e. *jitter*, as small as possible [20].

5.1 Transmission Delay and Jitter Measurements

In order to characterize the transmission delay Δt, we placed the WISP tag 20 cm away from the antenna using the setup presented in Fig. 3 and we sniffed the communication between the RFID reader and the WISP tag during the communication scenario of Fig. 5. Fig. 6 presents a summary of our measurements (with 300 samples gathered). We observed that the transmission delay is distributed with a mean of 1.89 ms and standard deviation of 0.0164 ms. Moreover, we observed some outliers, presented at the right

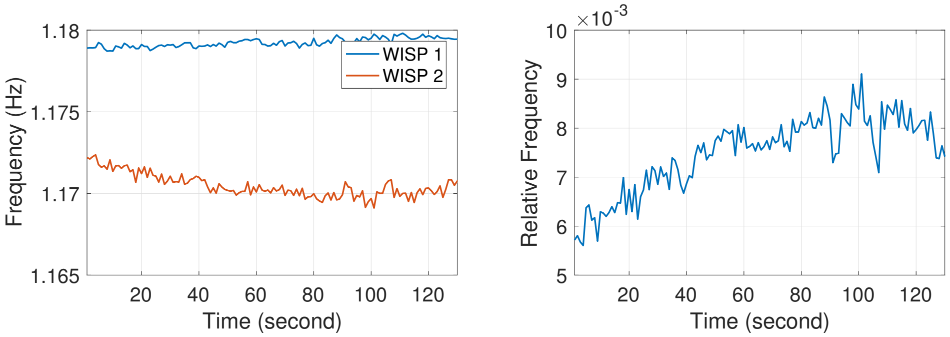

Fig. 4. The frequencies of the toggle events (obtained using the same steps presented in Fig.2) of two RF-powered WISP tags, placed at 5 cm distance from the RFID reader. On the left the individual event frequencies (f_1 and f_2) are presented, and their relative clock frequency ($\frac{f_1}{f_2} - 1$) is presented on the right. We measured the mean and standard deviation of f_1 as 1.1793 and 0.00026049 and those of f_2 as 1.1705 and 0.00068239, respectively.

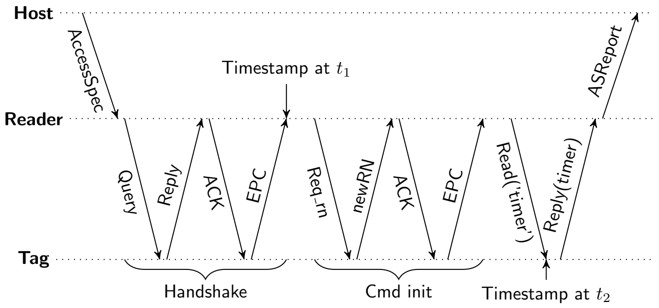

Fig. 5. The message exchange among the host computer, the RFID reader and the tag for sender-receiver synchronization. The host machine, i.e. a PC, sends the high level commands to the reader via LLRP *AccessSpec* message and receives the results through an *ASReport*. The RFID reader follows the steps defined in the EPC Gen2 standard: (i) performs a *Handshake* to initialize the communication with the active tag; (ii) performs a *Command Initialization* (Cmd Init) by requesting a random number from the tag (Req_n) and receiving the random number (newRN); (iii) performs a *Read* command by sending its request and receiving the timer value. The reader assigns the *FirstSeenTimestamp* to the tag at time t_1 and the tag timestamps the command reception event at time t_2 with its local clock reading.

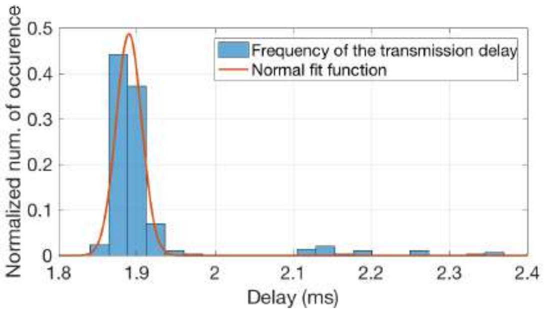

Fig. 6. The normalized number of occurrences of the transmission delays measured by sniffing the communication between the RFID reader and the WISP tag. We calculated the mean transmission delay and its standard deviation as 1.89 ms and 0.0164 ms with a 99% confidence interval of [1.8874,1.8925] and [0.0148,0.0184], respectively.

hand side of the figure, which we attribute to the EPC Gen2 implementation of the Impinj reader.

5.2 Collection of Clock Values for Off-line Processing

After characterizing the variation of the transmission delay, the next step was the collection of the clock values from the RFID Reader and the WISP tag in order to try synchronization mechanisms offline, which allowed to evaluate the performance of sender-receiver synchronization mechanism in CRFID networks. We collected $(C_w(t_2), C_r(t_1))$ pairs for offline processing by controlling the RFID reader to send a *Read* command to the tag and by programming the WISP tag so that it backscatters $C_w(t_2)$ upon receiving this command. The received pairs $(C_w(t_2), C_r(t_1))$ were logged by the host computer.

5.3 Software Clock Computation Using Least-Squares

Following studies [20], [26], [28], [29] that use a linear clock model, we assume a *linear* relationship between C_r and C_w and we model the software clock of the WISP tag as $S_w(C_w(t)) = a + bC_w(t)$ where S_w represents an estimator of the reader's clock C_r, a is the *offset* and b is the *relative speed* with respect to the local clock C_w. To establish such a relationship, we performed *least-squares regression* offline in MATLAB using the collected pairs in the previous subsection. As in [20], since the WISP tag has limited memory, computation capability and energy, at each step only the most recent N pairs are used to estimate the regression line.

Formally, let $[C_w(t_k), C_r(t_k)]$ denote the kth pair in the log file where t_k denotes the real-time at which *FirstSeenTimestamp* has been assigned during the collection of this pair. At each kth step, the pairs $\{[C_w(t_k), C_r(t_k)],\ldots,[(C_w(t_{k+N-1}), C_r(t_{k+N-1})]\}$ are used to calculate the parameters a and b, i.e. the *intercept* and the *slope* of the estimated regression line [47], using polyfit function of MATLAB. We calculated the synchronization error as $\gamma(t_{k+N}) \triangleq S_w(C_w(t_{k+N})) - C_r(t_{k+N})$ that represents the difference between the predicted reference time and the received reference time. In our implementation we used $N = 8$ as in [20] and Fig. 7 presents the synchronization error at each step. We observed a maximum synchronization error of 0.32 ms in this one-hop network, which is more than one order of magnitude larger than the

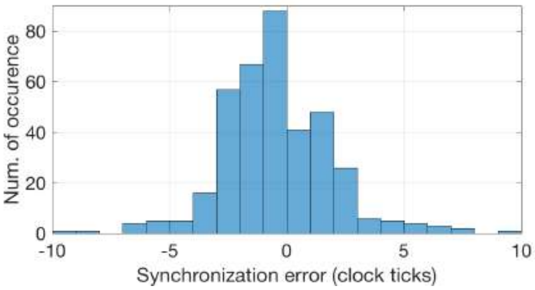

Fig. 7. Synchronization error by employing least-squares regression on the collected timestamps. We observed a maximum synchronization error of 10 clock ticks between the RFID reader and the WISP tag, leading to 0.32 ms synchronization accuracy.

synchronization performance of the de facto WSN solution [20], which was reported as approximately 10 μs.

5.4 Limitations of the Approach

In addition to the challenges listed in Section 2.2, we observed two crucial limitations for the WISP platform, which prevents to build up a sender-receiver synchronization building block:

5.4.1 Host computer computation

We are unaware of any EPC Gen2 command that will allow to send the *FistSeenTimestamp* to the tag. Hence, even though we were able to collect $(C_w(t), C_r(t))$ pairs for offline processing, it is not possible for the tag to collect $C_r(t)$ and synchronize itself to the reader. Therefore, with this limitation, only a host computer can collect and log the timestamps, calculate the relationship between the clock of the tag and the clock of the reader and send the data that represents this relationship to the tag for synchronization, making this method only of theoretical nature as of now.

5.4.2 The Lack of broadcast primitive

Since the RFID reader assigns the *FirstSeenTimestamp* for each tag, the synchronization steps in Fig. 5 should be repeated for each tag to obtain synchronization in the communication domain of the reader.

5.4.3 Big and reader dependent transmission delays

As depicted previously, we observed that transmission delays in our testbed were on the order of milliseconds, see Fig. 7, whereas they are reported as a few microseconds in conventional WSN platforms. This is one of the crucial limitations since the smaller the transmission delays are, the better the synchronization.

6 Event-Based Reader-Tag Synchronization In CRFID Systems

In this section, we introduce a simple but novel *event-based* reader-tag synchronization inspired by simple PI controllers and provide its advantages over the previous approach. In event-based synchronization, a common event which is observable by all receiver devices simultaneously is generated by the reference device. Upon receiving events generated at regular intervals, receiver devices can predict the occurrence time of future events. In order to synchronize the tag with such a mechanism, the RFID reader does not send explicit timestamp values as in sender-receiver based synchronization but instead generates events at regular intervals. Upon observing these events, the receiver tag adjusts the rate of its software clock so that it predicts the occurrence of the next event precisely. We explored the EPC Gen2 standard to see how to generate events at regular intervals and realized that the *BlockWrite* operation allows this feature.[1] Fig. 8 presents the steps of event-based synchronization. During the command phase, the EPC Gen2 standard allows a maximum of eight successive *BlockWrite* operations. It is desirable to use the first and the last *BlockWrite* events for synchronization since it is better to compensate frequency differences observed in longer time intervals to adjust the software clock.

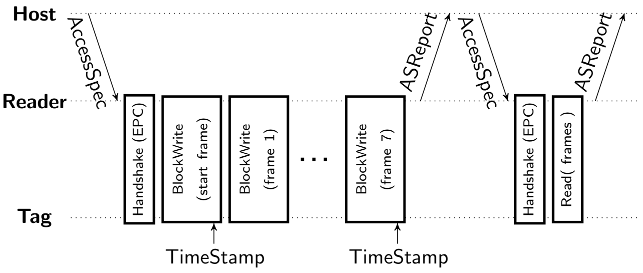

Fig. 8. Event-Based synchronization steps: The tag timestamps successive *BlockWrite* events and adjusts its software clock.

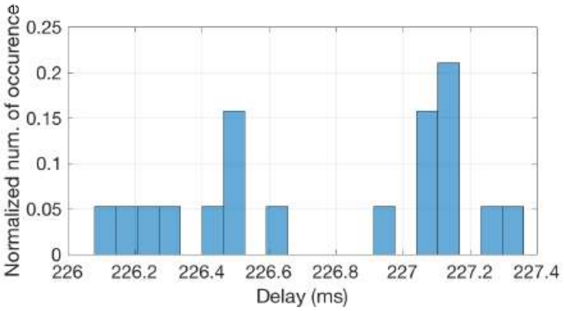

Fig. 9. The delay between the first and the last *BlockWrite* event, i.e. *event period*, by considering 20 samples during the communication scenario in Fig. 8. We measured its mean and standard deviation as 226.7667 ms and 0.4097 ms with a 99% confidence interval of [226.4961,227.0372] and [0.2852,0.6945], respectively.

6.1 Event Period Measurements

The real-time length between the first and the last *BlockWrite* operation is the *event period*, we denote by τ, and its variation is the main error source. Therefore, it is important

[1] It is possible to exploit modulation properties of signals from the reader to tags (such as bit timing in FM0, i.e. Miller 1, modulation) in order to get the timing measurements of the regular patterns in the signal. However, this requires using a high-precision clock to obtain microsecond-precision timing measurements. In this study, we used the higher-level properties of the EPC Gen2 standard, rather than exploiting its low-level signal properties. By means of this, we could use the external 32 KHz clock that runs even in low-power standby mode (see Section 3) to obtain timing measurements.)

to explore its characteristics. We sniffed the communication protocol between the RFID reader and a tag presented in Fig. 8. We took 20 sample measurements about the *event period*, which is summarized in Fig. 9. According to our measurements, the *event period* was distributed with a mean of 226.76 ms and standard deviation of 0.41 ms; respectively. We would like to mention that these values are dependent on the RFID Reader implementation of the EPC Gen2 protocol.

6.2 Event-Based Synchronization Inspired By PI Controllers

For the computation of the software clock, *adaptive* and *lightweight* solutions are crucial due to the following important requirements:

- Since the voltage level is not stable and the harvested power is limited, the computations should demand little amount of energy to calculate the software clock, i.e. marginal number of steps—*Addressing Challenge I in Section 1.2*.
- Since power is intermittent, the number of variables pertaining to the software clock should be marginal so that saving the state of the synchronization to non-volatile memory will demand little time and less energy. Therefore, lightweight computation methods in terms of main memory overhead are required—*Addressing Challenge I in Section 1.2*.
- Unstable voltage level leads to frequently varying clock frequencies. Therefore, the computation methods should adapt such dynamic conditions in a very fast manner to keep synchronization accuracy stable—*Addressing Challenge II in Section 1.2*

Considering these facts, we designed a lightweight and adaptive clock synchronization approach inspired by the PI-controller based solution introduced in [24], [25]: (i) we apply PI-controller based clock computation technique to the WISP platform due to its computational efficiency and fast adaptation; (ii) we introduce power-interrupt recovery mechanism so that the WISP tags do not loose their synchronization-related information. We modified the algorithm in our previous work so that PI-controllers use the periodic event occurrences rather than explicit clock values of a master node. We also provide a moderately different but more realistic theoretical analysis by using a more realistic clock drift model, i.e. Brownian motion rather than uniform distribution—see Appendix.

Let t_0 denote the first event reception time at the tag. Using the pre-measured event period τ of the RFID reader, the tag can estimate the next event reception time at time t_1 as

$$\hat{C}_w(t_1) \quad = \quad C_w(t_0) + \tau(1 + \hat{f}^r_w(t_0)), \tag{4}$$

where $\hat{f}^r_w(t_0)$ denotes the *estimated relative frequency* at time t_0, which is an estimate for (3). The intuition behind (4) is that $1 + \hat{f}^r_w(t_0)$ gives the estimated value of $f_r(t_0)/f_w(t_0)$, $\tau f_r(t_0)$ gives the estimated local time passed on the reader side and $\tau f_r(t_0)/f_w(t_0)$ gives the estimated number of clock ticks on the tag side that will pass until the next packet reception. Redefining (2), the *prediction error* at time t_1 can be written as

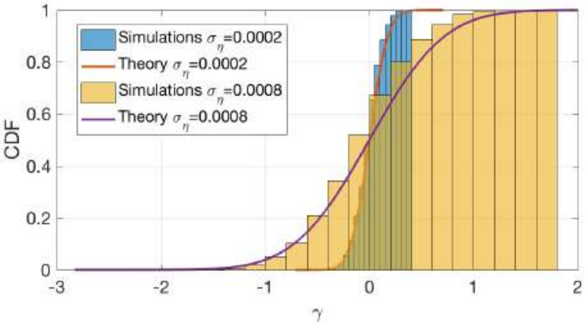

Fig. 10. The plot of the cumulative distribution function (CDF) of γ in (7) and the CDF of the steady-state γ obtained during numerical simulations of (6) (by implementing (15) and (16) in Appendix). We used $\tau = 1$, $\beta = 0.0001$, $\sigma_\tau = 0.00005$ and our plots depict the theoretical and simulation results with respect to $\sigma_\eta = 0.0002$ and $\sigma_\eta = 0.0008$. It can be observed that (7) is consistent with the simulations and bigger uncertainty leads to bigger synchronization error.

$$\gamma(t_1) = C_w(t_1) - \hat{C}_w(t_1) = C_w(t_1) - C_w(t_0) - \tau(1 + \hat{f}^r_w(t_0)). \tag{5}$$

Apparently, the objective is to minimize the prediction error so that the event reception time is estimated correctly. As can be observed from (5), the estimation error on the relative clock frequency at time t_0, i.e. $\hat{f}^r_w(t_0)$, depends on the mean event period τ, and its variance (since they affect $C_w(t_1) - C_w(t_0)$). Inspired by the integral controller update rule in [24, Section III, (5)], we propose to update the estimated relative frequency at time t_1 as

$$\hat{f}^r_w(t_1) \quad = \quad \hat{f}^r_w(t_0) + \beta\gamma(t_1), \tag{6}$$

where β is the *integral gain*. Assume that the uncertainty of the event period is $\varepsilon \sim \mathcal{N}(0, \sigma^2_\tau)$ and the instantaneous change on the relative clock frequency is $\frac{df^r_w(t)}{dt} \sim \mathcal{N}(0, \sigma^2_\eta)$. Under these assumptions, it can be proven that if $0 < \beta < \frac{2}{\tau}$ is satisfied the synchronization will hold eventually; i.e. $\hat{f}^r_w = f^r_w$. Moreover, the mean and the variance (σ^2_γ) of the steady-state (asymptotic) estimation error of WISP-Sync can be given by

$$\gamma \sim \mathcal{N}\left(0, \sigma^2_\gamma = \frac{\sigma^2_\eta\left(\tau^2 - \beta\tau^3 + \frac{\beta^2\tau^4}{3}\right) + \beta^2\sigma^2_\tau\tau}{(2\beta - \beta^2\tau)} + \frac{\sigma^2_\eta\tau^3}{3} + \sigma^2_\tau\right), \tag{7}$$

which is derived in Section A.2 of the Appendix.

In Fig. 10, we emphasize that our theoretical derivation in (7) is consistent with the simulations of (6). We refer again the reader to the Appendix for the detailed proofs of the convergence and the asymptotic estimation error. Moreover, in Section 6.2.4, we discuss the impact of the parameters τ and β on the synchronization accuracy.

6.2.1 *The Synchronization Algorithm*

Based on this model we introduce Algorithm 1, namely *WISP-Sync*, that synchronizes the speed of the software clock to the reference clock's speed using the update rule defined in (6). The steps of this algorithm can be explained as follows. Initially, the tag lets its software clock run at the same speed of its local clock by setting its relative clock rate

Algorithm 1 *WISP-Sync*: An integral controller-based algorithm at WISP tag side.

```
Definitions:
  $t_f$                                         ▷ local time of the first BlockWrite
  $C_w$                                         ▷ bult-in clock of the WISP tag
  $\hat{f}_w^r \leftarrow 0$                    ▷ estimated relative clock frequency
  $\tau \leftarrow$measured value               ▷ mean event period
  $\beta \leftarrow$ a value satisfying $0 < \beta < \frac{2}{\tau}$   ▷ integral gain
  $V_{\min}, V_{\text{op}}$                     ▷ voltage thresholds ($V_{\text{op}} > V_{\min}$)
  $V_{\text{cur}}$                              ▷ current voltage
1: □ Upon receiving the first BlockWrite
2:     $t_f = C_w$                              ▷ store the local time in $t_f$
3: □ Upon receiving the last BlockWrite
4:     $\gamma = C_w - (t_f + \tau(1 + \hat{f}_w^r))$   ▷ calculate estimation error $\gamma$
5:     $\hat{f}_w^r = \hat{f}_w^r + \beta\gamma$        ▷ apply integral control
6: □ Upon $V_{\text{cur}} = V_{\min}$
7:     store $\hat{f}_w^r$ into FRAM            ▷ store in non-volatile memory
8: □ Upon $V_{\text{cur}} < V_{\text{op}}$
9:     restore $\hat{f}_w^r$ from FRAM          ▷ restore the clock frequency
```

to zero. Upon receiving the first *BlockWrite* event (Line 1), the tag stores its local clock value at the variable t_f (Line 2). After receiving the last *BlockWrite* event (Line 3), first it calculates the estimation error (Line 4). Then, it applies the correction on its estimated relative rate $\hat{f}_w^r$ by multiplying the estimation error with the integral gain β and adding it to $\hat{f}_w^r$ (Line 5). After receiving successive events, $\hat{f}_w^r$ will converge to its desired value eventually, as proven in the Appendix.

6.2.2 State Recovery

Due to intermittent RF power, the tag is subject to frequent power losses, which will lead to loss of the synchronization state. In voltage-aware systems [19], [48], [49], [50], the general idea is to save the system state when the input voltage is below a pre-defined threshold $V_{\min}$ and then recover it back when the input voltage is above a pre-defined threshold V_{op}. As shown in Algorithm 1, the only part of the system state pertaining to synchronization is the variable $\hat{f}_w^r$. Therefore, if this variable is also saved to FRAM (when there is a power loss and recovered when there is sufficient voltage) the synchronization will resume from the point of power loss, that will prevent loss of synchronization and lead to improved synchronization accuracy. These steps are summarized in Lines 6–9 of Algorithm 1. It should be noted that these steps require hardware support so that the voltage level is monitored continuously and the corresponding voltage level indicator events are generated. Another approach can be keeping $\hat{f}_w^r$ in non-volatile memory as a *persistent variable* so that the corresponding FRAM memory location of $\hat{f}_w^r$ is updated immediately when this variable is updated—eliminating the need for Lines 6–9 of Algorithm 1, and in turn extra voltage measurement hardware, but introducing more FRAM access overhead.

6.2.3 Computation Complexity

It can be observed that WISP-Sync employs only very simple arithmetic operations at each step, e.g. Lines 3–5 are composed of only two subtraction, two addition and two multiplication operations. Moreover, the *state of synchronization*, i.e. the parameters that are required to be saved in the non-volatile memory, is represented by only the variable $\hat{f}_w^r$. As a consequence, WISP-Sync addresses *Challenge I* in Section 1.2 in terms of computation and memory overhead.

6.2.4 Adaptability

In Section 6.1 we presented our measurements pertaining to the event period and showed that $\sigma_\tau = 0.4097$. Moreover, from Fig. 4, we calculated $\sigma_\eta \cong 0.0008$ using the normfit function of MATLAB. Substituting these values in (7), we performed MATLAB simulations to calculate σ_γ^2, which are depicted in Fig. 11. These simulations led us to reveal the following points:

- Considering our previous measurements, it is apparent that σ_τ^2 is the dominant factor effecting the synchronization performance. Therefore, it is desirable to keep transmission delay variations and also τ, as small as possible. Apparently, the bigger the resynchronization period τ, the worse the synchronization error is (see left hand side of Fig. 11).
- Parameter β has significant impact on the performance of the WISP-Sync. From (7), it can be concluded that the synchronization error can be minimized using a proper value of β. However, the β value that minimizes (7) depends on the parameters σ_η and σ_τ which represent randomness of the system. In Fig. 11 (right hand side), we fixed τ and σ_τ, we used different values of σ_η and we plotted σ_γ^2 with respect to the values of the β that will lead to convergence. Observe that a different β value minimizes σ_γ^2 for each different σ_η.
- Due to varying RF power levels, σ_η is also a time-varying value. Therefore, β should be selected to compensate for also *dynamic* σ_η in a *fast manner* to keep synchronization error within desired bounds—which addresses *Challenge II* in Section 1.2. However, the β that minimizes σ_γ^2 may lead to longer convergence time (see Appendix). Therefore, β also defines how *reactive* WISP-Sync is against the dynamic environment, i.e. the *adaptivity*.

7 Evaluation of WISP-Sync

In order to evaluate WISP-Sync, we considered the value of γ in Algorithm 1 (the estimation error) as an evaluation metric. We collected the local clock readings of the tag at the first and last *BlockWrite* events using our testbed setup and sniffer, and used MATLAB to implement WISP-Sync algorithm which processes the collected clock values to output the software clock. This gave us the flexibility to try different approaches without reprogramming the tag.

7.1 Selection of the Integral Gain β

Since we measured the event period as 226.76 ms on average (see Fig. 9), we set $\tau = 7086$ clock ticks since the local clock of the tag is operating at 32 kHz and each clock tick occurs every 32 microseconds. First, we explored how β affects the performance of the algorithm. By substituting τ into (19), the convergence condition becomes $0 < \beta < 0.000284$ in

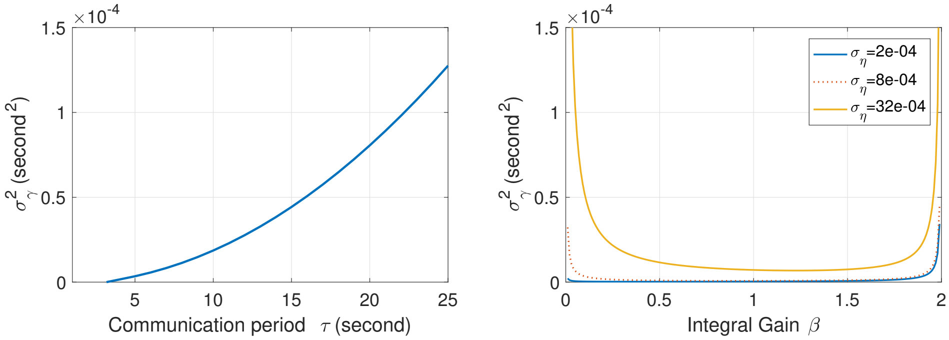

Fig. 11. Numerical results showing the relations between event period, integral gain and the steady-state variance. First, we set $\sigma_\tau = 0.4097$, $\sigma_\eta = 0.0008$, $\beta = 1$ and plotted σ^2_γ using different τ (left hand side). Observe that increasing τ increases σ^2_γ. Then, we set $\sigma_\tau = 0.4097$, $\tau = 1$ and $\beta \in (0, 2)$ due to (19). Using different values for σ_η, we plotted how σ^2_γ changes with respect to the β (right hand side). Observe that for each value of σ_η, there is a particular β that minimizes σ^2_γ.

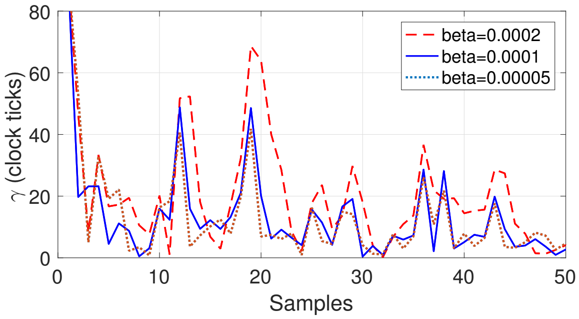

Fig. 12. The synchronization error tends to get smaller with smaller integral gains, but after some point decreasing it has no significant effect due to the precision of 32 kHz clock. Hence, we selected a fixed value of $\beta = 0.0001$ for the next steps.

our case. We present the synchronization error with different β values in Fig. 12. Observe that the synchronization error tends to get smaller with smaller β values but the convergence time increases negligibly. Moreover, due to the low-precision 32 kHz clock, decreasing β has no significant effect after some point. We mention that in order to make WISP-Sync adapt faster, WISP-Sync can start with a big value of β to converge fast and then decrease β gradually to decrease the synchronization error, i.e. the adaptive strategy proposed in [24]. However, from Fig. 12, we consider a fixed value of $\beta = 0.0001$ for the next evaluation steps.

7.2 Synchronization Performance

Fig. 13 presents the synchronization error measurements when the tag is powered by using a constant voltage source, and powered only by an RFID reader. Measurements under constant and stable voltage allowed us to observe synchronization under stable clock frequency. We obtained a significant synchronization performance with WISP-Sync—almost a factor of eight less synchronization error as compared to the case where we did not perform any synchronization. It should be noted that even in this case, we observed quite fluctuating synchronization errors due to the varying transmission delays, that led a peak error to appear between samples 50 and 60. Apart from experiments with constant voltage input, measurements under highly varying RF power led us to observe the behavior of synchronization under highly varying clock frequencies. In this case, there is a considerable amount of increase on the error, i.e. approximately twice as much, as compared to the stable voltage case. However, we also obtained considerable improvements with our approach with unstable clock frequencies. WISP-Sync was quite reactive to the frequently varying clock frequencies and provided almost a factor of eight better accuracy.

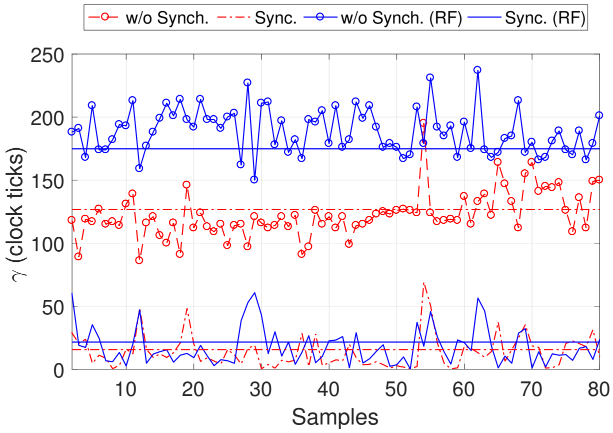

Fig. 13. Event-based synchronization performance when the tag is powered through a constant voltage source and through RF power. Mean synchronization errors without and with WISP-Sync were almost 127 and 16 clock ticks under stable voltage, and 175 and 22 clock ticks with RF power harvesting (represented by the straight lines in this figure), respectively.

7.3 Power Loss and Recovery

In order to evaluate the performance of WISP-Sync under power losses, we simulated power loss in MATLAB by omitting the collected clock readings during predefined time intervals. As presented in Fig. 14, after samples 10 and 30 WISP-Sync is disabled and before the beginning of the sample times 20 and 40 the algorithm is enabled. The solid line in this figure indicates enabling algorithm with state recovery, where the algorithm starts to execute using the value of $\hat{f}^r_w$ previously saved in the FRAM before power loss. It is apparent that with state recovery, the tag quickly adapts itself and re-synchronization is quickly established. On the other hand, starting the algorithm from scratch, as

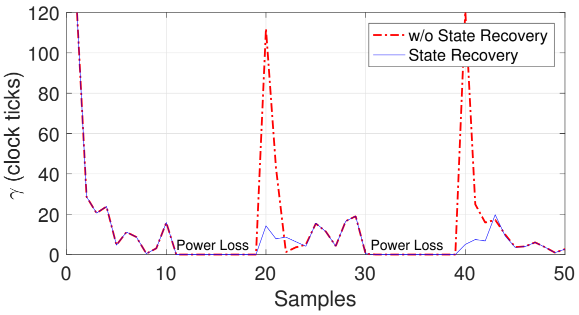

Fig. 14. Impact of the state recovery on the synchronization performance. During the power loss periods, the tag is assumed to be turned off. With state recovery, presented by the solid line, the relative clock frequency is assumed to be saved in the FRAM and recovered back when sufficient power is supplied. On the other hand, the dashed line represents the case of starting synchronization from scratch. It can be observed that with state recovery, the tag adapts itself quickly and synchronization performance is considerably better, i.e. more than 100 clock ticks less synchronization error on power loss.

presented by the dashed lines, is problematic since it takes several steps to re-synchronize the tag.

7.4 Energy Overhead

We now present the real energy measurements of the actual WISP tag-side C language implementation of lines 3–5 of Algorithm 1 which are the main computation steps to calculate the new values of the variables pertaining to the synchronization. We performed our measurements using the Energy-interference-free Debugger (EDB) [51] to measure the output voltage without affecting the energy state of the tag. EDB charges the capacitor of a device to a certain level and then disconnects itself. Using this feature of EDB we charge the capacitor of WISP to 2.45 V, i.e. just above the operational level of WISP. While the program is being executed, the value of the capacitor is measured at specific code locations—in this case before Line 3 and after Line 5 of Algorithm 1. The difference between these two results represents the energy overhead of WISP-Sync. We repeated this experiment ten times to eliminate quantization effects during measurements. We observed an average energy of $\approx 1.85\mu$J with a standard deviation of $0.038\,\mu$J is spent by the tag in order to perform the aforementioned computations.

The main source of energy consumption in our implementation is the *floating point arithmetic*: WISP-Sync requires two multiplications, three additions and one subtraction on float variables. Since MSP430 microcontroller does not have any floating point coprocessor, the float operations are handled using a software library, that makes them relatively slow. We used the standard gcc compiler (specifically, msp430-gcc version 6.2.1.16) math library for the sake of the portability of float operations. However, we expect that a more efficient architecture-specific implementation of the math library in software or an architecture with a math coprocessor will lead to less computation time and in turn relatively less energy overhead for WISP-Sync [52]. Another point we would like to mention is that non-volatile memory access consumes more power as compared to SRAM access [53, Section 3.3]. The only non-volatile variable in WISP-Sync is $\hat{f}_w^r$ in Algorithm 1. This variable is read twice but modified once in our implementation. On the other hand, WISP-Sync maintains only one variable in SRAM, i.e. t_f, which is read only once. We conclude that WISP-Sync performs three FRAM and one SRAM access operations—low energy overhead in terms of memory access.

7.5 Limitations of WISP-Sync

In conclusion, the experimental results so far indicated that a maximum synchronization error of approximately 1.5 ms can be ensured between an RFID reader and a single WISP tag when employing event-based synchronization. The main factors that limited the WISP-Sync synchronization accuracy were the clock precision of the WISP platform and the non-deterministic transmission delays during reader-tag communication. As we indicated, we used the external 32 kHz clock of the platform since it is also running when the tag is in very low-power operation mode. Therefore, in the best case when there are no transmission delays, the quantization effect due to the timing measurements using the low precision clock will not allow a better synchronization accuracy than $32\,\mu$s. On the other hand, to obtain this synchronization accuracy with WISP-Sync, the reader and the tags shoud support a high-precision and accurate packet timestamping as in sensor networks [20]. As indicated in Section 2.2, the effect of the non-deterministic delays should be reduced for a better synchronization accuracy.

Even though the event-based approach is relatively lightweight and adaptive as compared to the sender-receiver synchronization, it has some limitations. First, the tag is unable to obtain an *explicit reference clock value*. The only variable it tunes is the $\hat{f}_w^r$ that represents the relative frequency of the software clock with respect to the clock of the reader. Therefore, we require additional steps that will allow tags to obtain the actual reference clock value. Moreover, if the tags are covered by different readers, the first step should be synchronizing the readers to the real-time, e.g. by allowing them to access time servers using internet. Second, WISP-Sync requires knowledge about τ, which can be RFID reader dependent and requires a sniffer setup described in Section 3.3 to measure. This limitation can be addressed by employing an iterative estimation as an initial step to the WISP-Sync: the tag can start with an initial estimated τ value, and improve its estimation as it receives successive packets from the reader. After some iterations, the tag can have a rough estimate $\hat{\tau}$ that can be used on behalf of τ. However, estimation errors might also perturb the synchronization accuracy. Last, the steps in Fig. 8 present the communication scenario between one WISP and the reader. To allow other tags sniff this communication and synchronize, a broadcast primitive is crucial—in the next section we show how to overcome this limitation.

8 The Synchronization of Multiple WISP Tags Using the WISP-Sync

In this section, we turn our attention to the synchronization of multiple WISP tags using the WISP-Sync. To this end, we have done the *real implementation* of the WISP-Sync using the C programming language and TI's CCS [54], instead of

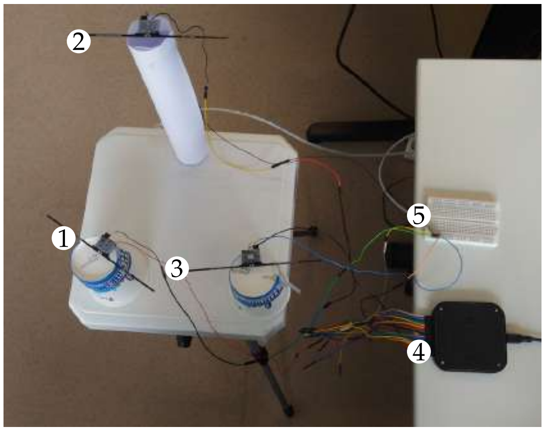

Fig. 15. A testbed of multiple tags: (1,2,3) tags, (4) logic analyzer, (5) breadboard. Tags 1 and 3 were at 8 cm while tag 2 was at 20 cm distance to the reader. The breadboard was used to make connections between the logic analyzer and the tags.

using MATLAB to simulate our protocol using the collected the clock values from our testbed. Moreover, we extended our experimental testbed in Section 3.2 by introducing two additional tags, as presented in Fig. 15.

We programmed the RFID Reader to send synchronization points via *BlockWrite* command with a period of one second and we programmed two tags using our WISP-Sync implementation. On the other hand, we disabled clock synchronization for the last tag—which was a baseline for our evaluation. In order to allow tags to simultaneously receive the information sent by the RFID Reader, we exploited an overhearing mechanism presented first in [45, Section III-B]. Therefore, during the experiments in our testbed (i) all of the tags received the periodic synchronization points sent by the RFID Reader; (ii) only two tags process this information and run Algorithm 1 to update their software clocks; (iii) the last tag did not participate in synchronization.

In order to measure the synchronization accuracies of the tags, we configured one of the ports of the MSP430 microcontroller as an output port and let tags (i) to set this port when they receive the first *BlockWrite* command in Algorithm 1 (Line 1); (ii) to set a timer that will fire at their predicted time of the reception of the last *BlockWrite* command from the RFID Reader; (iii) to clear this port when their timer fires. We connected the output ports to logic analyzer [55] and measured the time difference between the times at which the output ports are cleared. This allowed us to measure the prediction errors among the tags. We observed that all the tags set their output ports when they overhear the first *BlockWrite* command, however their clear times are quite different and depend on their prediction error—the port clear times of the synchronizing two tags were closer to each other while they were quite different than the unsynchronized tag.

In Fig. 16, we plotted two different results: (i) the blue line depicts the maximum synchronization error (i.e. the maximum pairwise difference between the port clear times) when we considered all of the tags; (ii) the red line depicts when only synchronized tags are considered. From this figure it can be concluded that synchronization led to a factor of approximately four times better prediction on average among the tags: we observed an average synchronization error of 1 ms while it was 4 ms when one of the tags was unsynchronized.

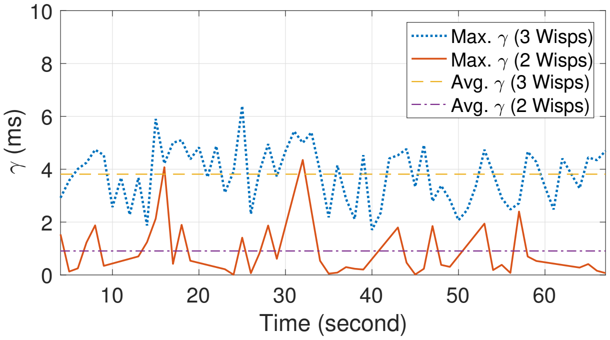

Fig. 16. The synchronization error calculated during the experiments in our testbed. The blue dotted line denotes the maximum synchronization error among all of the tags (including the one which did not participate in synchronization) whereas the red line denotes the maximum error just between the synchronized tags.

9 Conclusions and Future Research Directions

In this article we explored synchronization scenarios between an RFID reader and WISP tags. We studied sender-receiver and event-based synchronization mechanisms in this setting and provided initial designs that will guide future explicit synchronization mechanisms among individual WISPs that reside inside the communication range of a common RFID reader. We provided implementation and evaluation of these designs in our testbed and identified their limitations and drawbacks. Our main finding is that with lightweight mechanisms, as of now, an average synchronization error of approximately 1 ms can be ensured among tags. We provide the following issues for future studies in this domain:

- **Network-wide synchronization:** We studied event-based synchronization among many tags. However, synchronization of these tags where they can access the explicit clock of the RFID reader and also synchronization of the whole CRFID network composed of several RFID readers and tags is still an issue, due to the single-hop nature of backscatter communication. Since tags can only communicate with the reader, not with their neighboring tags, it is interesting to explore *WISP to WISP* synchronization.
- **Power loss and recovery:** It is crucial to save the synchronization status to non-volatile memory just before the power loss, but writing to non-volatile memory also consumes energy. Hence, when to save the synchronization status without extra hardware support is worth exploring [19], [48], [49], [50].
- **Voltage-frequency relations:** It might be interesting to explore the voltage-clock frequency relationship since1 tags are subject to varying voltage source when they are powered with only RF energy harvesting from the RFID reader. We anticipate that voltage level should be incorporated to the establishment of

the software clock so that voltage dependent instability of clock frequency can be compensated. However, this is a "chicken or egg" problem since reading the voltage level also consumes energy.

- **Synchronization in other IPD platforms:** Synchronization in other IPD platforms, e.g. that use ambient backscatter communication [56], is worth exploring as well.

References

[1] K. S. Yıldırım, H. Aantjes, A. Y. Majid, and P. Pawełczak, "On the synchronization of intermittently powered wireless embedded systems," *CoRR*, vol. abs/1606.01719, 2016. [Online]. Available: http://arxiv.org/abs/1606.01719

[2] S. Gollakota, M. Reynolds, J. Smith, and D. Wetherall, "The emergence of RF-powered computing," *Computer*, vol. 47, no. 1, pp. 32–39, Jan. 2014.

[3] J. R. Smith, *Wirelessly Powered Sensor Networks and Computational RFID*. Springer Science & Business Media, 2013.

[4] H. Gao, M. Matters-Kammerer, P. Harpe, D. Milosevic, A. van Roermund, J.-P. Linnartz, and P. Baltus, "A 60-GHz energy harvesting module with on-chip antenna and switch for co-integration with ULP radios in 65-nm CMOS with fully wireless mm-wave power transfer measurement," in *Proc. IEEE ISCAS*, June 1–5 2014.

[5] S. Bi, C. K. Ho, and R. Zhang, "Wireless powered communication: Opportunities and challenges," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 117–125, Apr. 2015.

[6] L. Xie, Y. Shi, Y. T. Hou, and W. Lou, "Wireless power transfer and applications to sensor networks," *IEEE Wireless Commun. Mag.*, vol. 20, no. 3, Aug. 2013.

[7] J. Garnica, R. A. Chinga, and J. Lin, "Wireless power transmission: From far field to near field," *Proc. IEEE*, vol. 101, no. 6, Jun. 2013.

[8] K. Finkenzeller, *RFID Handbook*. John Wiley & Sons, Ltd., 2010.

[9] P. Zhang, P. Hu, V. Pasikanti, and D. Ganesan, "Ekhonet: High speed ultra low-power backscatter for next generation sensors," in *Proc. ACM MobiCom*, Sept. 7–11 2014.

[10] A. Sample, D. Yeager, P. Powledge, A. Mamishev, and J. Smith, "Design of an RFID-Based Battery-Free Programmable Sensing Platform," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 11, pp. 2608–2615, Nov. 2008.

[11] "EPC Radio-Frequency Identity Protocols Generation-2 UHF RFID; Specification for RFID Air Interface Protocol for Communications at 860 MHz–960 MHz Ver. 2.0.1," 2015. [Online]. Available: http://www.gs1.org/

[12] J. Holleman, D. Yeager, R. Prasad, J. R. Smith, and B. Otis, "NeuralWISP: An energy-harvesting wireless neural interface with 1-m range," in *Proc. IEEE BioCAS*, Nov. 20–22 2008.

[13] S. Naderiparizi, A. Parks, Z. Kapetanovic, B. Ransford, and J. Smith, "WISPCam: A battery-free RFID camera," in *Proc. IEEE RFID*, Apr. 15–17 2015.

[14] S. Naderiparizi, Y. Zhao, J. Youngquist, A. P. Sample, and J. R. Smith, "Self-localizing Battery-free Cameras," in *Proc. UbiComp*, Sept. 7–11 2015.

[15] J. Tan, P. Pawelczak, A. Parks, and J. R. Smith, "Wisent: Robust downstream communication and storage for computational rfids," in *Proc. INFOCOM 2016*, 10-15 April 2016.

[16] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient backscatter: wireless communication out of thin air," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 39–50, 2013.

[17] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," in *in Ambient Intelligence*. Springer Verlag, 2004.

[18] B. Ransford, S. Clark, M. Salajegheh, and K. Fu, "Getting Things Done on Computational RFIDs with Energy-aware Checkpointing and Voltage-aware Scheduling," in *Proc. HotPower*, Dec. 2008.

[19] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," in *Proc. ASPLOS XVI*, March 05–11 2011.

[20] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proc. ACM SenSys*, Nov. 3–5 2004.

[21] T. Schmid, R. Shea, Z. Charbiwala, J. Friedman, M. B. Srivastava, and Y. H. Cho, "On the interaction of clocks, power, and synchronization in duty-cycled embedded sensor nodes," *ACM Transactions on Sensor Networks (TOSN)*, vol. 7, no. 3, p. 24, 2010.

[22] T. Schmid, Z. Charbiwala, R. Shea, and M. B. Srivastava, "Temperature compensated time synchronization," vol. 1, no. 2, pp. 37–41, Aug. 2009.

[23] K. S. Leong, M. L. Ng, A. Grasso, and P. Cole, "Synchronization of RFID readers for dense RFID reader environments," in *Proc. SAINTW*, Jan. 23–27 2006.

[24] K. Yildirim, R. Carli, and L. Schenato, "Adaptive control-based clock synchronization in wireless sensor networks," in *Proc. ECC'15*, July 15–17 2015.

[25] K. S. Yıldırım, R. Carli, and L. Schenato, "Adaptive proportional-integral clock synchronization in wireless sensor networks," *IEEE Transactions on Control Systems Technology*, 2017.

[26] T. Schmid, P. Dutta, and M. B. Srivastava, "High-resolution, low-power time synchronization an oxymoron no more," in *Proc. IPSN '10*, April 12–16 2010.

[27] Texas Instruments. (2015) Cc2420 (nrnd): Single-chip 2.4 ghz ieee 802.15.4 compliant and zigbee ready rf transceiver. [Online]. Available: http://www.ti.com/product/CC2420

[28] C. Lenzen, P. Sommer, and R. Wattenhofer, "Pulsesync: An efficient and scalable clock synchronization protocol," *IEEE/ACM Trans. Networking*, vol. 23, no. 3, pp. 717–727, Jun. 2015.

[29] R. Lim, B. Maag, and L. Thiele, "Time-of-flight aware time synchronization for wireless embedded systems," in *Proc. EWSN'16*, February 2016.

[30] K. S. Yildirim and Ö. Gürcan, "Efficient time synchronization in a wireless sensor network by adaptive value tracking," *IEEE Trans. Wireless Commun.*, vol. 13, no. 7, pp. 3650–3664, Jul. 2014.

[31] K. S. Yildirim, "Gradient descent algorithm inspired adaptive time synchronization in wireless sensor networks," *IEEE Sensors Journal*, vol. 16, no. 13, pp. 5463–5470, July 2016.

[32] P. Sommer and R. Wattenhofer, "Gradient Clock Synchronization in Wireless Sensor Networks," in *Proc. IPSN*, Apr. 13–16 2009.

[33] L. Schenato and F. Fiorentin, "Average TimeSynch: a consensus-based protocol for time synchronization in wireless sensor networks," *Automatica*, vol. 47, no. 9, pp. 1878–1886, 2011.

[34] K. S. Yildirim and A. Kantarci, "External gradient time synchronization in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 633–641, Mar. 2014.

[35] T. Schmid, Z. Charbiwala, Z. Anagnostopoulou, M. B. Srivastava, and P. Dutta, "A case against routing-integrated time synchronization," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2010, pp. 267–280.

[36] X. Liu, K. Li, G. Min, Y. Shen, A. X. Liu, and W. Qu, "Completely pinpointing the missing RFID tags in a time-efficient way," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 87–96, Jan. 2015.

[37] X. Liu, K. Li, A. X. Liu, S. Guo, M. Shahzad, A. L. Wang, and J. Wu, "Multi-category RFID estimation," *IEEE/ACM Trans. Networking*, vol. 25, no. 1, pp. 264–277, Feb. 2017.

[38] X. Liu, K. Li, S. Guo, A. X. Liu, P. Li, K. Wang, and J. Wu, "Top-k queries for categorized RFID systems," *IEEE/ACM Trans. Networking*, vol. 25, no. 5, pp. 2587–2600, Oct. 2017.

[39] I. in 't Veen, Q. Liu, P. Pawełczak, A. N. Parks, and J. R. Smith, "BLISP: Enhancing backscatter radio with active radio for computational RFIDs," in *In Proc. 2016 IEEE International Conference on RFID (RFID)*, May 2016, pp. 1–4.

[40] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 2007.

[41] Texas Instruments. (2015) Msp430fr59xx mixed-signal microcontrollers. [Online]. Available: http://www.ti.com/lit/gpn/MSP430FR5969

[42] B. Ransford. (2015) Llrp library controller. [Online]. Available: https://github.com/ransford/sllurp

[43] "LLRP Ver. 1.1," 2015. [Online]. Available: http://www.gs1.org

[44] "The gnu software radio," 2007. [Online]. Available: https://gnuradio.org

[45] H. Aantjes, A. Y. Majid, P. Pawelczak, A. Parks, and J. R. Smith, "Fast downstream to many (computational) rfids," in *Proc. IEEE INFOCOM 2017 (accepted)*, November 2017.

[46] C. Lenzen, P. Sommer, and R. Wattenhofer, "Optimal Clock Synchronization in Networks," in *Proc. ACM SenSys*, Nov. 4–6 2009.

[47] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists, Third Edition*. Elsevier Academic Press, 2004.

[48] M. Buettner, B. Greenstein, and D. Wetherall, "Dewdrop: An energy-aware runtime for computational rfid," in *Proc. NSDI'11*, 2011.

[49] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," in *Proc. PLDI 2015*, June 13–17 2015.

[50] A. Colin, G. Harvey, B. Lucia, and A. P. Sample, "An energy-interference-free hardware-software debugger for intermittent energy-harvesting systems," in *Proc. ASPLOS'16*, April 02–06 2016.

[51] ——, "An energy-interference-free hardware-software debugger for intermittent energy-harvesting systems," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 577–589, 2016.

[52] T. Instruments. (2016) Floating point math library for msp430. [Online]. Available: http://www.ti.com/tool/mspmathlib

[53] ——. (2014, June) Application note slaa628. [Online]. Available: http://www.ti.com/lit/an/slaa628/slaa628.pdf

[54] "Texas instruments code composer studio - integrated development environment," http://www.ti.com/tool/ccstudio.

[55] "Saleae logic analyzer," http://www.saleae.com.

[56] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient Backscatter: Wireless Communication out of Thin Air," in *Proc. ACM SIGCOMM*, Aug. 12–16 2013.

Kasım Sinan Yıldırım received his Ph.D. degree in Computer Engineering from Ege University, Izmir, Turkey, in 2012. He was a visiting scholar at the Department of Information Engineering, University of Padova in 2013. Between 2015 and 2017, he was a postdoctoral researcher at Embedded Software Group, Delft University of Technology, Delft, The Netherlands. Currently, he is an assistant professor at the Department of Computer Engineering, Ege University and also a research affiliate with the Embedded Software Group at Delft University of Technology. His research interests are embedded systems and software, distributed algorithms, wireless sensor networks, energy harvesting systems and computational RFIDs.

Henko Aantjes Henko Aantjes received his MSc degree in 2016 from Delft University of Technology, Delft, The Netherlands. Currently he is a security evaluator at Brightsight BV, Delft, The Netherlands. His research interests include computational RFIDs, secure computation, and contactless payments.

Przemysław Pawełczak (S'03-M'10) received the MSc degree from Wrocław University of Technology, Poland, in 2004 and the PhD degree from Delft University of Technology, The Netherlands in 2009. Between 2009 and 2011 he was a postdoctoral researcher at the Cognitive Reconfigurable Embedded Systems Lab, University of California, Los Angeles. Between 2012 and 2013 he was a research staff member at Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Berlin, Germany. Since 2013 he is a (tenured) assistant professor at the Embedded Software Group of Delft University of Technology. His research interests include wireless networking, RFID systems and intermittently-powered embedded devices.

Amjad Yousef Majid obtained his MSc degree in 2015 from Delft University of Technology, Delft, The Netherlands. Currently, he is a PhD candidate at the Embedded Software Group at the same university. His research focuses on intermittent software architecture, (computational) RFIDs, and backscatter communication.