On the Synchronization of Intermittently Powered Wireless Embedded Systems

Kasım Sinan Yıldırım Henko Aantjes Amjad Yousef Majid Przemysław Pawełczak
Embedded Software Group, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
{k.s.yildirim, a.y.majid, p.pawelczak}@tudelft.nl, h.aantjes@student.tudelft.nl

Abstract

Battery-free computational RFID platforms, such as WISP (Wireless Identification and Sensing Platform), are emerging intermittently powered devices designed for replacing existing battery-powered sensor networks. As their applications become increasingly complex, we anticipate that synchronization (among others) to appear as one of crucial building blocks for collaborative and coordinated actions. With this paper we aim at providing initial observations regarding the synchronization of intermittently powered systems. In particular, we design and implement the first and very initial synchronization protocol for the WISP platform that provides explicit synchronization among individual WISPs that reside inside the communication range of a common RFID reader. Evaluations in our testbed showed that with our mechanism a synchronization error of approximately 1.5 milliseconds can be ensured between the RFID reader and a WISP tag.

Categories and Subject Descriptors C.2.1 [*Network Architecture and Design*]: Wireless communication; C.2.2 [*Network Protocols*]; C.3 [*Special-Purpose And Application-Based Systems*]

Keywords Computational RFIDs, Wireless Identification and Sensing Platform, Synchronization

1. Introduction

Low-power wireless embedded systems consist of tiny, low-cost, battery-operated and spatially separated computers that communicate with eachother through wireless medium by exchanging radio packets. Powering these small-scale embedded systems, e.g. wireless sensor networks (WSNs), is still a crucial problem [5]. Replenishment or recharging their batteries are impractical and inhibit long-term operation. Moreover, batteries increase the size and cost of their hardware. Fortunately, the energy efficiency of these systems has improved considerably such that their power requirements are in the order of a few μW [19]. Furthermore, recent advancements in microelectronics technology enabled harvesting power from radio frequency (RF) sources which is sufficient to power low-power embedded systems in practice [4]. Nowadays, the growth of RF-powered computing paradigm is bringing new research opportunities and challenges [5], leading to a promising class of low-power embedded systems, 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 [19]. CRFIDs are equipped with a backscatter radio 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, which is a fundamental difference from sensor networks as the transceiver circuit is the most energy-hungry component in WSN. In CRFID domain, the bottleneck in terms of power consumption has shifted from communication to computation and sensing [23]. A typical example of CRFID systems is the WISP (Wireless Identification and Sensing Platform) [17]. Commercial RFID readers implementing EPC Gen2 standard [2] are used to provide power to WISP tags and to receive data from them. Apart from low data rate sensing [6], these maintenance-free devices are evolving to support also high data rate, more complex and richer sensing applications such as continuous sensor data-streaming, e.g. capturing and transfer of images using battery-free cameras, i.e. WISPCam [13, 14].

1.1 Motivation and Challenges

As CRFID platforms and applications are developing, these systems are anticipated to demand their own implementation of basic sensor network building blocks. As an example, consider a hypothetical application of multiple WISPCams that are deployed to capture images of an object from different angles simultaneously. Since each WISPCam is spatially distributed and has its own clock hardware whose oscillator generate pulses at slightly different speeds, they require a *synchronization* service to obtain a common time notion for such collaborative and coordinated actions. However, the characteristics of CRFID systems expose fundamentally different challenges than sensor networks to implement these services. The reason is mainly twofold:

- **Challenge I:** CRFID systems should perform computations in an energy efficient manner despite of intermittent RF power that leads to loss of computational state frequently, e.g. when RFID reader moves away from the CRFID tag [16].
- **Challenge II:** Continuously varying voltage supply introduces severe hardware instability, e.g. varying oscillator frequencies in short-term that effects the stability of the clocks, leading to degraded accuracy of computation and sensing.

1.2 Contributions

Considering these facts, our focus is to answer the question of *How to design a building block that synchronizes intermittently powered wireless embedded systems?* To this end, first we investigate the WISP platform and provide initial observations and limitations pertaining to the synchronization of these devices. Even though there are studies focused on the synchronization of multiple RFID readers in the current literature, e.g. [10], we are unaware of any existing study that provides explicit synchronization among individual WISPs that reside inside the communication range of a common RFID reader. Hence, our main contribution is to design and implement the first and very initial *reader-tag* synchronization primitive for the WISP platform. Evaluations in our testbed showed that a maximum synchronization error of approximately 1.5 milliseconds can be ensured between the reader and a WISP tag with this mechanism.

2. Synchronization in Conventional Wireless Sensor Networks

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

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, voltage level 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 [9]. Moreover, *quantization errors* occur with low-frequency built-in clocks, that prevents precise timing measurements.

Wireless Communication: Due to their different clock frequencies, sensor nodes exchange their clock information periodically to synchronize their built-in clocks by computing a *software clock* that represents synchronized notion of time. The difference between the reference time and the software clock is the *synchronization error*. In WSNs, the synchronization error is mainly affected by several sources of errors. Delays introduced during the wireless communication between participating nodes is the major error source. The *transmission delay*, defined as the time that passes between the start of the broadcast attempt and the receipt by the receiver node, is composed of deterministic and non-deterministic components [12]. Assigning timestamps at the MAC layer is a common method in WSNs to get rid of the deterministic delay components and to improve synchronization accuracy. This obligates radios like Chipcon CC2420 which allows assignment of time information to a radio packet just before transmission and reception.

Computation Methods: Due to the multi-hop nature of WSNs, 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 by employing *least-squares regression* [9, 12]. There are also fully-distributed approaches in which sensor nodes interact only with their direct neighbours in a peer-to-peer fashion and employ methods based on *distributed consensus* [18, 20]. Since the transceiver is the most power-hungry circuit, the objective of synchronization in WSNs is generally to reduce re-synchronization frequency to decrease communication overhead and save power.

3. Fundamental Challenges of Synchronizing IPDs: The CRFID Case

After presenting a brief summary of the synchronization in WSNs, we now delve into to the synchronization characteristics of IPDs. To this end, we will consider WISP [17], the de facto CRFID platform. The main aspects pertaining to synchronization in WISP can be stated as follows:

- **Single-hop reader-tag architecture:** WISPs 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 WISPs are unable to communicate with their neighboring nodes directly, *tag-tag synchronization* is more challenging.
- **Continuously varying voltage level:** In WSNs, the battery level decreases gradually that allows stable voltage levels in short-term. On the contrary, fluctuating input voltage prevents short-term stability of the clock hardware and introduces significant drift. Hence, the prominent factor affecting the frequency of the crystal oscillator is the varying voltage level rather than the temperature for WISPs.
- **Frequent loss of synchronization state:** On the contrary to sensor nodes, WISP tags frequently "die" due to power loss and they need to save synchronization state, e.g. data regarding to software clock, into the non-volatile memory to recover when they find sufficient energy. However, saving computational state to non-volatile memory is also an energy consuming task [7].
- **Computation and memory overhead sensitivity:** Classical motto of WSNs, "computation instead of communication whenever possible" [8, p. 44], is no longer valid for WISP platform since backscatter communication comes almost for free [23]. We require lightweight methods in terms of computation and memory for the synchronization due to the intermittent power limitations. Since methods like least-squares regression is computationally heavy and require considerable amount of memory [22], so they should be avoided.
- **Limitations of EPC Gen 2 standard:** WISP firmware implements the EPC Gen 2 standard [2] that 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 justified by our measurements presented in the following sections.

4. Reader-Tag Synchronization for WISP Tags

In this section, we provide initial observations, design and implementation of two reader-tag synchronization approaches and their evaluation in our testbed. We first provide a sender-receiver based synchronization design and present its limitations. Then, we introduce an event-based synchronization mechanism and provide its advantages over the former approach.

4.1 Hardware Related Issues and Experimental Testbed

Before delving into these issues, we first present a brief information about the clock hardware of WISP platform, our testbed setup and implementation details.

4.1.1 WISP Clock Hardware

WISP 5.0 platform comes with MSP430FR5969 [21] microcontroller with FRAM non-volatile memory for ultra low energy data storage and retrieval. The MSP430 clock system supports a 32 kHz external crystal oscillator, an internal very-low-power low-frequency oscillator, an integrated internal digitally controlled oscillator (DCO). The clock system includes auxiliary clock (ACLK) signal that can be sourced from the external 32 kHz oscillator. The MSP430 microcontroller has five built-in 16-bit timers that can be clocked with ACLK. Moreover, MSP430 has one active mode and seven software selectable low-power operation modes. In low-power operation mode LPM3, ACKL is active.

4.1.2 Testbed Setup

Our testbed is composed of an RFID reader, a host computer to control this reader and a single WISP tag placed at a university office. We used 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 approximately 10 cm from the antenna with line-of-sight and performed all of our experiments with people inside the office. For host-reader control operations, we used sllurp [15], a LLRP (Low-Level Reader Protocol) [3] control library written in Python. To program the WISP tag, we used a MSP430 Flash Emulation Tool (FET), in combination with TI Code Composer Studio (CCS), attached to the host. In order to explore and characterize the communication between the RFID reader and tag, we used USRP 210 software defined radio, another Laird antenna placed at 50 cm from the tag and GNU Radio [1] toolkit to sniff the radio packets during backscatter communication. This allowed us to measure the communication delays on the order of microseconds.

4.1.3 Software Implementation

We configured Timer B0 so that its clocked with ACKL and we allowed WISP to transition to LPM3 mode when its idle, thus allowing us to have an 16-bit running timer in low-power mode operation at 32 kHz precision. WISP-side implementations are done using C and MSP430 assembly.[1]

4.2 Notation

The timer running at 32 kHz can be considered as the built-in clock, i.e. *local clock*, of the WISP. This local clock is denoted by C_w and its value at any real time t can be modeled as $C_w(t) = \int_{t_0}^{t} f_w(\tau)d\tau$. Here, t_0 represents the time at which WISP powered on and $f_w(\tau)$ represents the *frequency* of the local clock at time τ. Since the crystal oscillators have bounded drifts, e.g. typically a nominal value f_{nom} is known together with a lower bound $f_{\min}$ and an upper bound $f_{\max}$, we assume that $f_{\min} \leq f_w(\tau) \leq f_{\max}$ holds. We denote the clock of the RFID reader by $C_r(t)$, which is considered as the reference time for the WISP tag. By collecting reference time information, the WISP tag can build a *software clock*, denoted by S_w, whose value at any real time t represents the synchronized notion of time. The objective of synchronization is to minimize the *synchronization error* at any time t, which is formally defined as $\gamma(t) = S_w(t) - C_r(t)$.

4.3 Approach 1: Sender-Receiver Based Synchronization

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 and 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 EPC Gen2 standard and LLRP protocol and found out that LLRP assigns a *FirstSeenTimestamp* in UTC, 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"* [3, p. 87]. From this definition, we assumed that this timestamp is assigned by the reader when it receives the EPC during the handshake operation with the corresponding tag, as shown in Fig. 1 as the timestamp assigned at time t_1. Therefore, *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 reader to send a special "synchronization" command after the handshake so that the tag timestamps the command reception event using its local clock, as shown in Fig. 1 as time t_2. The *transmission delay* in this case is $\Delta t = t_2 - t_1$ and it is desirable to keep Δt as small as possible so that $C_w(t_1) \approxeq C_w(t_2)$.

[1] We note that all scripts used to generate the results in the paper (including parsing, post-processing and measurement results) are available upon request.

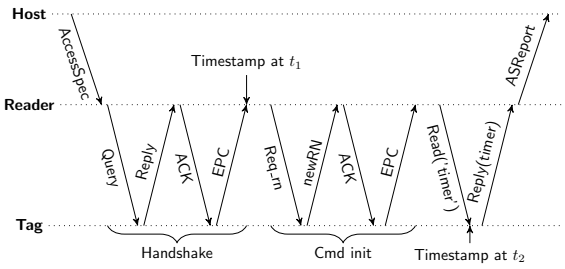

Figure 1: The message exchange among the host computer, the RFID reader and the tag for sender-receiver synchronization. The host machine sends the high level commands to the reader via *AccessSpec* message and receives the results through an *ASReport*. The reader follows the steps defined in EPC Gen2 standard: (i) performs the *Handshake* to initialize the communication with the active tag; (ii) performs the *Command Initialization* 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.

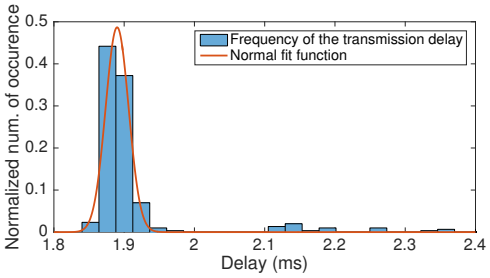

Figure 2: The normalized number of occurence of the tranmssion delays measured by sniffing the communication between REFID reader and the WISP tag. We calculated the mean transmission delay and its standard deviation as 1.89 ms and 0.0164 ms with a 99% confidence interval of [1.8874,1.8925] and [0.0148,0.0184], respectively.

Transmission delay measurements: In order to observe and characterize the transmission delay Δt, we sniffed the communication between the RFID reader and the WISP tag delay during communication scenario in Fig. 1 by obtaining 300 samples. Fig. 2 presents a summary of our measurements. We observed that, the transmission delay is distributed with a mean of 1.89 ms and standard deviation of 0.0164 ms. Even though we observed some outlier values, we think that these are due to the EPC Gen2 implementation of the Impinj reader. After observing the variations of the transmission delay, the next step is to evaluate the limits of sender-receiver synchronization mechanism. To this end, 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 $C_w(t_2) \approxeq C_w(t_1)$ upon receiving this command. The received pairs $(C_w(t_1), C_r(t_1))$ are logged by the host computer.

Software clock computation: By assuming a linear relationship between C_r and C_w, we modeled the software clock of the WISP tag as

$$S_w(C_w(t)) = \alpha + \beta C_w(t),$$

where S_w represents an estimator of reader's clock C_r, α is the offset and β is the relative speed with respect to local clock C_w. To establish such a relationship, we performed least-squares regression in MATLAB, as in [12]. The idea is that since the WISP tag has limited

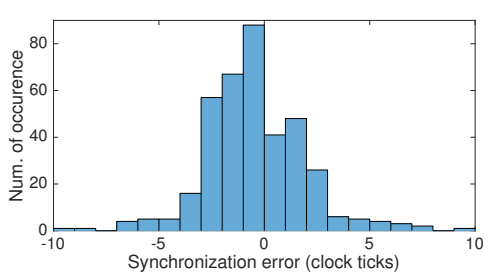

Figure 3: Synchronization error by employing least-squares regression on the collected timestamps. We observed a maximum synchronization error of 10 clock ticks between the RFID reader and the WISP tag, leading to 0.32 ms synchronization accuracy.

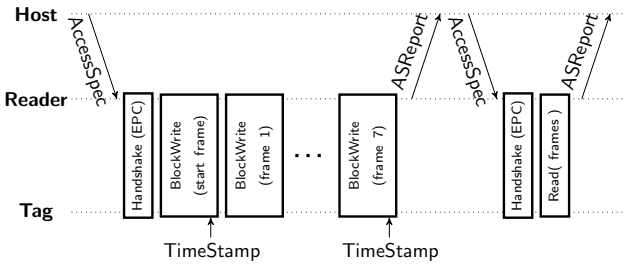

Figure 4: Event-Based synchronization steps: The WISP tag timestamps successive *BlockWrite* events and adjusts its software clock.

memory, computation capability and energy, at each step the most recent N pairs are used to calculate the estimated 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 $\{[C_w(t_k), C_r(t_k)], ..., [(C_w(t_{k+N-1}), C_r(t_{k+N-1})]\}$ are used to calculate the software clock S_w. We calculated the synchronization error as $\gamma(t_{k+N}) = S_w(C_w(t_{k+N})) - C_r(t_{k+N})$ that represents the difference between the predicted reference time and the received reference time. In our implementation, we used $N = 8$ as in [12] and Fig. 3 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 1 order of magnitude than the synchronization performance of the de facto WSN solution [12], which was reported as approximately 10 μs.

Limitations of the approach: In addition to the challenges listed in Section 3, we observed two crucial limitations for WISP platform, that prevents to build up a sender-receiver synchronization building block:

- **Lack of broadcast primitive:** Since RFID reader assigns *FirstSeenTimestamp* for each tag, the synchronization steps in Fig. 1 should be repeated for each WISP tag to obtain synchronization in the communication domain of the reader. Unfortunately, in current EPC Gen2 and LLRP standard, we were unable to find any other mechanism to send a global timestamp that is received by all WISP tags inside the communication range of the RFID reader and used to establish synchronization in one step.
- **Host computer computation:** We are unaware of any command that will allow to send *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 WISP 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 WISP tag and the clock of the reader and send the data that represents this relationship to the WISP tag for synchronization.

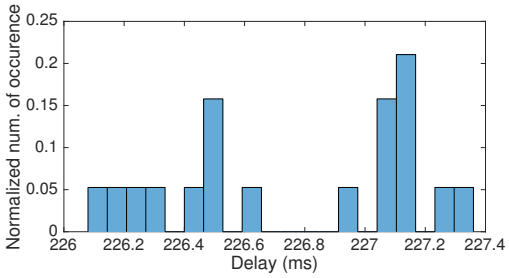

Figure 5: The delay between the first and the last *BlockWrite* event, i.e. *event period*, by considering 20 samples during the communication scenario in Fig. 4. We measured its mean and standard deviation as 226.7667 ms and 0.4097 ms with a 99% confidence interval of [226.4961,227.0372] and [0.2852,0.6945], respectively.

4.4 Approach 2: Event-Based Synchronization

In event-based synchronization mechanisms, 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 occurence time of the future events. In order to synchronize the WISP 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 WISP tag adjusts the rate of its software clock so that it predicts the occurence of the next event precisely.

Observation: We explored the EPC Gen2 standard to see how to generate events at regular intervals and realized that the *BlockWrite* operation allows us this feature. Fig. 4 presents the steps of event-based synchronization. During the command phase, EPC Gen2 allows to perform maximum 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. The real-time length between the first and the last *BlockWrite* operation is the *event period* and its variation is the main error source. Therefore, it is important to explore its charecteristics.

Measurements: To this end, we sniffed the communication between the RFID reader and a WISP tag presented in Fig. 4. We took 20 sample measurements about the *event period*, which is summarized in Fig. 5. According to our measurements, we observed that the *event period* is distributed with a mean of 226.76 ms and standard deviation of 0.41 ms; respectively.

Proposed Solution: For establishment of the software clock, lightweight solutions in terms of computation and memory overhead are crucial due to two important requirements: (i) since voltage level is time-varying, computations should demand little amount of energy, i.e. marginal number of steps, to consume less power (ii) since power is intermittent, the number variables pertaining to the software clock should be marginal so that saving the state of the synchronization to non-volatile memory will demand less time and energy. Considering these facts, we designed a lightweight approach inspired from the PI-controller based solution introduced in [22], which is justified to be an efficient practical solution in WSNs. In order to implement our approach, we model the software clock of the WISP tag in the real-time interval $[t_0, t]$ as

$$S_w(t) = \alpha(C_w(t) - C_w(t_0))$$

Algorithm 1 Integral controller-based synchronization algorithm.

Definitions:
t_f: local time of the first *BlockWrite*
α: rate multiplier
μ_e: mean *event period*
β: constant integral gain

1: **Initialization**
2: $\alpha = 1$ // initialize rate multiplier α
3:
4: □ **Upon receiving the first *BlockWrite***
5: $t_f = C_w(t)$ // store the local time in t_f
6:
7: □ **Upon receiving the last *BlockWrite***
8: $\gamma(t) = \alpha(C_w(t) - t_f) - \mu_e$ // calculate estimation error γ
9: $\alpha = \alpha - \beta\gamma(t)$ // apply integral control to update α

where $f_{\text{nom}}/f_{\max} \leq \alpha \leq f_{\text{nom}}/f_{\min}$ denotes the *rate multiplier* whose value is adjusted to increase or decrease the speed of the software clock with respect to the reference clock. Based on this model, we introduce Algorithm 1 that synchronizes the speed of the software clock to the reference clock speed using an *integral control* strategy. The steps of this algorithm can be explained as follows. Initally, the WISP tag lets its software clock run at the same speed of its local clock by assigning its rate multiplier to 1 (Line 1). Upon receiving the first *BlockWrite* event (Line 4), WISP tag stores its local clock value at the variable t_f (Line 5). After receiving the last *BlockWrite* event (Line 7), first it calculates the difference between the amount of software clock progress and the mean event period μ_e (Line 8). Then, it applies the correction on its *rate multiplier* α by multiplying the error with the integral gain β and subtracting it from α (Line 9). After receiving successive events, α will converge to its desired value eventually, as can be proven by applying similar analytical steps in [22].

Convergence conditions: The selection of the integral gain has significant impact on the performance of the algorithm. In [22], it has been proven that $0 < \beta < 2/Bf$ should be satisfied to achieve synchronization in theory where B denotes the event period in seconds and f denotes the clock frequency in Hz. However, smaller values of β lead to smaller synchronization error but longer convergence time.

Advantages: From the steps of Algorithm 1, it can be observed that this approach employes only very simple arithmetic operations at each step, e.g. Lines 7-9 are composed of only three subtraction 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 α. As a consequence, this matches the requirements of the WISP platform.

4.4.1 Evaluation of Algorithm 1

In order to evaluate the aforementioned synchronization approach, we considered the value of γ in Algorithm 1, which represents the estimation error, as an evaluation metric. We collected the local clock readings of the WISP tag at the first and last *BlockWrite* events and used MATLAB to process the collected data in order to simulate the real behavior of the WISP tag. This gave us flexibility to try different approaches without reprogramming the WISP tag.

Selection of integral gain: Since we measured the mean event period as 226.76 ms on average by experimental evaluation, we set $\mu_e = 7086$ clock ticks since the local clock of WISP tag is operating at 32 kHz and each clock tick occurs at 32 microseconds. First, we explored how β effects the performance of the algorithm. By substituting $B = 0.22676$ sec and $f = 32$ kHz, the convergence condition becomes $0 < \beta < 0.000276$ in our case. In consistency with this bound, we observed that synchronization cannot be established when β is outside this boundary. We present the

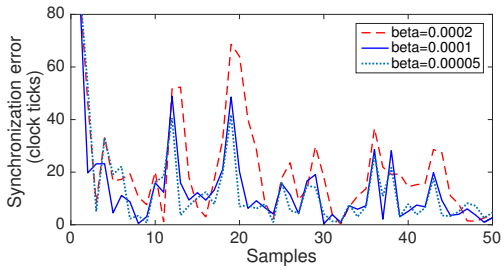

Figure 6: The synchronization error tends to get smaller with smaller integral gains, but after some point deacreasing it has no significant effect due to the precision of 32 kHz clock.

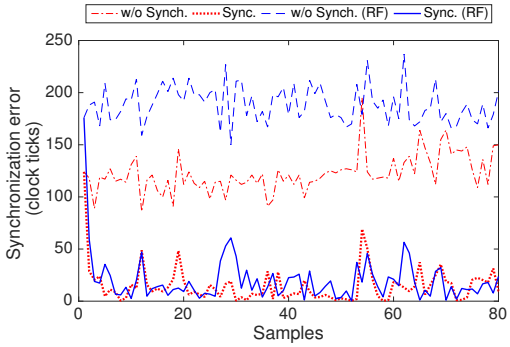

Figure 7: Event-based synchronization performance when WISP tag is powered through a constant voltage source (red lines) and through RF power harvesting (blue lines). Mean synchronization errors without and with Algorithm 1 were almost 127 and 16 clock ticks under stable voltage and they were 175 and 22 clock ticks with RF power harvesting; respectively.

synchronization error with different β values in Fig. 6. Due to the low-precision 32 kHz clock, the synchronization error tends to get smaller with smaller β values but after some point, deacreasing β has no significant effect. Therefore, we chose $\beta = 0.0001$ during next evaluation steps.

Synchronization Performance: Fig. 7 presents synchronization error measurements when the WISP tag is powered by using constant voltage source and powered by only RFID reader through RF waves. Measurements under constant and stable voltage allowed us to observe synchronization under stable clock frequency. We obtained a significant synchronization performance with Algorithm 1, almost a factor of 2 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 error due to the varying transmission delays, that led a peak error appear between the 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 more, as compared to the stable voltage case. However, we also obtained considerable improvements with our efficient approach with more unstable clock frequency, almost a factor of four better synchronization accuracy.

Limitations of Algorithm 1: In conclusion, experimental results indicate that a maximum synchronization error of approximately 1.5 milliseconds can be ensured between the RFID reader and a WISP tag most of the time by employing event-based synchro-

nization mechanism. However, even though event-based approach is relatively lightweight and simple as compared to the sender-receiver synchronization, the WISP tag is unable to obtain an explicit reference clock value. The only variable it tunes is the α 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, Algorithm 1 requires knowledge about μ_e, which can be RFID reader dependent and challenging to measure it. Last, the steps in Fig. 4 presents the communication scenario between one WISP and the reader. To allow other WISP tags sniff this communication and synchronize themselves, broadcast primitive is crucial.

5. Conclusions and Future Research Directions

In this paper, we explored a synchronization scenario between an RFID reader and a WISP tag. 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 to show that with lightweight mechanisms, as of now, a maximum synchronization error of approximately 1.5 milliseconds can be ensured.

We provide the following issues for future studies in this domain:

- **Network-wide synchronization:** Currently, we provided synchronization between an RFID reader and a single WISP tag. However, synchronization among all WISP tags inside a single communication domain and also synchronization of the whole CRFID network composed of several RFID readers and WISP tags is still an issue, due to the single-hop nature of backscatter communication. Since WISP tags can only communicate with the reader, not with their neighboring tags, it is also interesting to study *wisp-wisp* synchronization and explore its feasibility.
- **Design of a broadcast primitive:** As we discussed in Section 4.3, lack of a broadcast primitive for the WISP platform limited our designs and implementations. We think that broadcast primitive is crucial not only for synchronization but also for other protocols and applications in the CRFID domain.
- **Power loss and recovery:** It is crucial to save the synchronization status to non-volatile memory, just before power loss. However, it is not the only system state to be saved and writing to non-volatile memory also consumes energy. Hence, when and how to save the synchronization status is worth to explore.
- **Voltage-frequency relations:** It might be interesting to explore the voltage-clock frequency relationship since WISP 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 eg" problem since reading voltage level also consumes energy.
- **Synchronization in other IPD platforms:** Synchronization in other IPD platforms, such as embedded systems that use ambient backscatter communication [11], is worth to explore as well.

References

[1] The gnu software radio. 2007. URL `https://gnuradio.org`.

[2] 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. URL `http://www.gs1.org/`.

[3] LLRP Ver. 1.1. 2015. URL `http://www.gs1.org`.

[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. of IEEE ISCAS*, Melbourne, Australia, June 1–5 2014.

[5] S. Gollakota, M. Reynolds, J. Smith, and D. Wetherall. The emergence of RF-Powered computing. *Computer*, 47(1):32–39, Jan 2014.

[6] 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*, Baltimore, MD, USA, Nov. 20–22 2008.

[7] 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. 2016. conditionally accepted to IEEE RFID 2016.

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

[9] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal Clock Synchronization in Networks. In *Proc. ACM SenSys*, Berkeley, CA, USA, Nov. 4–6 2009.

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

[11] 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*, Hong Kong, China, Aug. 12–16 2013.

[12] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *Proc. ACM SenSys*, Baltimore, MD, USA, Nov. 3–5 2004.

[13] S. Naderiparizi, A. Parks, Z. Kapetanovic, B. Ransford, and J. Smith. WISPCam: A battery-free RFID camera. In *Proc. IEEE RFID*, San Diego, CA, USA, 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*, Osaka, Japan, Sept. 7–11 2015.

[15] B. Ransford. Llrp library controller, 2015. URL `https://github.com/ransford/sllurp`.

[16] 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*, San Diego, CA, USA, Dec. 2008.

[17] A. Sample, D. Yeager, P. Powledge, A. Mamishev, and J. Smith. Design of an RFID-Based Battery-Free Programmable Sensing Platform. *IEEE Transactions on Instrumentation and Measurement*, 57(11):2608–2615, Nov 2008.

[18] L. Schenato and F. Fiorentin. Average TimeSynch: a consensus-based protocol for time synchronization in wireless sensor networks. *Automatica*, 47(9):1878–1886, 2011.

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

[20] P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *Proc. IPSN*, San Francisco, USA, Apr. 13–16 2009.

[21] Texas Instruments. Msp430fr59xx mixed-signal microcontrollers, 2015. URL `http://www.ti.com/lit/gpn/MSP430FR5969`.

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

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