Integrating Time-Stamped Synchronization into a Periodical MAC Protocol – Problems and Experiences

Clemens Mühlberger

Department of Computer Science
University of Würzburg
97074 Würzburg, Germany
muehlberger@informatik.uni-wuerzburg.de

Abstract. One main characteristic of a Wireless Sensor Network (WSN) is the wireless communication of the participating sensor nodes. Especially for complex and multi-hop topologies, a MAC protocol is essential to provide a (more or less) collision-free access to the shared communication medium. We prefer a periodical and self-organized MAC protocol because of its robustness, adaptivity and scalability. Another important service for various WSN applications is time synchronization. For this reason, several synchronization protocols and techniques have been investigated already.
In this paper we present our real-world experiences when integrating a simple time-stamped synchronization protocol into a self-organized and periodical MAC protocol. This integrative approach requires just marginal additional data, but achieves quite accurate synchronization results. The problems and experiences during our real-world implementation are discussed in detail within this paper.

Keywords: time synchronization, periodical MAC protocol, real-world experience, practical problems, desynchronization

1 Introduction

The wireless communication of sensor nodes is one essential property of *Wireless Sensor Networks* (WSNs). Due to the shared communication medium, this sort of interaction implicates the typical problems of radio wave propagation, like fading or interference. These phenomena appear in form of packet collision and data loss at single-hop topologies, and due to the *hidden terminal problem* (first described by Tobagi and Kleinrock in [24]) even more noticeable at multi-hop topologies. The re-transmission of lost packets consumes additional time and energy, and therefore is undesired – even though it would be possible. That's why the access to the communication medium is often managed by so called *Medium Access Control* (MAC) protocols. For our research, we are using a biologically inspired and self-organized MAC protocol, which provides a collision-free communication even within multi-hop topologies.

On the other hand, the increasing performance and computational power of recent sensor nodes allows a much wider scope of useful application scenarios for WSNs. For example, the broadening to *Wireless Sensor and Actor Networks* (WSANs) will be essential for the upcoming *Cooperating Objects* [11], but comes along with new challenges due to the unique characteristics of WSANs, like coordination and real-time requirement (cf. [1], [2]). Hence, coordination as a special case of interaction strongly relies on precise time synchronization to achieve a common task successfully in an autonomous manner. Further needs of time synchronization in WSN are emphasized for example in [19], [9], and [20].

During our research on self-organized MAC protocols, we found an easy way to integrate a continuous time-stamped synchronization into a certain MAC protocol. So, in this paper we analyze the fusion of time-stamped synchronization and an appreciative class of MAC protocols, which will lead to a reduction of the communication overhead without a specific master node, but offers a robust and accurate time synchronization. We thus present practical problems during the implementation of our integrative approach and discuss further observations which we made at our real-world testbeds.

This paper is organized as follows. First, we introduce the underlying MAC protocol and address its qualification for time synchronization in Sect. 2. Next, we give a short survey of existing time synchronization protocols for WSNs in Sect. 3. The desired characteristics for our time synchronization protocol are discussed in Sect. 4 together with the characteristics of the available hard- and software. Section 5 presents the practical problems and further refinements during the integration of our simple time synchronization algorithm into the self-organized MAC protocol. In Sect. 6 we analyze the capability of our integrative approach and describe some observations made at our real-world testbeds, before a short conclusion and an outlook to topics of our future research in Sect. 7 closes this paper.

2 The underlying MAC Protocol

This section introduces the biologically inspired primitive of desynchronization and its implementation as MAC protocol for WSNs in special. The qualification of this self-organized communication protocol to involve a simple time synchronization service also is presented in detail.

2.1 General Assumptions

At first, we specify the following general assumptions for our further studies. The (idealized) sensor network is composed of a set N of sensor nodes, where each node $i \in N$ owns a unique identifier (ID) as well as a finite buffer for storing (incoming) packets. But each node has just one transceiver in half-duplex mode, i.e. not a single node can transmit and receive packets simultaneously. We further assume that the communication range of each node equals its interference range. Finally, the network is build upon symmetrical links, i.e. communication between two nodes always works bidirectionally.

2.2 Desynchronization – The Basics

The MAC protocol used within this paper is based on the biologically inspired primitive of desynchronization [13]. As logical opposite of synchronization, desynchronization in general means that each device tries to perform its (periodic) tasks as far away as possible from all other affected devices. Within the scope of WSNs, desynchronization describes the temporally equidistant transmission of radio packets. That means, each node oscillates at an identical frequency $f = \frac{1}{T}$, i.e. each node tries to transmit a so called *firing packet* after every period T.

Thus, when a node finishes its period, it broadcasts a firing packet and immediately resets its phase, i.e. it will broadcast its next firing packet after exactly another period T. Each one-hop neighbor of the currently transmitting node $j \in N$ receives this firing packet (if there was no collision), and logs the sender's ID together with its local time of reception to calculate its individual phase shift ϕ_j towards the sender j relative to the current period of the receiving node. Each node i is able to determine a more appropriate firing phase (according to an equidistant distribution), just based on the individual knowledge of the phases of its so called *phase neighbors*:

- the predecessor $p(i) \neq i$ broadcasts its firing packet (from i's point of view) just before node i and thus is called *previous phase neighbor*,
- the successor $s(i) \neq i$ broadcasts its firing packet (from i's point of view) just after node i and thus is called *successive phase neighbor*.[1]

Note that node i determines its phase neighbors itself according to its just limited and local view of the overall network. That means from a global perspective (which in general is not available for every node of the network), in multi-hop topologies further nodes may broadcast their firings chronological between node i and its phase neighbors due to the spatial distribution of the network. It is also noteworthy, that both phase neighbors $p(i)$ and $s(i)$ can be one-hop or two-hop neighbors of node i. Unfortunately, the information about the phase of the phase neighbors might be stale, even at single-hop topologies. For example, $p(i)$ could have determined a more appropriate time of firing just after the firing of node i, and therefore node $p(i)$ could have changed its phase already. However, according to [16], this stale information does not affect the functionality of the desynchronization algorithm in general, but slows down the convergence rate.

On the basis of its (absolute) last time of firing $t_{TX}(i)$, node i can now calculate the midpoint of its phase neighbors, and finally estimates its new time of firing $t'_{TX}(i)$ as

$$t'_{TX}(i) = (1-\alpha) \cdot \big(t_{TX}(i) + T\big) + \alpha \cdot \frac{\big(\phi_{s(i)} + \phi_{p(i)} - T\big)}{2}. \tag{1}$$

Here, the *jump size parameter* $\alpha \in [0.0, 1.0]$ regulates, how fast the node moves toward the assumed midpoint of i's phase neighbors. Using absolute time values

[1] Besides, within a connected topology of size $|N| = 2$ both phase neighbors are the very same node $p(i) = s(i)$.

for the time of firing avoids modular arithmetic for normalization. Convergence to the stable state of *desynchrony* (cf. [3]) is achieved, if each node has the same distance to its phase neighbors and thus the transmission times do not change anymore - unless the system changes.

2.3 Desynchronization – As MAC Protocol

The first implementation of the primitive of desynchronization as MAC protocol for WSNs was the DESYNC protocol [3], [16]. This collision-free MAC protocol does not require any specific time synchronization due to the periodical transmission of firing packets, but just works well for single-hop topologies. However, the so called *hidden terminal problem* inheres in multi-hop topologies, which complicates a collision-free communication. To solve the hidden terminal problem within the desynchronization approach each node needs additional knowledge about its two-hop neighborhood. To the best of our knowledge, just a few approaches for multi-hop topologies using desynchronization are already available and will be briefly described below.

The M-Desync Algorithm The M-DESYNC algorithm [8] for (single-hop and) acyclic multi-hop topologies is based on the *local max degree* of each node, i.e. the maximum degree among the node i and its one-hop neighbors $N_1(i)$. Here, the *degree* of a node equals the cardinality of its one-hop neighborhood $|N_1(i)|$. This algorithm requires an initial phase, at which each node exchanges its degree with all its one-hop neighbors to determine its local max degree. This phase may take quite long, because the algorithm uses just a random back-off protocol without further optimization. After this preliminary phase, every node requires local max degree plus an additional time slots for a collision-free communication within its interference range. At the next step, each node just has to occupy its individual time slot. For this slot selection, a modulo pre-coloring as well as a priority-based strategy are suggested instead of just a random competition. Using the local max degree, the minimum number of required time slots per period for each node was proven. However, the M-DESYNC approach is not very flexible to topology changes due to the lengthy exchange phase, but even not applicable for cyclic topologies. This is the reason, why this algorithm does not meet our requirements of a self-organized MAC protocol for universal use.

The extended-Desync Algorithm At the EXTENDED-DESYNC algorithm [14] each node broadcasts its (currently known) one-hop neighbors in combination with their relative phase shifts, always corresponding to the point of view of the current sender. With it, each node gets to know its two-hop neighborhood $N_2(i)$ in addition. To solve the hidden terminal problem with respect to the primitive of desynchronization, predecessor and successor may now be two-hop neighbors, more formal $p(i), s(i) \in N_1(i) \cup N_2(i)$. As a result, the relative phase shifts may become stale, because phase changes of two-hop neighbors emerge not until two periods. But this delayed information becomes more accurate and reliable

with each subsequent period and thus just slows down convergence rate a little. Here, no initial exchange phase is required. Instead, a new joining node just has to listen for a few periods to make itself familiar with its local topological conditions. Afterwards, it can interact immediately with its well-known one-hop neighbors and thus be integrated into the network easily. Hence, the EXTENDED-DESYNC approach is very flexible, reacts quite fast on topology changes, and also scales well with network size.

2.4 Qualification

Both MAC protocols for multi-hop topologies introduced in Sect. 2.3 rely on neither a fixed offline scheduling of the individual time slots, nor a central arbiter for slot allocation. They also do not require an explicit time synchronization protocol (cf. Sect. 3) for a collision-free communication, the coordination of the nodes is just based on the periodical firing packets.

We prefer the EXTENDED-DESYNC algorithm for our further research, although its packet overhead may be large, especially in dense networks and at nodes with a high degree. Indeed, it is not as topology-restricted as M-DESYNC, but furthermore does not require any pre-desynchronization process to exchange degree information among neighboring nodes. Due to the self-organizing character, the EXTENDED-DESYNC protocol scales well with network size, is flexible and reactive to topology changes, and thus deals well with moving, leaving or joining nodes. Furthermore, collisions due to the hidden terminal problem are now minimized, because each node autonomously estimates its next point in time for transmission – considering its complete one-hop and even two-hop neighborhood.

The EXTENDED-DESYNC protocol already offers some useful instruments for an explicit time synchronization service, which in turn should be easily be integrated. First of all, the firing packets at this MAC protocol are transmitted periodically. This enables a continuous re-synchronization of the sensor nodes – every firing packet implies a permanent correction, refinement, and update of time data of one-hop as well as two-hop neighbors. Next, this MAC protocol relies on broadcasts, namely firing packets. This is the reason why each firing packet could be the inspiration for a reference broadcast (cf. Sect. 3.1). Finally, each receiver already has to time-stamp every incoming firing packet in conjunction with (1). Therefore, some temporal information about one-hop neighbors is available in any case.

3 Related Work

Before we discuss our experiences and problems of our integration of time synchronization into the EXTENDED-DESYNC protocol in detail, this section takes a short look to similar but yet available time synchronization protocols for WSNs, namely the *Reference Broadcast Synchronization*, the *Timing-Sync Protocol for Sensor Networks*, the *Flooding Time-Synchronization Protocol*, and the *Time-Stamp Synchronization*.

3.1 Reference Broadcast Synchronization (RBS)

The Reference Broadcast Synchronization (RBS) [4] uses a receiver-to-receiver synchronization and tries to shorten the critical path when transmitting a packet by removing the send time as well as the access time from the message's latency. Therefore, a reference node broadcasts a simple synchronization packet, which at all must not contain any timing information. Each receiver of that synchronization packet records its local time of reception and exchanges this information with its neighboring nodes. Due to the different local time-stamps of each receiver of the very same synchronization broadcast, the nodes can calculate a time offset to all of its neighbors and thus obtain a local timescale for each neighbor. Mostly, quartz crystals are installed as clock-pulse generator for sensor nodes, but these clocks do not have a perfect rate. And because RBS basically does not depend on periodical broadcasts, it tries to compensate this so called clock drift by a simple linear regression.

Here, no special packet type for synchronization is required, basically every broadcast can be used for RBS. Hence, lots of existing communication protocols can be extended for synchronization using RBS. By contrast, the communication effort is quite high due to the numerous exchange packets between the receivers of such a broadcast. This exchange takes a certain amount of time, that's why it also takes time until the whole network will be synchronized.

3.2 Timing-Sync Protocol for Sensor Networks (TPSN)

The Timing-Sync Protocol for Sensor Networks (TPSN) [5] uses a sender-to-receiver synchronization. Here, the synchronization process consists of the two phases *Level Discovery* and *Synchronization*. The Level Discovery Phase generates a hierarchical topology which assigns each node to a specific level. The single node at level 0 is called *root* node. The Synchronization Phase is initiated by the root node. According to the hierarchical structure, each node at level i is pairwise synchronized to one node from level $i-1$, until the whole (connected) network will be synchronized to the single root.

In contrast to RBS (see Sect. 3.1), there is no reduction of the critical path under TPSN, but since the time-stamping of messages occurs at MAC layer, quite accurate synchronization results can be achieved. The round-trip method is used herein to adapt the local time-stamps according to the estimated transmission delays. Depending on the network size, the Level Discovery Phase as well as the Synchronization Phase will take quite long, and so will the overall synchronization of all nodes to the clock of the root node.

3.3 Flooding Time-Synchronization Protocol (FTSP)

Like the TPSN protocol from Sect. 3.2, the Flooding Time-Synchronization Protocol (FTSP) [10] uses a sender-to-receiver synchronization and also relies on a root node to synchronize with the remaining nodes. Here, the root node initiates the synchronization procedure by broadcasting a so called *synchronization*

packet, which contains the (unique) root ID, a sequence number and the transmission time-stamp, which is recorded at MAC layer again. Each receiver rejects those packets, whose root ID is greater than a stored root ID of some packet received before, or whose sequence number is less or equal to the stored sequence number of some packet received before. In turn, each receiver broadcasts a synchronization packet itself. This way, the network is flooded until every node of the network is synchronized towards the clock of the root node. Similar to RBS (see Sect. 3.1), FTSP also uses a simple linear regression algorithm for drift compensation.

If a node does not receive any further synchronization packets, because it is disconnected or the root node fails, this node declares itself as new root after a specific *root timeout*, and re-initiates the synchronization process by using its own identifier as new root ID. If this recent root node receives a synchronization packet containing a lower root ID, it quits claiming to be root, but adopts that node with this lower identifier as exclusive root and tries to synchronize to that one. Due to the root timeout, the time of the reelection process of a new root node is limited but yet depends on the radius of the network measured by the root node. The linear regression requires also a couple of synchronization packets, thus a node is not synchronized immediately after startup

3.4 Time-Stamp Synchronization (TSS)

The Time-Stamp Synchronization (TSS) [18] (an improved version of this protocol is presented in [12]) is designed specifically for ad-hoc networks and thus does not provide a global clock synchronization. Each node keeps its local timescale, time-stamps are just piggybacked to normal messages. Therefore, every time-stamp transmitted from a sender to a receiver will be transformed into the receiver's local timescale by the receiving node itself. For this transformation accurate knowledge about the total amount of time required to transmit the time-stamp from sender to receiver is needed. This knowledge is gained by round-trip measurements of e.g. the message and its acknowledge of receipt. The time measurements of a previous message exchange is used to determine an interval with upper and lower bounds for the delay of the current message. This allows the assignment of a time interval to aged time-stamps, even over several hops.

Indeed, the measurement of round-trip time of a message can raise the synchronization error. However, the synchronization error will increase as well with the number of hops. This might be a problem in sparse networks which typically exhibit a high number of hops due to their spatial distribution. Furthermore, the synchronization is short-dated. On the one hand, this will be in line with applications of high mobility, but on the other hand may limit the applicability of this protocol. Altogether, the TSS protocol will be most similar to our simple time-stamped synchronization approach. In contrast, our approach does not rely on round-trip measurements due to the already periodical firing packets.

4 Characteristics

This section first outlines the desired characteristics of our integrative time synchronization approach, which is a blending of some features from the time synchronization protocols named above (cf. Sect. 3). Next, it briefly describes the characteristics of the used real-world system, because some statements in Sect. 5 are quite system specific.

4.1 Desired Characteristics

First of all, our synchronization protocol shall work well at ad-hoc networks, that means, it must be scalable and adaptive. This is already fulfilled due to the underlying MAC protocol. Thus, this self-organized approach requires neither a special topology control, like the Level Discovery Phase of the TPSN (cf. Sect. 3.2), nor a single master node, like at FTSP (cf. Sect. 3.3). Therefore, we avoid the single point of failure in form of a global clock generator, which would cost additional energy and time in case of network re-synchronization.

We also want to offer just a precise but relative timescale like RBS (cf. Sect. 3.1) and TSS (cf. Sect. 3.4). However, it should be mentioned that as soon as the nodes are synchronized with relative timescales, a global time can be achieved afterwards easily. For example, if a certain node's local time is propagated as global time, or additional hardware gains access to another timescale (like GPS [17]), which in turn is stated as global time then.

Our time synchronization mechanism is integrated into a periodical MAC protocol (see Sect. 2), thus we can easily supply a periodical synchronization procedure. This not only makes clock drifts manageable up to a certain degree (cf. [4]), but also allows a fast reaction on topology changes like moving, leaving or joining nodes due to this continuous information update.

Furthermore, environmental factors are on the one hand non-deterministic and non-linear (e.g. see [7]), but they have some influence over the installed quartz crystals of the sensor nodes. This makes it hard to find a straight line reflecting the real clock drift very well for a longer period of time. For example, the influence of thermal factors are remarkable compared to the already existing frequency tolerance of such crystals: the Vishay XT49S Crystal [26] as typical representative of such low cost crystals often deployed on sensor nodes, has a frequency tolerance of ± 30 ppm at 25 °C, a frequency tolerance over the operating temperature range (from -10 °C to $+70$ °C) of ± 50 ppm, and an aging effect in the first year of just ± 5 ppm. On the other hand, assuming a short-term frequency stability implicates an ongoing and steady execution of the drift compensation. Here, the periodical transmission scheme of the underlying MAC protocol may assist.

The calculation of the simple linear regression algorithm takes computing time. Additionally, memory has to be reserved for data points, which must be collected first. In consequence, time synchronization is not available at once, because the gathering of these data takes time, and moreover consumes memory for saving. For example, FTSP maintains a table of eight items and requires

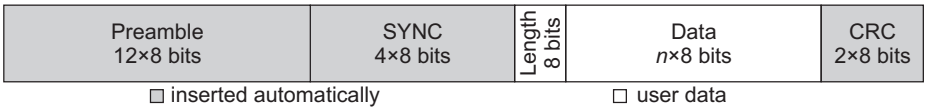

Fig. 1. The simplified packet format used here.

at least three of them for a first run of the simple linear regression algorithm (cf. [10]). This is the reason, why we do not install an algorithmic drift compensation for now. But this also implies, that the accuracy of our time-stamped synchronization approach strongly depends on the period T of the underlying MAC protocol and also on the available temporal resolution, in particular the frequency of the installed quartz crystal as well as the utilization of this quartz by the used operating system.

Finally, in opposition to FTSP (cf. Sect. 3.3) or TPSN (cf. Sect. 3.2), we dislike special synchronization packets. Instead, the additional data required for our simple synchronization approach is kept small, because it will be integrated into the regular control packet structure similar to TSS (cf. Sect. 3.4). This mainly minimizes the communication overhead for the synchronization process itself.

4.2 Present Characteristics

All sensor nodes used for our integration as well as the real-world testbenches are mounted with a Texas Instruments MSP430 microcontroller [21], [23] and a CC1100 radio transceiver [22]. The clock of the microcontroller runs at 8 MHz, and the radio unit uses the *minimum shift keying* (MSK) modulation at the transmission frequency of 433 MHz and a transfer rate of 250 kbps. All radio packets have the following same format (cf. Figure 1): first are 12×8 preamble bits, succeeded by a 32 bit sync word. Thereon follows the 8 bit length field, which defines the size of the successive data field. A 16 bit CRC field marks the end of each radio packet.

Furthermore, the radio transceiver asserts a certain general digital output pin when the sync word has been sent or received (cf. Figure 2). At the used sensor nodes, this digital output pin of the CC1100 radio controller is connected to a general digital input pin with interrupt capability at the MSP430 microcontroller. With it, an interrupt marks the end of the sync word transmission, and reception respectively.

The quasi-standard operating system for WSN is the non-preemptive TinyOS [6], [25], but the underlying MAC protocol already states some strict temporal conditions to be fulfilled, e.g. the next firing packet has to be transmitted exactly one period T after the current one. Therefore, we favor a preemptive operating system with real-time capabilities. This operating system also provides an accurate timeline with a temporal resolution of 1 µs.

Additionally, the operating system used herein further offers an interrupt time-stamping at a very deep level: If an interrupt request (IRQ) from a hard-

ware component (e.g. the radio chip) occurs, the interrupt service routine (ISR) responsible for the interrupt level of that IRQ first logs the current local time. Afterwards, it calls the proper interrupt handler. So far, no (non-deterministic) time was spent for context switches, further calls, or maybe complex branches. That means, if an interrupt occurs, this operating system always records the local time for that interrupt as soon as possible, i.e. the logged time-stamp is very close to the real time of the interrupt's occurrence. Also, no specific interrupt handler must take care itself about time-stamping: if required, it just has to read out the stored timing value.

5 Integration

This section specifies the observations and problems which arose during the integration procedure. So far, the underlying MAC protocol together with the preemptive operating system already provide some features useful for a time synchronization service, e.g. the periodical update of temporal data of neighboring nodes, reference broadcasts in form of firing packets, and according to (1) the (automatic) time-stamping of received firing packets to determine the next time of firing. Thus, to integrate a service for time synchronization into the underlying MAC protocol, just little modifications have to be done. However, there are still some problems to solve, and there is also room for improvements.

5.1 Send Command vs. Transmission Start

First, in order that the receiver of a firing packet is able to synchronize itself (relatively) towards the sender's clock, the current clock of the sender (e.g. its local time of transmission) has to be integrated into the firing packet. One easy but non-deterministic way is the integration of the time of the send command into the firing packet. However, due to e.g. occuring interrupts, there is a non-deterministic delay between the transmission command and the start of the transmission in general. When using a non-preemptive operating system this non-deterministic delay between send command and actual transmission of a radio packet may be on the order of several milliseconds (cf. [6], [10]). But even our preemptive operating system causes a shorter (but still non-deterministic) delay in the order of a few milliseconds: the oscilloscope screenshot in Figure 2 denotes a delay of 1.55 ms. This smaller delay here is specific to the configuration of the radio unit, because it is based mainly on calibrating and oscillating constraints of the radio unit, especially when switching from an idle or listening state to the transmission mode (which is shown by Figure 2). Anyway, accurate knowledge about this delay is important for a proper operation of the EXTENDED-DESYNC protocol to schedule the time of the next firing packet adequately. Otherwise each node tries to re-adjust its next time of firing according to its retarded phase neighbors.

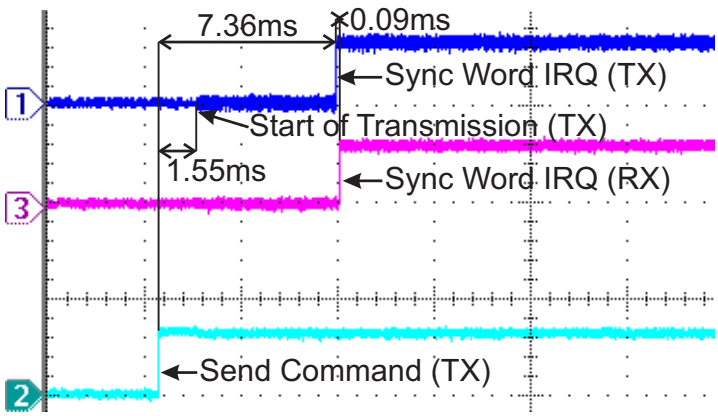

Fig. 2. The timing characteristics during the transmission of a firing packet (as excerpt of an oscilloscope screenshot). Channel 1 (blue) indicates the end of the sync word at the sender (TX), channel 2 (cyan) displays the active time of the sender task, and channel 3 (magenta) indicates the end of the sync word at the receiver (RX).

5.2 More Appropriate Time-Stamps

Due to the used hardware, an interrupt marks the end of the sync word transmission and reception (cf. Sect. 4.2). Moreover, this interrupt is time-stamped automatically by the operating system, and thus this time-stamp would be a more appropriate time value for the firing packet. Fortunately, this time-stamp of the sync word interrupt still can be attached to the data field of a packet, even while the radio unit already is transmitting the first bytes of that packet. In this way, we do not get the real start time of transmission again, but the difference between the start of transmission and the end of the sync word is fixed. By broadcasting both the time-stamp of the desired transmission start and the time-stamp of the sync word interrupt, each receiver gets to know the delay from Sect. 5.1 accurately. According to Figure 2, the transmission of the 16×8 bit of preamble and sync word takes 5.81 ms at the used radio configuration.

5.3 Prolonged Reception

Because radio waves travel in normal air with about the speed of light, the sync word interrupt should occur almost simultaneously at both sender and receiver. Indeed, due to filtering and circuitry delays at the receiver, there is a quite constant, but notable time lag of 90 µs (cf. Figure 2, see also [15]). This lag far exceeds the propagation time[2], and thus is independent of the distance between the nodes, but strongly depends on the configuration of the radio unit, e.g. transfer rate and transmission frequency. For a high precise time synchronization, this time lag has to be taken into account for further timescale calculations again.

[2] A radio wave requires in normal air about 1 µs for a distance of 300 m.

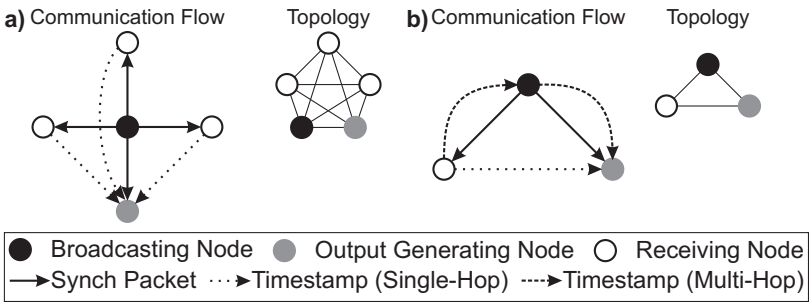

Fig. 3. The experimental setup of our single-hop 6.1, and multi-hop testbed 6.2 respectively.

6 Evaluation

This section discusses the results obtained by several real-world testbeds. A description of the used hardware (and software) within our testbeds can be found in Sect. 4.2. We analyzed single-hop as well as multi-hop topologies to demonstrate the applicability and practicability of our integrated synchronization approach. We could further observe several phenomena which we will also explain within this section. For all test cases, the period of the underlying EXTENDED-DESYNC protocol was set to $T = 1000$ ms.

6.1 Single-hop Topology

First, we tried to analyze our integrated synchronization protocol at a single-hop topology (cf. Fig. 3a). Therefore, we dropped five sensor nodes so that each node could communicate in a direct way with each other. The firing packets of one certain node were chosen for analysis purposes. These packets were received by the remaining four nodes at about the same time due to the small area covered by this testbed. Each receiver recorded the time-stamp of the receiving sync word interrupt at its local timescale and exchanged this time-stamp with each other – corresponding to the self-organized transmission schedule of the underlying MAC protocol. One of the four receiving nodes was taken to output the difference of the correlated time-stamps of the certain firing packets. Even after thousands of synchronization packets, all time-stamps had a relative deviation within ± 4 µs.

6.2 Multi-hop Topology

Next, we analyzed the performance of our integration approach at a multi-hop topology (cf. Fig. 3b). We used just three nodes within a yet single-hop topology to get a directly (single-hop) measured time-stamp in comparison to the modified (multi-hop) time-stamp. Again, the firing packets of one node were chosen for analysis purposes. Each receiver again recorded the receiving sync word interrupt at its local timescale. One of these receiving nodes was taken to

output the difference of the correlated time-stamps again. But now, to inspect the performance of our integrated approach within a multi-hop topology, the time-stamp of one receiver was additionally transmitted via the broadcasting node as intermediate station to the output generating node. That means, the output generating node knew both the single-hop transmitted time-stamp of the other receiver, and the multi-hop forwarded version of the very same time-stamp. The synchronization error when passing the intermediate station, i.e. all multi-hop time-stamps, exceed that one without intermediate station about twice. For this testbed, the clock drifts appeared to sum up. That means, the clock drift at links toward the intermediate station, and the clock drift at links from the intermediate station must have the same sign, i.e. they are both positive, or they are both negative. Anyway, a slightly worse time synchronization over several hops was observable.

6.3 Phase to Phase

During our tests before, we could observe another phenomenon. Because we are using the periodical MAC protocol EXTENDED-DESYNC, the relative phase between two nodes can be essential for the relative synchronization error, if their clocks are drifting apart. That means, the relative temporal ordering of the time slots directly influences the detectability of the clock drift and thus the synchronization error. Since the frequency tolerance of crystals are declared as *parts per million ticks*, the absolute clock drift grows with the number of the ticks. That is why a too long period T let increase the synchronization error observably: the temporal distance between some neighboring nodes within a single period grows, hence the clock drift becomes well noticeable. For this reason, algorithmic clock drift compensation as implemented in RBS (cf. Sect. 3.1) and FTSP (cf. Sect. 3.3) will be necessary for an accurate time synchronization.

6.4 Accuracy at Startup

Some of the time synchronization protocols introduced in Sect. 3 require an initial phase or have to collect synchronization packets first. Thus, we tried to analyze, how long it takes within a single-hop topology, until a newly joining node is synchronized relatively to its neighbors. At first, we had a single-hop topology consisting of three nodes again. At period 113 since start-up of the node which was chosen to generate the output, we powered up a fourth sensor node, which joined the already synchronized network after two periods of observation. The differences of the correlated relative time-stamps are displayed in Figure 4: at period 115 the new node is registered by the output generating node for the first time. The operating system predefines a temporal resolution of 1 µs here, this is the reason, why all synchronization errors in Figure 4 appear as discrete values. The synchronization error also lies within an interval of $[-1\ \mu s, 2\ \mu s]$. Moreover, the joining node can synchronize itself "silently" during its observing periods. Hence, the startup of a node does not have any significant effect neither on the synchronization error nor on the time until the joining node is synchronized.

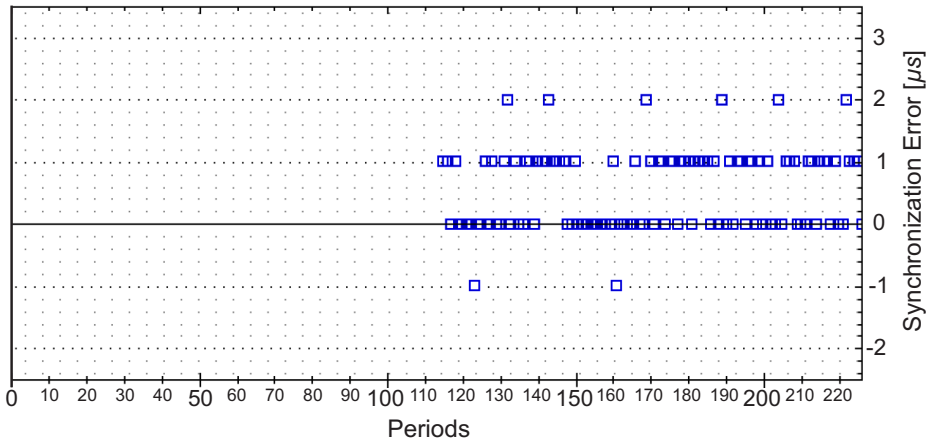

Fig. 4. Synchronization error after startup of a node at period 115.

7 Conclusion and Outlook

In this paper we presented the practical problems and experiences of our real-world implementation, when integrating a simple time-stamped synchronization into a periodical MAC protocol. We therefore introduced the underlying MAC protocol EXTENDED-DESYNC and surveyed some yet available time synchronization protocols. We sketched the desired characteristics of our integrated approach, but also the features provided by the used hardware and the used operating system. During the integration, several points of interest arose, e.g. the non-deterministic time delay between send command and start of transmission, or the deep interrupt time-stamping which is provided by the used operating system. We also established some testbeds to verify the performance of our time synchronization and to explain some interesting scenarios, for example when using a multi-hop topology, or when a node joins a yet synchronized single-hop topology.

For future research, we plan further real-world testbeds to analyze the clock drift and the impact of drift compensation methods like linear regression. The need for drift compensation in relation to the length of the period T is of interest, too. With it, we want to evaluate the accuracy, memory usage, and computational complexity of diverse compensation algorithms. We also aim to compare implementations of the time synchronization protocols named within this paper under identical conditions. Maybe, some synchronization protocols benefit from some features presented within this paper, e.g. the deep interrupt time-stamping (cf. Sect. 4.2). Other plans mainly affect the EXTENDED-DESYNC protocol: overhead reduction ranks first, followed by the analysis of the protocol's operability under more realistic circumstances, like asymmetrical or unreliable links, as well as individual periods.

References

1. Akyildiz, I.F., Kasimoglu, I.H.: Wireless sensor and actor networks: research challenges. Ad Hoc Networks 2(4), 351–367 (May 2004)
2. Baunach, M.: Dynamic Hinting: Real-Time Resource Management in Wireless Sensor/Actor Networks. In: RTCSA. pp. 31–40. IEEE Computer Society (Aug 2009)
3. Degesys, J., Rose, I., Patel, A., Nagpal, R.: DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks. In: Abdelzaher, T.F., Guibas, L.J., Welsh, M. (eds.) IPSN. pp. 11–20. ACM, Cambridge, MA, USA (2007)
4. Elson, J., Girod, L., Estrin, D.: Fine-grained network time synchronization using reference broadcasts. In: In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI). pp. 147–163. Boston, MA, USA (Dec 2002)
5. Ganeriwal, S., Kumar, R., Srivastava, M.B.: Timing-sync Protocol for Sensor Networks. In: SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems. pp. 138–149. ACM Press, New York, NY, USA (2003)
6. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System Architecture Directions for Networked Sensors. SIGPLAN Not. 35(11), 93–104 (2000)
7. IQD Frequency Products Ltd: HC49/H4 Crystals (Nov 2008), http://www.iqdfrequencyproducts.com/products/details/4/
8. Kang, H., Wong, J.L.: A Localized Multi-Hop Desynchronization Algorithm for Wireless Sensor Networks. In: INFOCOM. pp. 2906–2910. IEEE (2009)
9. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. John Wiley & Sons, Ltd (2005)
10. Maróti, M., Kusy, B., Simon, G., Lédeczi, Á.: The Flooding Time Synchronization Protocol. In: Stankovic, J.A., Arora, A., Govindan, R. (eds.) SenSys. pp. 39–49. ACM (2004)
11. Marrón, P.J., Minder, D., Lachenmann, A., Saukh, O., Rothermel, K.: Generic Model and Architecture for Cooperating Objects in Sensor Network Environments. African Journal of Information & Communication Technology 2(1), 1–11 (Mar 2006)
12. Meier, L., Blum, P., Thiele, L.: Internal Synchronization of Drift-Constraint Clocks in Ad-Hoc Sensor Networks. In: MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing. pp. 90–97. ACM, New York, NY, USA (2004)
13. Mirollo, R.E., Strogatz, S.H.: Synchronization of Pulse-Coupled Biological Oscillators. SIAM Journal on Applied Mathematics 50(6), 1645–1662 (1990)
14. Mühlberger, C., Kolla, R.: Extended Desynchronization for Multi-Hop Topologies. Tech. Rep. 460, Institut für Informatik, Universität Würzburg (Jul 2009)
15. Namtvedt, S.: Design Note DN506 - GDO Pin Usage. Texas Instruments Inc., Dallas, Texas (USA) (Oct 2007), http://focus.ti.com/lit/an/swra121a/swra121a.pdf
16. Patel, A., Degesys, J., Nagpal, R.: Desynchronization: The Theory of Self-Organizing Algorithms for Round-Robin Scheduling. In: SASO. pp. 87–96. IEEE Computer Society, Cambridge, MA, USA (2007)
17. Prasad, R., Ruggieri, M.: Applied Satellite Navigation Using GPS, GALILEO, and Augmentation Systems. Artech House (2005)
18. Römer, K.: Time synchronization in ad hoc networks. In: MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile Ad hoc Networking and Computing. pp. 173–182. ACM, New York, NY, USA (2001)

19. Römer, K., Blum, P., Meier, L.: Time Synchronization and Calibration in Wireless Sensor Networks. In: Stojmenović, I. (ed.) Handbook of Sensor Networks: Algorithms and Architectures, chap. 8, pp. 199–237. Wiley and Sons (Oct 2005)
20. Su, W.: Overview of Time Synchronization Issues in Sensor Networks. In: Zurawski, R. (ed.) Embedded Systems Handbook: Networked Embedded Systems, chap. 5, pp. 5–1–5–12. CRC Press, 2nd edn. (2009)
21. Texas Instruments Inc., Dallas, Texas (USA): MSP430x1xx Family User's Guide (2006), http://focus.ti.com/lit/ug/slau049f/slau049f.pdf
22. Texas Instruments Inc., Dallas, Texas (USA): CC1100 Low-Power Sub-1 GHz RF Transceiver (May 2009), http://focus.ti.com/lit/ds/symlink/cc1100.pdf
23. Texas Instruments Inc., Dallas, Texas (USA): MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller (May 2009), http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf
24. Tobagi, F.A., Kleinrock, L.: Packet Switching in Radio Channels: Part II–The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution. Communications, IEEE Transactions on 23(12), 1417–1433 (1975)
25. UC Berkeley: TinyOS. http://www.tinyos.net/ (Jul 2010), http://www.tinyos.net/
26. Vishay Intertechnology, Inc.: Low Profile Holder Type Crystal Units - XT49S (Mar 2010), http://www.vishay.com/docs/35014/xt49s.pdf