Maintaining Constructive Interference Using Well-Synchronized Sensor Nodes

Michael König
mikoenig@ethz.ch
Distributed Computing Group
ETH Zürich, Switzerland

Roger Wattenhofer
wattenhofer@ethz.ch
Distributed Computing Group
ETH Zürich, Switzerland

Abstract—**Traditionally, achieving constructive interference (CI) required specialized timekeeping hardware. Recently, the ability and interest to employ CI distributedly at any time using groups of ordinary single antenna wireless sensor nodes have grown.**

In this paper, we investigate achieving CI on sensor nodes. We consider the commonly employed IEEE 802.15.4 wireless standard, which uses a chip frequency of 1 MHz. This means signals need to be synchronized with an error below 0.5 microseconds to allow for CI. Hence, excellent clock synchronization between nodes as well as precise transmission timing are required.

We implemented and tested a prototype addressing the implementation challenges of synchronizing the nodes' clocks up to a precision of a few hundred nanoseconds and of timing transmissions as accurately as possible. Our results show that, even after multiple minutes of sleep, our approach is able to achieve CI in over 30% of cases, in scenarios in which any influence from the capture effect can be ruled out. This leads to an increase in a packet's chance of arrival to 30-65%, compared to 0-30% when transmitting with either less synchrony or different data payload. Further, we find that 2 senders generally increase the signal power by 2-3 dB and can double the packet reception ratio of weak links.

I. Introduction

In recent years, the benefits of constructive interference (CI) have been discovered for wireless sensor networks: By synchronizing identical signals well enough, the interference caused by the collision is not destructive but rather unnoticeable or even strengthening. On the one hand, this allows the development of algorithms which can expect identical packets not to destructively interfere. On the other hand, more unstable and longer links can be used more reliably through the strengthening of the signal.

Traditionally, the frugal nature of sensor node hardware and timekeeping moved synchronizing clocks and transmissions out of reach for commodity sensor nodes. In the past, several workarounds have been proposed to nevertheless enable the signals of separate nodes to interact meaningfully:

- One way is to provide a highly accurate external global clock source by some means, e.g. GPS, with each sensor node.
- Another approach is to forgo the ability to send fully articulated data packets – and to simply transmit waveforms instead, which are unlikely to fatally destructively interfere [1].
- Most recently, Glossy [2] explored how to leverage incoming reference packets to send a packet immediately afterwards with an almost constant delay. When multiple nodes use the same reference packet (or different well-synchronized reference packets), their outgoing packets are then synchronized enough to achieve CI.

All of these approaches have in common that they try to avoid dealing with the rather unreliable internal clocks of commodity sensor nodes. In this paper, we explore the possibilities of creating synchrony sufficient for harnessing CI without relying on any of these workarounds. By meticulously keeping time using the available local clocks to extrapolate a global time value, and by minimizing the variance in packet departure delay, we manage to at least partially obtain this goal. In particular, we present a proof of concept implementation achieving CI on the popular TelosB sensor nodes, confirming the viability of our approach: Even for only 2 senders we obtain an increase in signal strength of at least 2 dB in 60% of cases and an increase in packet reception ratio of 20-35% when signal strengths are equal. For ideal results, we need the last synchronization of the senders to lie at most 20 seconds in the past when attempting to send simultaneously, but even after multiple minutes of sleep our approach can benefit from CI. We carefully compare several parameters in order to understand the limits of CI on sensor nodes. For instance, already a synchronization error in the order of 1 µs is problematic. More precisely, as we are using the IEEE 802.15.4 standard, the transmissions need to start within less than 0.5 µs of each other to avoid destructive interference.

Hence, a significant part of this work is dedicated to synchronizing the clocks of the sensor nodes, maintaining this synchrony and, finally, precisely dictating the time span of the wireless transmission. Clock synchronization for wireless nodes in general is a well-studied problem, but the synchronization precision requirements by our setup are more stringent than those of the common use cases such as TDMA.

II. Related Work

A. Clock Synchronization

Time synchronization between network nodes has been studied for a long time even before wireless nodes became widespread. The 30 years old Network Time Protocol (NTP) [3] is still in use today for time synchronization between two machines over the Internet, achieving accuracies of about 10 milliseconds. Recently, adjustments have been proposed improving its accuracy to about 1 millisecond [4], which is still off by a factor of over 1000 from the precision we require.

Wireless networks face additional challenges [5] but also have additional options at their disposal due to the different nature of their medium. Römer [6] proposes a synchronization algorithm focusing on sparse ad hoc networks. Dozer [7] minimizes energy consumption by keeping sender-recipient pairs synchronized over long periods of time and only rarely waking them for brief scheduled rendezvous exchanges. To maintain synchrony in spite of clock drift, the drift is modeled and compensated for. We employ a similar scheme, see Appendix B.

RBS [8] makes use of the broadcasting nature of every wireless transmission to synchronize multiple nodes with a single transmission. RBS achieves synchronization accuracy on the order of microseconds. TPSN [9] builds a hierarchical tree structure for synchronization from a root node. TPSN also introduces the concept of MAC layer timestamping, which we build upon in this work (see Appendix A). FTSP [10], [11] floods periodical synchronization waves through the network, improving accuracy to around 1 µs. Further improvements for synchronizing networks of nodes are RITS [12], reactively synchronizing nodes using global events, and RATS [13] and PulseSync [14], which propose rapid network-wide flooding while employing schedules to avoid collisions.

An analysis of the single-hop and multi-hop performance of the different synchronization protocols and a proposal for a non-hierarchical synchronization method improving the multi-hop case can be found in [15].

B. Constructive Interference

Already early on, primarily flooding algorithms became privy to the possible gains of using non-interfering signals and thus being able to flood the network without the overhead of building and adhering to a flooding schedule. We distinguish single and multiple source flooding: while in single source flooding a node desires to disseminate information or an event in the whole network, in multiple source flooding any number of nodes may raise redundant alarm events which need to be propagated. Lu et al. [16] proposed a single source flooding protocol, which simply ignores collisions between neighboring nodes at the flooding front, leaving it to the capture effect to let each node receive a copy of the message sooner or later. They show that such recklessness can in fact improve latency by as much as 80%.

Slotos [1] implemented multiple source flooding in form of an alarm system in which an alarm signal, a simple waveform, was propagated through a network by nodes which received the alarm starting to send out the signal as well. In this case the concurrent signals were not artificially aligned but the resulting amount of destructive interference was acceptable to the protocol, as the signal did not carry any data beyond its existence. Similarly, the Black Burst Synchronization (BBS) scheme [17] employs non-destructively interfering pulses, so called "black bursts". Dutta et al. [18] exploit the fact identical acknowledgment packets automatically sent by the receiving hardware are aligned well enough to cause CI. While essentially all of these approaches have the common disadvantage of having very little control over the data being sent, this is well suited for multiple source flooding, as such events typically are not expected to carry detailed information.

More recently, Glossy [2] proposed a technique by which simple single antenna sensor nodes could send with unprecedented synchrony, leading to a fast single source flooding algorithm avoiding collisions through constructive interference and the capture effect. This technique is based on tying the departure time of each packet to the end of the incoming transmission of the previous packet, effectively reducing the timespan during which clock skew can erode synchrony. The goal of this paper is to do away with this restriction while still allowing collision-free transmission of data packets instead of mere events.

The introduction of this new tool spawned a multitude of applications: Not only have several new protocols for flooding and data dissemination been developed, e.g. [19], [20], but also more advanced packet pipelining has been proposed [21]. A general model for the occurrence of CI and the capture effect was proposed by Yuan et al. [22].

III. Experiment Setup

A. Hardware

As wireless nodes for our experiments we used the Tmote Sky sensor nodes (also known as "TelosB") [23]. They feature a 16-bit RISC microcontroller, the TI MSP430, and a TI CC2420 wireless transceiver. The clocks available to the MSP430 are a 32 kHz (2^{15} Hz) quartz crystal as well as an internal digitally controlled oscillator (DCO) which serves as the pulse generator for the instruction execution. The frequency of the DCO is very prone to being skewed through fluctuations in temperature and voltage, but can generally be configured to be anywhere in the range from 100 kHz to 5 MHz.

The CC2420 is tailored to support IEEE 802.15.4 and offers many convenience features such as framing given packet data including automatically including a 16-bit CRC checksum field in the packet footer. We rely on this checksum mechanism as our main way to tell whether a received packet has been corrupted. The CC2420 offers a range of transmit powers from -30 dBm to 0 dBm. We make use of this feature as it is easier to adjust to output power than to physically move the nodes every time changes in the environmental conditions require it (see Section VII).

We made use of an installation of 30 of these sensor nodes part of the wireless testbed FlockLab [24], placed at various locations at the ceiling of the same floor in an office building, some inside enclosed offices, some on the hallway. This allowed us to consider a variety of real life scenarios.

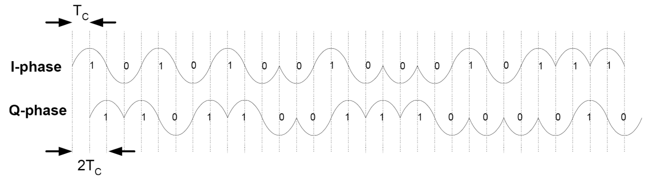


Fig. 1: A figure from the CC2420 Manual [25] depicting the pseudo-random modulation pattern of the zero symbol on the I and Q phases. T_C is defined to be 0.5 µs by [26], i.e., the chip rate is 1 Chip/$T_C = 2$ MChips/s.

B. *Software*

On the sensor nodes we run code based on Contiki version 2.7. We chose Contiki because it allows making the kind of low-level modifications we required the easiest. In particular, we overhauled the clock module which is concerned with using the MSP430's Timer_A (configured with the 32 kHz quartz crystal as clock source) to count elapsed seconds and periodically test for expired user-defined "timers". We decrease the time span between timer interrupts to $\frac{1}{2^{10}}$ seconds, and instead of counting elapsed seconds we count the occurrences of overflow interrupts for the Timer_A timer register.

Contiki also offers a mechanism to periodically re-configure the DCO to best match the desired frequency. We make use of this to keep the DCO frequency near 2^{22} Hz (roughly 4 MHz), the highest supported power-of-two multiple of the quartz crystal frequency (2^{15} Hz). This essentially subdivides each tick of the quartz crystal into 2^7 sub-ticks, i.e., increasing the combined precision by 7 bit (see Section V).

The DCO is usually turned off to conserve energy during idle low-power phases of the processor. While our approach allows such sleep phases, we naturally do require the DCO to have been running since at least the last quartz crystal tick to be able to provide accurate timestamps. This incurs an overhead of up to $1/2^{15}$ seconds of DCO operation after each wake-up.

We also configure the CC2420 driver to not use CCA (clear channel assessment) for packet transmission and extend it to support recording 64-bit timestamps on incoming packets and adding 64-bit timestamps to outgoing packets (see Appendix A).

For the process of the experiments themselves see Section VII-A.

IV. Timing Requirements

The IEEE 802.15.4 standard specifies the encoding of the raw logical data stream to electromagnetic signal as follows: First, the data is segmented into groups of 4 bits, called "symbols". Each of the 16 possible symbols is then mapped to a certain pseudo-random noise sequence of 32 binary "chips". Finally, the chip sequences are concatenated and modulated using O-QPSK, i.e., every chip is modulated alternatingly onto the I and Q phases, offset by half a chip duration. At a chip rate of 2 MChips/s this pans out to 62,500 symbols per second or a 250 kbit/s data rate. See Figure 1 for an example modulation of the zero symbol (corresponding to 4 zero bits).

In the ideal case all the senders' (identical) chip sequences arrive at the recipient node at the same time overlapping each other at a small random (carrier) phase offset, likely causing CI and thus increasing the signal strength. Unfortunately, such alignment does not come easily, and in the remainder of this section we will discuss the different sources of signal misalignment.

As an upper limit for the acceptable signal shift between two signals we can identify $T_C = 0.5$ µs, half the duration of a single chip within one of the two phases. If the signal shift is any greater, every chip of the second signal will superimpose the chip subsequent to the corresponding chip of the first signal more than the actually corresponding chip. This would cause both of the signals to be harder to decode correctly as some of the bits of each signal will experience destructive interference, i.e., the reduction of signal strength in a chip through summing up of opposing signal values. This upper limit of 0.5 µs is certainly not tight as that is merely the point at which chips certainly cannot be unambiguously matched to a signal anymore. In practice, we are aiming for a signal shift of 0.2 µs or lower to ensure a strong CI effect. When using a set of more than 2 senders, we aim to minimize the shift between every pair of signals.

We identify 3 main sources of signal misalignment between the signals of two senders A and B: the clock synchronization error e_{clock}, the transmission timing error $e_{transmit}$, and the difference in the signal travel times e_{travel}:

$$e_{total} = e_{clock} + e_{transmit} + e_{travel} \overset{!}{\ll} 0.5 \text{ µs}$$

Each of the components can be positive or negative and thus they can cancel each other out. They are defined as follows.

The clock synchronization error is simply the difference in the senders' views of the global clock value. We discuss reducing this error in detail in Section V.

$$e_{clock} = clock_A - clock_B$$

The transmission timing error of a node is the delay the node starts the transmission after a given desired local time. The difference in transmission timing errors (delays) at each node forms $e_{transmit}$. We discuss reducing this error in detail in Section VI.

$$e_{transmit} = e_{transmitB} - e_{transmitA}$$

Finally, the travel time error is the difference in each senders' signal's travel time.

$$e_{travel} = traveltime_B - traveltime_A$$

The travel time error is of note because at the precision of time we are working with here – tens of nanoseconds – the travel time of the electromagnetic waves becomes significant, even on a testbed located in a medium-sized office building floor. For instance, if one of the senders is 30 meters further away from the recipient than the other the difference in signal travel time is 100 nanoseconds.

We tried measuring these travel times by measuring the round-trip times of links and subtracting the time the responding node measured between the arrival of the incoming and the departure of the outgoing packet. Two factors thwarted this attempt: The CC2420 transceiver we are using experiences a "data latency", which denotes the time between the sender and the receiver activating their SFD ("start of frame delimiter") pin, which indicates the start of an incoming or outgoing transmission. This delay is apparently caused by the processing of the non-packet signal in the receiver and specified to be $3\,\mu s$ by the CC2420's data sheet. Our measurements show that this delay is probably closer to 3.6 μs and that it can fluctuate by several dozen nanoseconds in subsequent measurements. Additionally, we observed the round-trip time to vary by hundreds of nanoseconds over the course of a day. This is likely caused by environmental effects, such as temperature influencing the wireless transceiver's clock and the opening and closing of doors in the office building changing the actual travel times, although we doubt that hundreds of nanoseconds of difference can be explained by travel time changes. In the end, we did not come to a conclusive explanation and decided to simply make best effort to eliminate this error: we pick senders at roughly equal distance to the recipient node and adjust for the "data latency" with a constant time offset of 3.6 μs.

V. Clock Synchronization

In general, quartz crystals exhibit a more stable and thus more desirable behavior than digitally controlled oscillators (DCOs), both in terms of long term drift as well as short term frequency stability. The TelosB possesses 2 quartz crystals: a 32768 Hz one attached to the MSP430 for general high-accuracy timekeeping, and a 16 MHz one attached to the CC2420 transceiver used for signal processing and as data rate reference. Unfortunately, the latter is not accessible to the CPU, so we need to make do with the MSP430's built-in DCO as sole source for high granularity clock ticks: It offers a resolution of about 0.24 μs, compared to the resolution of the 32 kHz quartz crystal of about 30 μs.

To get the best from both worlds – the stability of the quartz crystal and the precision of the DCO – we build a hybrid clock, similar in spirit to the method proposed in [27]. The 2^{15} Hz quartz crystal has priority and ensures relatively high stability over the long term while the DCO, set to a speed as close to 2^{22} Hz as possible, fills out the time span in between the quartz crystal ticks with $2^{22}/2^{15} = 2^7$ finer-grained ticks. To accomplish this, we configured the MSP430 to copy the current value of the DCO clock (a 16-bit counter) at each quartz crystal tick into a special capture register. The MSP430 allows doing this in parallel with regular instruction execution at no additional overhead. A complete current time value could then be obtained by subtracting the current DCO counter from the value in the special capture register and using the result as 7 bits of additional precision together with the current quartz

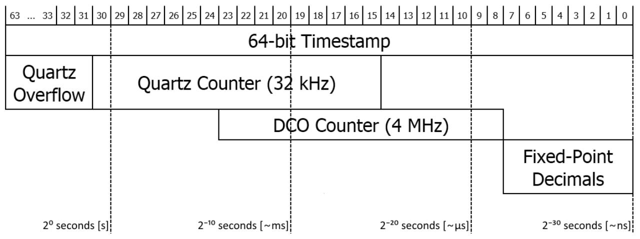


Fig. 2: The final layout of our timestamps with the different clock sources in different rows. The quartz clock has priority over the DCO clock, so actually only 7 bits of the DCO clock are used. No clock we use is able to specify the lowest 8 bits, but these bits allow for storing extra precision when a timestamp is the result of arithmetic operations.

crystal counter. Additionally, we keep the number of times the quartz crystal counter overflowed in a variable. This variable is incremented in an overflow interrupt handler causing a negligible overhead every 2 seconds. Figure 6 shows how a separate DCO clock would move in relation to the quartz crystal. Not only is there a clear drift and oscillating behavior (both of which could be accounted for to some degree), but there also is a notable amount of randomness that would make relying on the DCO alone as a clock source a poor choice.

As all the MSP430's registers only hold 16 bits, the question of the size for timestamps arises, as for the transmission between nodes they should denote globally unambiguous values. 16-bit variables do not suffice as the DCO clock alone wraps around 256 times a second. 32-bit variables would wrap every 1024 seconds or roughly 17 minutes, and since we want to enhance our timestamps by an additional 8 bits of precision when extrapolating (see Appendix B), we settled on 64-bit variables. Such variables are well supported by our compiler (a GCC variant for the MSP430), but special care has to be taken when performing arithmetic operations on variables of this size, as they quickly grow to require hundreds of instructions on 16-bit RISC CPUs. The final layout of our timestamps can be seen in Figure 2. The precision of these timestamps is $1/2^{30}$ seconds, roughly a nanosecond, and the range is sufficiently large (several years).

For our implementation of MAC layer timestamping and clock drift compensation refer to Appendices A and B. In the end, we managed to solve the problem of single-hop clock synchronization on TelosB as well as possible but still have to admit an error e_{clock} of up to 250 nanoseconds. However, as the error is almost uniformly distributed, we will hit cases as good as $|e_{clock}| \leq 50$ nanoseconds over 20% of the time.

VI. Transmission Synchronization

Even if our clock was perfectly synchronized to the global clock, transmitting a packet exactly at some given time is not a simple task. Once the transmission of a packet has begun, its synchronization error will no longer change, since its transmission is now controlled by the CC2420's 16 MHz quartz

$t_{\text{SFD}} - t_{\text{STXON}}$	Instance Count	Fraction of Total
1593	15	0.26%
1594	893	15.71%
1595	3726	65.53%
1596	1049	18.45%
1597	3	0.05%
Σ	5686	100.00%

TABLE I: Measured STXON→SFD times collected from a single test run with 15 nodes. The time differences are specified in DCO ticks.

crystal. However, the delay between reaching the desired departure time and beginning the transmission is not necessarily constant. It can be split into two parts: the time between reaching the desired departure time and issuing the STXON command strobe (requesting the transmission of a previously loaded packet) to the transceiver, and the time between issuing STXON and the actual start of the packet transmission. We measured the latter of these two by comparing the DCO clock values right before issuing the STXON command and at the SFD event (see Appendix A). The distribution of the measured values are shown in Table I. The fluctuations are well within the limits of the frequency noise the DCO experiences (see Figure 6). While it is possible that there is a small variable delay, it is not discernible and we assume this time span to be constant.

It remains to scrutinize the time span between reaching the target time and issuing STXON. Initially, we employed busy waiting: once the target time was close, we stop relinquishing control to the operating system and enter a loop in which all we do is query the time until the target time has been passed. Immediately after, we call the driver routine to start the transmission (one of the first instructions of which is to issue STXON).

Listing 1: Simple Busy Wait

```
if (TargetTime - GetGlobalTime() < 10 ms) {
    while (TargetTime > GetGlobalTime())
        ; // do nothing
    cc2420_driver.transmit();
}
```

This approach experiences a large variance in loop exit times and thus transmission times (relative to the target time) due to the large amount of instructions within the loop: assembling the local timestamp requires 55 instructions, computing the global time from it requires at least 110 instructions and comparing it to the target time requires 22 instructions. To address this problem we made 3 changes. The first change was to apply our model of the global time "backwards" to compute the target local time before entering the busy wait loop. The second change was to decompose the target local time into its 3 clock sources (see Figure 2) and spin on the 3 16-bit clocks one after another. These two changes reduce each loop to the minimum of 3 instructions. See Listing 2 for an approximate implementation. TAR is the quartz counter register, TBR is the DCO counter register and TBCCR6 contains the value of TBR at the last quartz crystal tick.

Listing 2: Busy Wait Split by Clock Sources

```
void await(uint64_t local_target) {
    uint16_t target_tarof =
        (local_target >> 23) & 0xFFFF;
    uint16_t target_tar =
        (local_target >> 7) & 0xFFFF;

    while (TAR_overflows < target_tarof)
        ;
    while (TAR < target_tar)
        ;
    uint16_t target_tbr =
        (local_target & 0x007F) + TBCCR6;
    while (TBR < target_tbr)
        ;
}
```

It is worth noting that not every instruction requires the same amount of time to be executed. However, due to the absence of caches and pipelining in the MSP430's architecture, the execution time of any given instruction can be specified exactly in terms of a number of CPU cycles, which directly correspond to the impulses generated by the DCO.

Listing 3: Busy Wait Assembly

```
.L24:
mov &__TBR, r15 ; 3 cycles
cmp r12, r15    ; 1 cycle
jlo .L24        ; 2 cycles
```

The third and final change we made was to insert NOPs (1-cycle "no operation" instructions) ahead of the final loop to ensure we exactly matched the target time when exiting the final loop. The three instructions comprising the loop execute in exactly 6 cycles (see Listing 3), so before the loop we wait for $(\text{target\_tbr} - \text{TBR}) \mod 6$ cycles by using a jump table into a series of NOPs.

After applying these optimizations, the time offset between the target time and the issuing of the STXON command strobe was constant in over 80% of cases. By artificially delaying the next periodic timer interrupt before entering the last busy-wait loop to a point 1 ms in the future, this number increased to $> 99\%$ of cases.

The constant delay is easily adjusted for, leaving the only remaining transmission error we can observe the delay between issuing STXON and registering the SFD event as measured by the DCO clock. Note that in this case the DCO clock is used only for measuring and does not in any way influence the wireless transceiver preparing for the transmission. Thus, it is a reasonable assumption that the jitter seen in Table I is merely a product of the DCO's instability, albeit we cannot rule out that the transceiver itself introduces a small variable delay.

Finally, as with clock synchronization, we suffer a transmission error $e_{transmit}$ of up to 250 nanoseconds, the resolution of our most precise clock, independent of the clock synchronization error. However, this error is also distributed uniformly in the interval $[-1/2^{23}\text{seconds}, +1/2^{23}\text{seconds}]$ and in a certain fraction of all samples $|e_{transmit}|$ will be acceptably small.

VII. Constructive Interference

A. Experiment Procedure

A particular concern when designing the experiments was to ensure we would be able to discern which transmissions were successful due to CI and which were successful due to the so-called capture effect. The capture effect is a phenomenon in FM wireless receivers to be able to commit to a transmission and continue to decode it correctly in spite of other later transmissions starting during the duration of the first one. For this to work, the "captured" transmission needs to be slightly stronger than the sum of the remaining transmissions' signal powers. On our testbed we found the required extra signal power to vary between 2 or 3 dB, highly depending on the nodes (see Figure 3). As link qualities in real life scenarios and on our testbed can easily vary by ± 3 dB within minutes or sometimes even mere seconds, completely avoiding power settings in which the capture effect can occur is not feasible. To nevertheless be able to detect the capture effect, we thus frequently measure the quality of every link used.

In our experiments we considered several tuples of nodes, where one node of each tuple was designated the recipient and was known to have somewhat stable links to the other nodes, the senders. We proceed in rounds, in which first we let all senders send simultaneously twice: first with the same data packet and then with data packets individual to the senders. Then, as a second stage, each of the senders sends alone once. Finally, the recipient gives the senders feedback on the round and optionally also supplies them with fresh synchronization information. During each round the senders keep their sending powers constant. The purpose of the second stage is to ensure we know the received signal strengths (RSS) for all senders. Because a round takes less than half a second, we make the assumption that the wireless environment does not change significantly during the vast majority of all rounds. Knowing the difference in RSS values is vital for us, as it allows us to discern successful receptions which may have been caused by the capture effect, as discussed above. We fix one of the senders' transmission powers at a medium value, such that the other senders will be able to create RSS values both stronger and weaker than the fixed sender. The other senders iterate over a range of power settings whose limits are regularly adapted based on the feedback received between rounds.

B. Results

First, consider the case of 2 senders. Figure 3 shows the relation between the difference in received signal strengths (RSS) and the packet reception ratio (PRR) for senders sending the same data versus sending different data, in which case the senders would set every symbol of the packet payload to a symbol corresponding to their ID. A large positive RSS difference means sender B, which is iterating over different transmission powers over the rounds, was received a lot stronger than sender A, whose transmission power stays constant. Which sender was received can be seen when different data is sent, as is shown by the dashed lines. This plot is particularly interesting because it highlights the interplay of CI and the capture effect: if either sender is received a lot more strongly than the other, it will be received correctly regardless of it sending a different packet. However, as the signal powers become more similar, destructive interference occurs more often. By sending the same data, we are able to avoid a portion of the destructive interference, strongly implying the occurrence of CI.

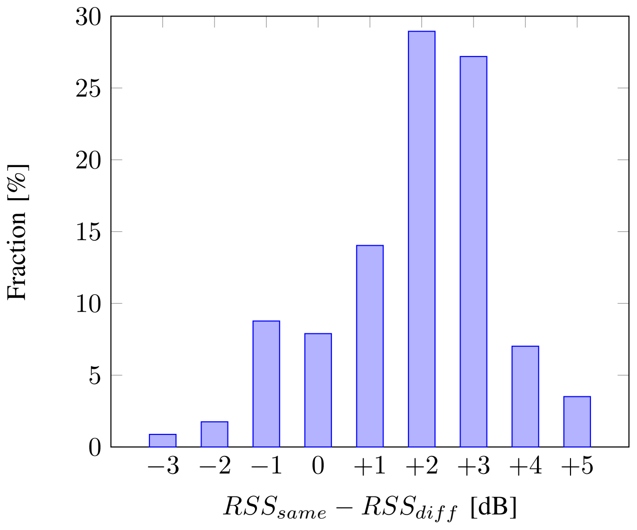


Fig. 4: The distribution of of the measured RSS gain of simultaneous sending versus either of the two senders, given both senders sending at the same power.

We were able to observe CI to varying degrees for any triplet of nodes, as long as the senders are capable of creating signals with similar strength at the recipient. As a result of CI the PRR observed at equal RSS increases by 25-35% of samples. The width of the gap between one sender dominating and the other sender dominating appears to be a property of the hardware and environment of a certain node tuple, but does not appear to be connected to the CI we achieve.

Figure 4 displays the distribution of the signal strength gained through CI at equal RSS for 2 senders. We observe a large variation both due to an inherent gray area in link quality and due to the variation in transmission synchrony achieved. However, in over 65% of cases we measure an increase of signal strength by 2 dB or more.

We also conducted experiments with 3 and 4 senders, making an even larger case for CI as the capture effect is known to occur less and less as the number of senders increases [16]. Tables II, III and IV display the PRRs aggregated in two different ways for one run with 3 senders and one run with 4 senders. Empty cells correspond to configurations with fewer than 5 samples. While there is a significant fluctuation in values due to the unavoidably low number of samples in many cells, as we cannot dictate the RSS values, a few clear trends can be observed. Most visibly, the same data case usually exhibits a PRR 25-40% higher than the different data case. Further, the capture effect, whose occurrence can easily be

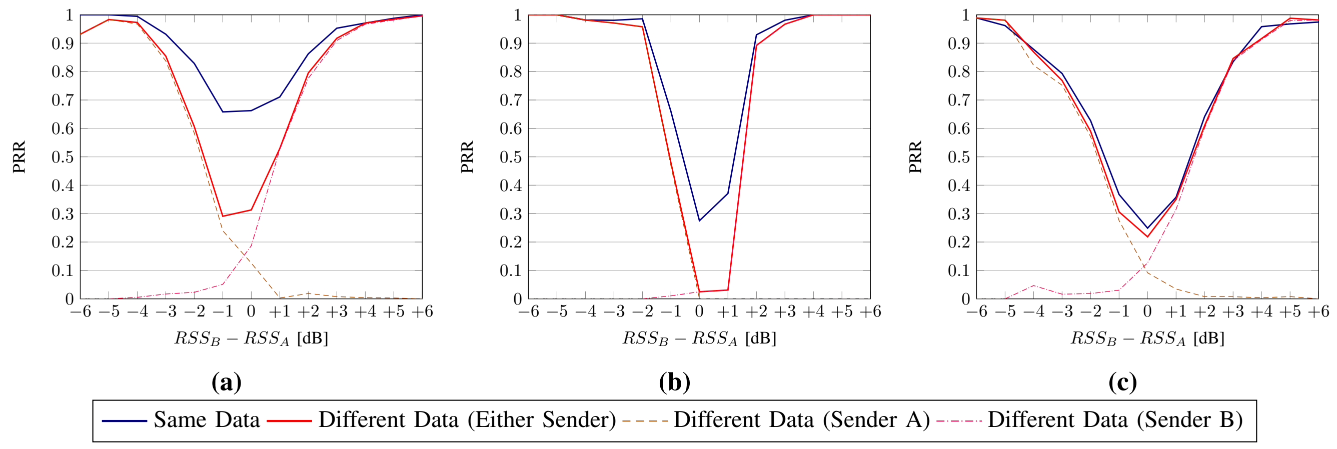


Fig. 3: The PRR plotted against difference in RSS for two senders for 3 different node triplets, clearly showing CI occurring at a RSS difference close to zero for each triplet, while the capture effect causes larger differences to almost always succeed. (a) and (b) show two pairs of senders with differing behavior. (c) depicts the result of deliberately mistiming one of the senders by 1 µs, preventing any CI from occurring.

Same Data	0	1	2	3	4	5
0						
1	32%	35%				
2	39%	35%	34%			
3	67%	36%	40%	41%		
4		43%	48%	41%		
5	58%	59%	51%	46%	42%	56%
Different Data	**0**	**1**	**2**	**3**	**4**	**5**
0						
1	0%	2%				
2	0%	0%	1%			
3	7%	3%	2%	0%		
4		14%	9%	8%		
5	83%	50%	21%	3%	3%	0%

TABLE II: The PRRs measured in an experiment with 3 senders. The results are split by RSS difference between the senders: rows correspond to the difference between the weakest and second weakest sender, columns correspond to the difference between the weakest and strongest sender. (Column/row labels are in dB.)

Same Data	0	2	6	10
0	34%			
2	36%	38%	56%	
4	51%	48%	52%	50%
6	70%	91%	78%	62%
8				100%
Different Data	**0**	**2**	**6**	**10**
0	2%			
2	1%	2%	0%	
4	23%	10%	12%	0%
6	80%	78%	50%	24%
8				67%

TABLE III: The same data as in Table II, aggregated differently: the results are split by average RSS difference to the strongest sender (rows) and the variance amongst the powers of the non-strongest senders (columns). (Column/row labels are in dB.)

Same Data	0	2	4	6	8	10
0	10%					
2	23%	24%	36%	21%		17%
4	30%	32%	30%	18%	9%	56%
6	25%	34%	36%	33%	22%	
8	50%	50%	50%			
Different Data	**0**	**2**	**4**	**6**	**8**	**10**
0	0%					
2	1%	0%	0%	0%		0%
4	1%	1%	2%	0%	0%	0%
6	4%	6%	9%	6%	6%	
8	30%	28%	29%			

TABLE IV: The same aggregation as in Table III, but for a different experiment with 4 senders.

discerned from the different data case, occurs more often the further the strongest sender's signal strength is from the rest (bottom left in Table II, bottom and to a lesser degree also left in Tables III and IV). Finally, particularly high values can be found in the bottom right of Table II, which are likely a result of the possibility of the two much stronger signals invoking both CI as well as the capture effect to survive the occasional timing errors in the weakest signal.

As a further confirmation of our approach, we also conducted a few runs of the experiment in which we configured one of the senders to deliberately send 1 µs late. The expectation is to have this completely remove all possibility for CI. The result of one such run can be seen in Figure 3(c), exhibiting almost no CI at all. This underlines the importance of the synchronization precision we strove for. Similarly, when increasing the resynchronization interval we found the effect to diminish after varying periods of time: in some cases as soon as after 30 seconds, in others not for multiple minutes. This is attributable to the clock synchronization error rising due to changes clock drift, which drift compensation can no longer account for (see Figure 7).

For the packets sent by the senders we used payload sizes ranging from 12 to 26 bytes. Together with the 2 final checksum bytes, 4 bytes of preamble and 1 SFD byte, transmissions were thus $19 - 33$ bytes or $608 - 1056$ µs in length. The different packet sizes did not exhibit a significant difference in performance.

VIII. Conclusion

We predicted the total allowable error to be the sum of the clock synchronization error, the transmission synchronization error and the travel time error:

$$e_{total} = e_{clock} + e_{transmit} + e_{travel} \overset{!}{\ll} 0.5\ \mu\text{s}$$

We reduced e_{clock} by combining the TelosB's 32 kHz quartz crystal with the MSP430's DCO to obtain a stable 4 MHz clock, and we reduced $e_{transmit}$ by optimizing our transmission code path down to the CPU cycle. Both of these values were minimized into the range of the length of a single clock tick (2^{-22} s). In an attempt to reduce e_{travel}, we picked senders such that they had roughly the same distance to the recipient nodes.

Using our example implementation we conducted several experiments with 2 to 4 senders trying to create CI by simultaneously sending the same packet to a recipient node. To ensure packets were not arriving merely due to the capture effect we also sent packets with differing data. Our results show an increase in signal strength of at least 2 dB in 60% of cases for 2 senders, and an increase in PRR of 20-35% for any amount of senders when signal strengths are equal. Considering we expected to only meet the necessary signal alignment requirements in the fraction of samples, in which both e_{clock} and $e_{transmit}$ happened to be small enough, these 20-35% appear quite satisfactory and are likely mainly held back by the precision of the underlying hardware.

While our implementation and results are based on the TelosB and the 802.15.4 standard, we believe the concepts to apply in a similar fashion to most sensor nodes and wireless standards. Further, there are two properties the TelosB was lacking, which were stunting our results: 1) a clock with a precision a magnitude higher than the used chip length and 2) a transceiver designed for a similar precision in transmission timing.

In conclusion, we showed that maintaining the option of CI on commodity sensor node hardware over longer periods of time is feasible, through extraordinary inter-node synchronization and without incurring the overhead of a global "wave" of synchronizing packets before every transmission requiring CI.

References

[1] R. Flury and R. Wattenhofer, "Slotted programming for sensor networks," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: ACM, 2010, pp. 24–34.

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

[3] D. Mills, "Internet time synchronization: the network time protocol," *Communications, IEEE Transactions on*, vol. 39, no. 10, pp. 1482–1493, Oct 1991.

[4] H. Li, X. Feng, S. Shi, F. Zheng, and X. Xie, "A high-accuracy clock synchronization method in distributed real-time system," in *Computer Engineering and Technology*, ser. Communications in Computer and Information Science, W. Xu, L. Xiao, J. Li, C. Zhang, and Z. Zhu, Eds. Springer Berlin Heidelberg, 2015, vol. 491, pp. 148–157.

[5] J. Elson and K. Römer, "Wireless sensor networks: A new regime for time synchronization," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 149–154, Jan. 2003.

[6] K. Römer, "Time synchronization in ad hoc networks," in *Proceedings of the 2Nd ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, ser. MobiHoc '01. New York, NY, USA: ACM, 2001, pp. 173–182.

[7] N. Burri, P. von Rickenbach, and R. Wattenhofer, "Dozer: Ultra-low power data gathering in sensor networks," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, April 2007, pp. 450–459.

[8] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Dec. 2002.

[9] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 138–149.

[10] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 39–49.

[11] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "Robust multi-hop time synchronization in sensor networks." in *International Conference on Wireless Networks*, 2004, pp. 454–460.

[12] J. Sallai, B. Kusý, Á. Lédeczi, and P. Dutta, "On the scalability of routing integrated time synchronization," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science, K. Römer, H. Karl, and F. Mattern, Eds. Springer Berlin Heidelberg, 2006, vol. 3868, pp. 115–131.

[13] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, "Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 1, no. 4, pp. 239–251, 2006.

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

[15] P.-H. Huang, M. Desai, X. Qiu, and B. Krishnamachari, "On the multihop performance of synchronization mechanisms in high propagation delay networks," *Computers, IEEE Transactions on*, vol. 58, no. 5, pp. 577–590, May 2009.

[16] J. Lu and K. Whitehouse, "Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks," in *INFOCOM 2009, IEEE*, April 2009, pp. 2491–2499.

[17] R. Gotzhein and T. Kuhn, "Black burst synchronization (bbs) - a protocol for deterministic tick and time synchronization in wireless networks," *Computer Networks*, vol. 55, no. 13, pp. 3015 – 3031, 2011.

[18] P. Dutta, R. Musaloiu-e, I. Stoica, and A. Terzis, "Wireless ack collisions not considered harmful," in *Proceedings of the 7th ACM Workshop on Hot Topics in Networks (HotNets-VII)*, 2008, pp. 1–6.

[19] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*. ACM, 2012, pp. 1–14.

[20] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2013, pp. 1–14.

[21] M. Doddavenkatappa and M. C. Chan, "P 3: A practical packet pipeline using synchronous transmissions for wireless sensor networks," in *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*. IEEE, 2014, pp. 203–214.

[22] D. Yuan and M. Hollick, "Let's talk together: Understanding concurrent transmission in wireless sensor networks," in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*. IEEE, 2013, pp. 219–227.

[23] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*. IEEE, 2005, pp. 364–369.

[24] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Information Processing in Sensor Networks (IPSN), 2013 ACM/IEEE International Conference on*. IEEE, 2013, pp. 153–165.

[25] *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, Texas Instruments, cC2420 Data Sheet.

[26] *IEEE Standard 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks*, Institute of Electrical and Electronics Engineers.

[27] T. Schmid, P. Dutta, and M. B. Srivastava, "High-resolution, low-power time synchronization an oxymoron no more," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: ACM, 2010, pp. 151–161.

Appendix

A. MAC Layer Timestamping

To synchronize the sending nodes' clocks, a reference node sends a couple of synchronization packets at the start of each round (see Section VII-A). Every packet contains the timestamp of the reference node's clock at the time of the packet's departure. For this we employ MAC layer timestamping [9], which makes use of some wireless transceivers providing a pin (the SFD pin for the CC2420) which goes active exactly when the transmission or reception of a packet begins. This pin is usually connected to a timer capture input pin at the microcontroller to allow obtaining an accurate timestamp of this event common to sender and receiver. Additionally, the transceiver needs to offer a way to edit a packet's contents after its transmission has already begun. The CC2420 allows editing its TXFIFO buffer in a non-FIFO fashion for this purpose.

Contiki's CC2420 driver already provided this feature, but only supported 16-bit timestamps from the 32 kHz quartz crystal. We extended the driver to support our mixed-source 64-bit timestamps. As mentioned above, assembling such a 64-bit timestamp from its individual components requires a fair amount of instructions on this architecture – a total of 55 in our case. Further, applying drift compensation requires the multiplication of two 64-bit integer variables (see Appendix B), which requires an additional few hundred instructions. As our synchronization packets were 60 bytes in length, which require roughly 2 milliseconds or 8500 CPU cycles to transmit, there is still ample time to compute this timestamp and insert it at the end of the packet.

B. Drift Compensation

Although the properties of quartz crystals are generally rock-solid, their frequencies are not completely set in stone: temperature changes affect their frequency and multiple instances of the same crystal may have slightly different frequencies. In a system with multiple such quartz crystals this leads to observable clock drift: clocks which were synchronized at some point in time may drift apart as a result of their ever so slightly varying speeds. Worse, these speeds change over time as the environment temperature changes.

To combat this, we implemented a drift compensation mechanism based on linear regression with a rolling buffer: For the last k received synchronization packets we store $(\mathit{local}_i, \mathit{offset}_i)$ pairs, where local_i is the local time at which the ith synchronization packet was received and $\mathit{offset}_i = \mathit{global}_i - \mathit{local}_i$ was the clock offset at that time. We then compute on the last k values:

$$\mathit{drift} = \frac{\overline{\mathit{local}_i \cdot \mathit{offset}_i} - \overline{\mathit{local}_i} \cdot \overline{\mathit{offset}_i}}{\overline{\mathit{local}_i^2} - \overline{\mathit{local}_i}^2}$$

$$\mathit{baseoffset} = \overline{\mathit{offset}_i} - \mathit{drift} \cdot \overline{\mathit{local}_i}$$

where $\overline{\exp}$ denotes the average of the expression $\exp$ for $i \in \{n, n-1, n-2, \ldots, n-k+1\}$ where n denotes the number of

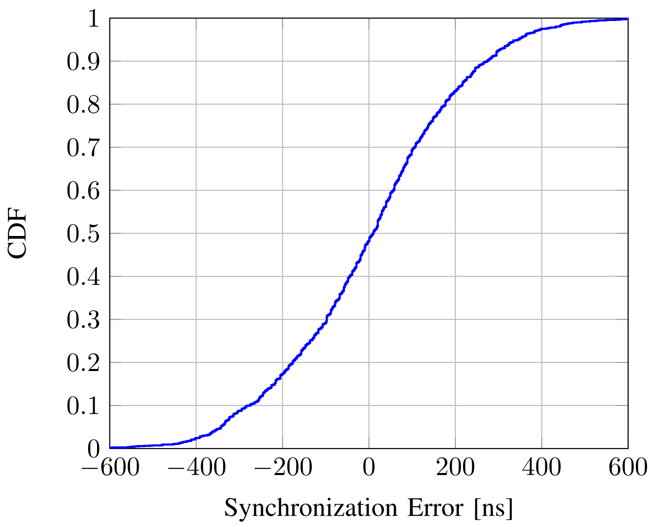


Fig. 5: The distribution of the clock synchronization error a single node is detecting while being re-synchronized every few seconds. For this plot the sample size was 1934.

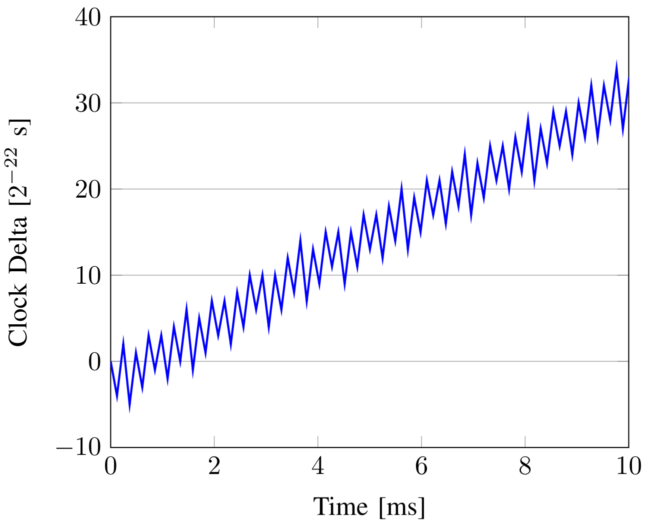
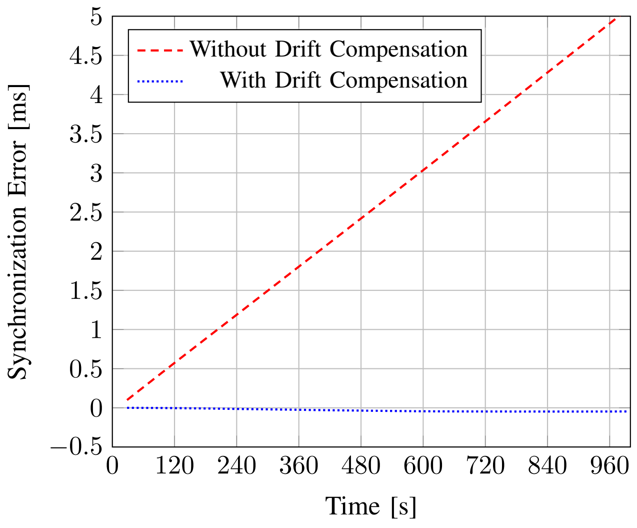


Fig. 6: The delta between the quartz crystal clock and the DCO clock over the span of 10 milliseconds, sampled at every fourth tick of the quartz crystal.



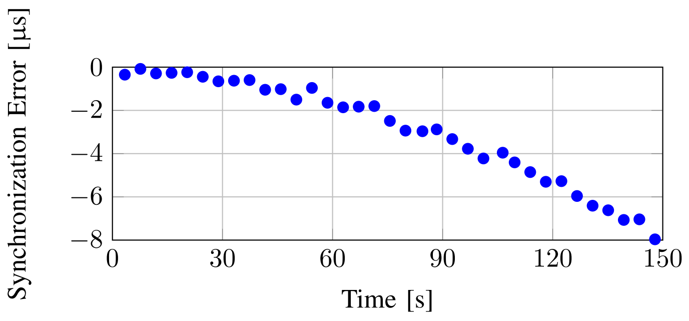


Fig. 7: Synchronization error of a single node over time.

received synchronization packets. To extrapolate an estimate of the global time we compute:

$$globalest(local) = local + baseoffset + drift \cdot local.$$

For the size of the rolling buffer we found $k = 8$ to be adequate, adapting to changes quickly enough, while avoiding wild fluctuations from noise and outliers. Other settings with less frequent synchronization rounds might find smaller values to suit their needs better.

While drift compensation is absolutely crucial in situations where nodes are not re-synchronized for longer periods of time, it can already offer substantial benefits after shorter periods when high precision is required. To be able to store and forward global timestamps computed using the formulas above while preserving the gained precision, we append 8 additional bits to our timestamps (see Figure 2).

Figure 5 shows the quality of the clock synchronization that we were able to achieve using both MAC layer timestamping and drift compensation. We observe most instances (over 70%) to be spread uniformly between -250 and $+250$ nanoseconds. We cannot hope to improve this by much since the timestamps we measure when sending or receiving a synchronization packet are only granular to $1/2^{22}$ seconds. Hence, each of these timestamps introduces an error distributed uniformly at random in the interval $[-1/2^{23}\text{seconds}, +1/2^{23}\text{seconds}]$ (roughly $[-120\text{ns}, +120\text{ns}]$).

Figure 7 shows the development of the synchronization error between two nodes which were synchronized a couple of times at the start (to allow for drift detection) and then not anymore. Clearly, not employing drift compensation when not re-synchronizing nodes over longer periods of time is fatal to the synchronization error. The almost constant slope shows why linear regression is the right tool to combat clock drift. We also see in this example that, in spite of drift compensation, the synchronization error exceeds 0.5 µs after about 25 seconds. Hence, when attempting simultaneous sending, the most recent synchronization should ideally not lie further than 10 or 20 seconds in the past.