Time-of-Flight Aware Time Synchronization for Wireless Embedded Systems

Roman Lim, Balz Maag, Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
{lim, bmaag, thiele}@tik.ee.ethz.ch

Abstract

Accurate time synchronization is an important prerequisite for many applications. Synchronization down to sub-microsecond precision, as required by distributed control in automation or network event analysis, is prevalently a domain of wired or expensive GPS-enabled systems. Existing time synchronization protocols for wireless embedded systems exhibit errors that are orders of magnitude higher. We identify propagation delay compensation as a key requirement to achieve sub-microsecond precision in typical deployments. As a result, we present the Time-of-Flight Aware Time Synchronization Protocol (*TATS*), a new protocol that combines fast multi-hop flooding and message delay compensation at similar message cost as existing delay-unaware protocols. Experiments conducted in a public testbed of 31 nodes show that *TATS* achieves sub-microsecond synchronization error over 22 hops, while outperforming state-of-the-art protocols by a factor of up to 6.9.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*; C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Design, Experimentation, Measurement, Algorithms

Keywords

Propagation delay compensation, time Synchronization, wireless embedded systems

1 Introduction

Time synchronization is an important primitive for distributed systems. Such systems have different requirements on the accuracy of time synchronization. For many applications an accuracy in the millisecond range is sufficient [4]. Other applications like distributed control in automation or distributed measurements, *e.g.*, for network event analysis [24, 17] or data acquisition during flight tests [18], require a higher degree of time synchronization in order to guarantee failure-free operation.

Distributed systems that require sub-microsecond time synchronization are generally synchronized either using satellite communication, such as the Global Positioning System (GPS), or wired infrastructure. Using GPS, it is possible to acquire very accurate timing, *i.e.*, average timing error of 60 ns on standard commodity L1-GPS receivers [29]. For small and spatially limited deployments, or for locations without satellite reception, wired approaches are an alternative. A prominent example is the precision time protocol [3], which can leverage existing Ethernet infrastructure. However, the cost for integrating a GPS receiver is high, both economically and power-wise, and wired solutions for places without exiting infrastructure have high initial cabling cost.

Wireless time synchronization protocols for embedded systems achieve synchronization errors in the millisecond to microsecond range using inexpensive hardware, *e.g.*, *PulseSync* has a worst case synchronization error of 19 μs in a 30-hop line topology [15], which is two orders of magnitude away from sub-microsecond accuracy. In addition, experimental evaluation of protocols is mostly conducted in a setting that does not resemble real deployments well, *i.e.*, nodes are arranged within close proximity and in radio range of a single reference broadcaster [15, 19]. This leads to small distances between nodes, and hence propagation delays are not considered in the evaluation of these protocols. It's therefore unclear how well these results apply for network structures with longer and more diverse distances between individual nodes.

Emerging embedded platforms for cyber-physical systems are more sophisticated than first generation wireless sensor network platforms, thereby providing higher clock rates and radio transceivers integrated into computation units. Examples of such integrated chips are the Texas Instruments CC430 or CC2538 series, combining an MSP430 microcontroller core with a sub-1 GHz radio transceiver or an ARM Cortex-M3 and an IEEE 802.15.4 compliant 2.4 GHz radio. Existing wireless time synchronization protocols can profit from this development in several ways. Faster system clocks result in higher time resolution of packet timestamps, while integration of MCU and radio core on one chip facili-

tates tighter control of the radio core.

In light of these developments, we revisit existing concepts with the aim to narrow the gap between wireless multi-hop time synchronization and its wired and GPS counterparts, and therefore bringing flexible and lower cost time synchronization to a wide set of applications that require sub-microsecond timing accuracy.

Challenges. To increase the coverage of a wireless network, nodes use intermittent hops to forward information to receivers that are not in communication range. When targeting sub-microsecond time accuracy, propagation delays are not negligible and need to be appropriately considered when exchanging time information. In addition, it has been shown that fast propagation of time information is essential, as error accumulation is proportional to the time spent in the network [14, 30]. Bringing both objectives together is a non-trivial task.

Contributions and road-map. We identify differing propagation delays as an important aspect to further increase the accuracy of current time synchronization protocols. Based on the obtained insight, we propose the Time-of-Flight Aware Time Synchronization Protocol (*TATS*), a new protocol that compensates propagation delays on communication paths. *TATS* builds on existing flooding based synchronization approaches and introduces propagation delay measurements *without* sending additional packets.

In summary, this paper makes the following contributions:

- In Sec. 3, we assess the impact of propagation delays on two recently proposed synchronization protocols, namely *Glossy* [6] and *PulseSync* [14]. We reveal a dependency between the minimal achievable global synchronization error and network topology, and thus motivate the need for propagation delay compensation.
- In Sec. 4, we design *TATS*, a multi-hop time-synchronization protocol that compensates the propagation delay experienced on communication paths with no additional packet complexity.
- We discuss implementation details of *TATS* on a recent hardware platform in Sec. 5.
- We evaluate *TATS* in Sec. 6 and compare its performance against *Glossy* and *PulseSync* on a public testbed with 31 nodes. To show the impact of network topology on the global synchronization error, we use three different topologies: a short and a long 22-hop line topology, and a dynamically built distribution tree. Overall, *TATS* achieves an average synchronization error of $0.24\,\mu$s and a maximal synchronization error of $0.54\,\mu$s, which is up to a factor of 6.9 better than its competition.

To the best of our knowledge, we are the first to report of a *sub-microsecond* synchronization error over *tens* of hops using off-the-shelf wireless embedded nodes.

2 Related Work

This section summarizes publications closely related to the proposed *TATS* protocol. For an exhaustive survey on time synchronization protocols, the reader is referred to [27].

Wireless sensor networks. Table 1 gives an overview on synchronization accuracies reported for time synchronization protocols running on typical sensor node platforms. Direct comparison by numbers is not possible because evaluations are conducted under different circumstances. In addition, accuracies are sometimes reported as mean absolute error, *i.e.*, the unsigned deviation, and sometimes as mean signed deviation. The latter generally results in lower values.

TPSN [7] employs a two-way message exchange to measure the delay introduced by the communication stack. Although such measurements include physical propagation delays of electromagnetic waves, other delays, *e.g.*, jitter in radio interrupts, dominate the measurements. Different to our approach, TPSN creates a fixed hierarchical network structure and two messages are exchanged *per link* for a two-way delay measurement and time synchronization. In contrast, *TATS* builds on a dynamically built flooding tree and only one broadcast message per node to perform the same measurements.

By employing more sophisticated MAC-layer timestamping [19] or capture-registers, the measured delay between sending and receiving a packet has a significantly narrower distribution and can be approximated by a constant value. This enables time synchronization using only unidirectional communication, *e.g.*, *FTSP* [19] *RATS* [12] and *PulseSync* [15] synchronize a network by flooding time information using broadcast messages. Communication patterns are less complex because flooding does not need a sophisticated routing tree. Different to *TATS*, performance of these protocols depends on the choice of a message delay calibration and on the distribution of propagation delays between individual nodes.

Glossy [6], a flooding architecture for wireless sensor networks based on concurrent transmissions implicitly provides network-wide synchronization. As such, common reference times can be computed on every node of the network. In contrast to *TATS*, *Glossy* does not foresee propagation delay compensation or drift compensation.

TATS builds on various concepts introduced by other time synchronization protocols. We use linear regression, as introduced by RBS [5], to compute the offset and the speed of the local clock relative to a reference clock. Same as *RATS* [12] and *PulseSync* [15], *TATS* coordinates packet transmissions in order to achieve fast information propagation over several hops. By doing so, clock drift on forwarding nodes have a lower impact on time synchronization error. In [25], the idea of high resolution, low-power clocks is introduced, which reduces the synchronization error. To accurately timestamp radio packets, *TATS* relies on a high frequency clock.

In summary, propagation delay has been treated as a negligible source of synchronization error in wireless sensor networks. In contrast to the mentioned work, we show that propagation delay plays a major role for sub-microsecond time synchronization accuracy and we propose *TATS*, a new time synchronization protocol that compensates for different propagation delays per link. *TATS* brings together the simplicity of unidirectional network flooding and the propagation delay awareness of two-way message exchange

Table 1. Reported synchronization errors. *If not mentioned otherwise, error values are reported as magnitudes.*

	Hops	Synch. Interval	Avg	Max	Platform
TPSN	1	-[b]	16.9 μs	44 μs	Mica
FTSP	6	30 s	2.3 μs	14 μs	Mica2
FTSP[a]	1	10 s	0.13 μs [c]	n/a	Epic
RATS	11	30 s	2.7 μs	26 μs	Mica2
PulseSync	30	10 s	2.06 μs	19 μs	Opal
Glossy	8	-[b]	0.4 μs[c]	n/a	TelosB

[a] using virtual high resolution timer [25]
[b] single measurements, no linear regression
[c] mean signed deviation, not mean absolute error

schemes.

High latency acoustic networks. While the need for compensation of propagation delays in air has been mostly neglected, it is more prominent in underwater networks [8]. In water, acoustic waves are used for communication. Compared to RF communication, the speed of acoustic waves is five orders of magnitudes slower. Due to different environmental influences such as multi-path effects or time dependent propagation characteristics, it is unclear whether such synchronization protocols can be directly applied to RF based communication. In addition, complex messaging hinders fast dissemination of time information over several hops. TSHL [28] employs a two-phase approach to first estimate the local speed of the clock and then use a two-way message exchange to measure the propagation delay.

3 Impact of Propagation Delay

In this section we motivate that it is important to take propagation delay into account when designing a time synchronization protocol. We use the term *propagation delay* to refer to the duration a signal travels between the antennas of two communication partners. First, we quantify the propagation delay in wireless embedded systems and compare it against other errors present when synchronizing time in multi-hop networks. Then, we analyze the impact on *PulseSync* and *Glossy*, two state-of-the-art time synchronization protocols that treat propagation delay as a negligible quantity.

3.1 Time-of-Flight vs. Other Sources of Error

In order to assess the impact of propagation delays in wireless embedded systems, we put the error into perspective. Radio communication, based on electromagnetic waves, propagates at the speed of light, *i.e.*, approximately 3×10^8 m/s. The indoor communication range of a typical wireless sensor node is $20 - 30$ m [20]. Accordingly, a message traveling between two nodes can experience a propagation delay of up to 100 ns.

To put this value into perspective, we consider time synchronization accuracies reported in literature as listed in Table 1. Although one can not directly compare the protocols, as they are evaluated on different hardware platforms and using different settings and algorithms, we get a good picture of accuracies currently attained. As stated by [14], synchronization error in a multi-hop network is a function of the network diameter. The more hops involved, the worse the accuracy. Depending on the algorithm, the error grows exponentially, linearly, or sub-linearly ($\sqrt{\text{diameter}}$) with the network diameter [14]. To approximate the error introduced by each hop, we divide the average error by the number of hops. The lowest value for protocols in Table 1 results from the *PulseSync* protocol: $2.06\,\mu\text{s}/30 = 67$ ns. Except for *TPSN*, all the listed protocols treat propagation delay as being constant.

We conclude that the impact of varying propagation delays is comparable to the error introduced by other effects like jitter when timestamping a packet or clock drifts between nodes. For outdoor deployments, the effect of propagation might be even more severe, as communication ranges are larger. Some deployments exhibit node distances of several hundreds to thousands of meters, *e.g.*, on bridges [11] or in alpine environments [2].

3.2 Existing Multi-hop Time Synchronization Protocols

To assess the potential of propagation delay compensation, we investigate two recent protocols that are representative for the current state-of-the-art for time synchronization in wireless sensor networks. We identify shortcomings that prevent better accuracy just by increasing the frequency with which nodes re-synchronize. In the following, we use the term *message delay* to refer to the time between timestamping a message on the sender and the receiver. This delay also includes the propagation delay.

PulseSync [15] builds on the insight that it is beneficial to forward time information as fast as possible through the network. To do that, the protocol floods *pulses* through the network. Each node sends exactly one message within each pulse after having received the message from its predecessor. The initiating reference node embeds its current clock value into the message. All forwarding nodes update the time value by adding the message delay and the dwell time of the message. Here, the dwell time is the difference of the local clock values taken at receive-time and at send-time. The message delay for all pair-wise links is assumed to have the same normal distribution with a known mean value, *i.e.*, the "differences in radio propagation times can be neglected in sensor networks" [14]. The message delay is determined during a calibration phase.

Every message received serves as a sample point that relates the reference time to a local clock value. The slope and the offset of each node's local clock is then calculated using least squares linear regression over the last k sample points. *PulseSync* implements an optional drift compensation to reduce the error that is added by updating the time information in the packet on each node. This error stems from measuring the dwell time using local clocks that run at a slightly different speeds than other nodes.

To estimate the impact of calibrating the protocol with a single propagation delay parameter τ_c, we assume in the following a network where packets are perfectly timestamped (no jitter) with an arbitrarily accurate time resolution. Nodes have perfect clocks without drift and the message delay is

equal to the propagation delay. Let us denote the number of hops a packet in a pulse travels from the reference node to node v as h_v, and the real accumulated propagation delay on this path from the reference node to v as τ_v. The error that results from imperfect knowledge of the propagation delay at this node is $h_v\tau_c - \tau_v$. The resulting global synchronization error $\mathcal{G}$, *i.e.*, the maximal pairwise error across all nodes in the network is

$$\mathcal{G} = \max_v(h_v\tau_c - \tau_v) - \min_v(h_v\tau_c - \tau_v). \quad (1)$$

We see that the resulting global synchronization error heavily depends on the network topology, *i.e.*, h_v and τ_v. An optimal parameter τ_c that minimizes $\mathcal{G}$ can be found using linear programming. In general, it is difficult to find the optimal τ_c as this requires knowledge of all possible paths of the flood and the respective path delays. In addition, network structures change over time, *e.g.*, due to mobility or changes in the environment, necessitating an adaption of τ_c. A necessary condition for the error to vanish completely is that all path delays τ_v are multiples of τ_c, which is very unlikely to be the case in a real wireless sensor network deployment.

Glossy [6] is a flooding mechanism based on concurrent transmissions that allows to disseminate messages in a multihop network as fast as possible. To synchronize time, an *initiator node* starts a flood and embeds its clock value into the first packet. Every node that receives a packet immediately retransmits it, thereby effectively synchronizing packets sent in the same slot. With every transmission, a counter value c contained in the packet is incremented by one. With this information it is possible to calculate an estimate of the start time of the flood as

$$\hat{t_{ref}} = T_{c_0} - c_0 t_{slot}, \quad (2)$$

where T_{c_0} is the local time of the first received packet and c_0 the counter value contained in this packet. The propagation delay is contained in the t_{slot} value, as this is the interval between the start of a packet transmission with relay counter c and the start of the following packet transmission with relay counter c+1. Nodes estimate t_{slot} locally using packet timestamps. In [6], the authors assume that "t_{slot} is a network-wide constant, since during a flood nodes never alter the packet length".

There are two aspects where propagation delay plays a role: (*i*) the timestamp T_{c_0} is affected by different propagation delays, leading to a similar effect as for *PulseSync* if a constant propagation delay is assumed for the whole network; (*ii*) the slot time t_{slot} is strictly speaking not a network-wide constant, but rather depends on the immediate neighborhood of a node. To show this, we conducted following experiment. We let *Glossy* run in a setup as shown in Fig. 1: three nodes are directly connected using wires and a signal splitter to enforce fixed communication channels. Communication channel (I)-(F) experiences a longer delay channel than (I)-(N). Node (I) starts a flood, while (F) and (N) participate in the flood. After receiving (I)'s message, (F) and (N) are transmitting the message concurrently. As (N)'s signal is received stronger at (I), due to capture effect [13], the packet sent by (F) has no impact on the timing at (I). After a while, we turn off (N). The acquired slot estimates on individual nodes are shown in Table 2. When all nodes are participating in the flood, the estimates are similar. However, as we turn off the closer node (N), slot estimates become larger by approximately 150 ns. Such variations in t_{slot} have a large impact on the calculated reference time (2) because they are multiplied by the number of hops c_0.

Table 2. Slot times estimated during *Glossy* floods.

	Initiator (I)	Node (F)	Node (N)
All nodes	516.78 μs	516.78 μs	516.78 μs
Without (N)	516.93 μs	516.95 μs	-

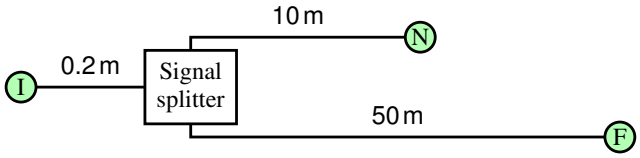


Figure 1. Experimental setup to show the influence of the capture effect on slot length measurements. *The antenna connectors of three nodes are wired to a signal splitter, resulting in different signal delays due to differences in cable length.*

3.3 The Need For Propagation Delay Compensation

Varying propagation delays can introduce per-hop errors as high as 100 ns for indoor deployments and are therefore relevant when aiming for sub-microsecond synchronization accuracy. State-of-the-art time synchronization protocols handle errors well that stem from clock drift and message delay jitter, by providing a fast flooding mechanism or combining several measurement points using linear regression, but lack the awareness for propagation delays.

4 Time-of-Flight Aware Time Synchronization

In this section, we describe *TATS*, our new protocol that combines per-link message delay compensation and fast flooding for highly accurate time synchronization. As seen in Sec. 3, synchronization accuracy suffers from unknown propagation delay between nodes. Therefore we want to compensate for varying propagation delays between nodes in a network, while keeping the advantages of state-of-the-art protocols, namely high synchronization accuracy due to fast dissemination, and low overhead due to flooding. Furthermore, the number of additional messages needed should be minimal.

We decompose propagation delay compensation on a link into two steps: (*i*) estimating the delay on a link, and (*ii*) updating the time value contained in a message by adding the delay estimate. The message delay on a link can be estimated using *two-way delay* measurements, as depicted in Fig. 2, also applied by TPSN [7]. Two messages are exchanged per link: node 0 sends a packet to node 1 and remembers the timestamp T_0. Upon reception, node 1 replies with a packet that contains the dwell time ω_1, which is then used by node

0 to compute the two-way delay as $R_{1\to0} - T_0 - \omega_1$, where $R_{1\to0}$ is the reception time of the packet at node 0. The one-way delay is computed by dividing the two-way delay by two.

Adding *low overhead* message delay compensation to existing flooding based protocols is challenging for three reasons:

1. In contrast to flooding, where every node sends just one broadcast packet, the two-way delay measurement involves two packets *per link* and adds therefore considerable overhead.
2. After a message exchange, only the initiating node (node 0 in Fig. 2) knows the delay. As floods are based on broadcasts and it is therefore unknown who will receive the packet, the delay estimate has to be compensated by the receiving node(s), hence the propagation delay knowledge is needed at the receiving node 1.
3. Two-way delay measurements are only feasible if links are bidirectional. Flooding does not have this restriction and therefore might use links for which message delays are not obtainable. Unidirectional links are very common in real deployments, *e.g.*, [22] reports of a testbed where 46 % of the links are unidirectional.

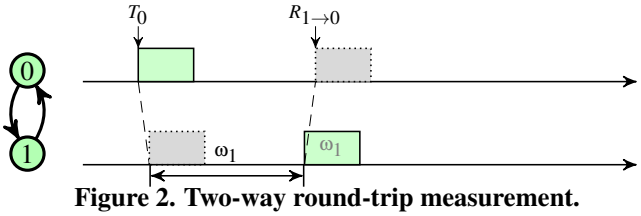


Figure 2. Two-way round-trip measurement.

Next, in Sec. 4.1, we give an overview of our approach. In Sec. 4.2, we describe our method to measure message delays using broadcast packets, and finally, we introduce a heuristic that makes our protocol more resilient to non-symmetric links in Sec. 4.3.

4.1 Overview

The aim of *TATS* is to establish a global time that is synchronized to a reference node on all nodes in the network. The network is assumed to be short term stable, *i.e.*, the mean propagation delay between nodes only changes slowly over time. This assumption is reasonable for static networks. We use *PulseSync* [15] as a starting point and extend it with a propagation delay compensation: Similar as in *PulseSync*, messages containing the reference time are periodically flooded to all nodes, initiated by the reference node. Each node participates in the flood by (*i*) reading the reference time (*ii*) adding the message delay to it and, (*iii*) on transmission, adding the dwell time to the reference time. All communications are broadcasts and nodes transmit once for every flood, after a random and short timeout after receiving a packet. Each flood implicitly creates a routing tree, thereby defining a parent-child relation between nodes.

Received delay-compensated reference times are stored in a table together with the corresponding local reception times of packets. A node then performs a least squares linear regression on these value pairs to calculate the time offset and clock drift of the local clock relative to the reference clock.

Different to *PulseSync*, *TATS* applies individual message delays for each link in step (*ii*). Two-way message delays are measured by piggy-backing additional information onto regular synchronization packets. In this way, we add propagation delay compensation to *PulseSync without* sending additional packets. Next, we detail our approach for propagation delay measurements.

4.2 Propagation Delay Estimation

Let's consider Fig. 3, which depicts a small part of a network consisting of three nodes. All links are bi-directional, *i.e.*, communication is possible in both directions. Node 0 starts a flood by sending a broadcast packet containing the reference time. This packet is received by nodes 1 and 2. After a random timeout, each receiver updates and forwards the packet. As all transmissions are broadcasts, node 0 overhears the forwarded packets. This chain of actions resembles the same information flow as is needed to carry out a round-trip time measurement, as shown in Fig. 2. To use this information flow for two-way measurements, messages need to contain the dwell time ω and the identifier of the parent. The latter is needed because communication is based on broadcast packets. Without that information, node 0 could not distinguish between messages of its child nodes and messages of other nodes. The one-way message delay between nodes v and w after flood k is computed as

$$\delta^k_{v\leftrightarrow w} = \frac{R_{w\to v} - T_v - \omega_w}{2}. \tag{3}$$

T_v and $R_{w\to v}$ are the timestamps taken at node v when node v sent its message and when it received a message from node w. The dwell time on w is denoted as ω_w. Because propagation delays are short term stable, a more accurate delay estimate $\bar{\delta}^k_{v\leftrightarrow w}$ can be obtained by averaging N consecutive measurements:

$$\bar{\delta}^k_{v\leftrightarrow w} = \frac{1}{N}\sum_{i=k-N+1}^{k} \delta^i_{v\leftrightarrow w} \tag{4}$$

In this way, parents can obtain message delay estimates for all links towards all their children. Every node keeps a number of most recent delay measurements in a table. As stated in Sec. 3, the estimates are needed on child nodes to compensate for propagation delays. To inform child nodes about message delays, parents embed the obtained average message delay into the time synchronization packet. The resulting packet format of *TATS* is shown in Table 3. A parent needs to forward estimates to potentially many children. As only a limited number of estimates fit into a synchronization message, parents select estimates to send in a round-robin fashion.

For every received time synchronization message in a flood, a child node w compensates the message delay by looking up the value belonging to the link $v \to w$ and adds that to the received reference time $G_{v\to w}$ to obtain the compensated reference time G_w:

Table 3. Structure of synchronization packets.

Name	Description
Sequence number[a]	Sequence number of flood
Reference time G^a	Global time
Node ID of parent	Parent ID in this flood
Dwell time ω	Elapsed time between receiving and sending
Node ID of measurement	Identifies the link of the measurement
Message delay $\bar{\delta}$	Average message delay measured by this node

[a]Same as in *PulseSync*

$$G_w = G_{v\to w} + \bar{\delta}_{v\leftrightarrow w} \tag{5}$$

The proposed mechanism does not prevent collisions, *e.g.*, node 1 and node 2 in Fig. 3 could potentially send their packets at the same time during a flood, therefore render it impossible to perform a delay measurement. As timeouts are random, eventually, in a consecutive flood, a measurement will be possible. Because message delays are short term stable, a certain delay in measuring and distributing estimates is permitted and does not hamper the performance.

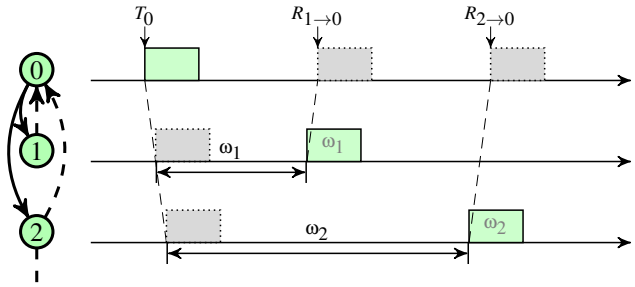


Figure 3. Round-trip measurements are based on time information embedded into time synchronization packets. *A parent node 0 can acquire several measurements by listening to the packets that are transmitted by its children 1 and 2.*

4.3 Avoiding Unidirectional Links

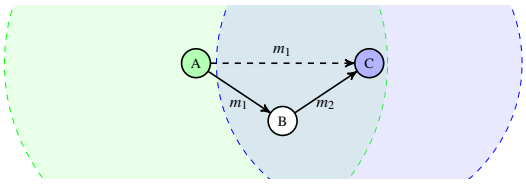


Figure 4. Unidirectional links prevent round-trip measurements. *By introducing a short delay, intermediate nodes get the chance to forward the time information over bidirectional links.*

The proposed mechanism for round-trip measurements requires that links can be used in both ways, otherwise message delay measurements are not possible. Unidirectional links manifest when the received signal power is close to the receiver's sensitivity threshold. In the situation illustrated in Fig. 4, node C can hear node A, but communication in the opposite direction is impossible. Therefore, the propagation delay between A and C can't be estimated. If there is an additional node B with bidirectional links to both A and C, route A→B→C would allow for round-trip measurements and consequently propagation delay compensation could be applied. We observe that in Fig. 4, node C will eventually receive a message directly from A (m_1) and another one relayed over B (m_2). By ignoring the earlier message m_1, we establish the desired route.

TATS exploits this observation to reduce the number of unidirectional links used in a flood. Every time a packet is received over a link with unknown message delay, an additional waiting period is introduced before forwarding the message. If a messages from a neighbor with known message delay arrives during this period, the earlier message is ignored.

Our evaluation in Sec. 6.2 shows that this heuristic results in more round-trip measurements and less missing delay estimates.

5 Implementation

We implement *TATS* in Contiki OS [1] on a CC430 developer board to show its feasibility and to benchmark the performance.

Hardware platform. We use the Olimex MSP430-CCRF developer board (16.16€ per piece) [21] as hardware platform for our implementation. This board features a low-power Texas Instruments CC430F5137 SoC, providing 32 kB of program memory and 4 kB of RAM. The chip integrates an MSP430 core and a CC1101 sub-1 GHz radio with configurable bit rate and radio modulation. The on-board printed PCB-antenna is used. A 26 MHz quartz oscillator provides the basis of a stable 13 MHz system and timer clock. The quartz has a nominal frequency deviation of ±10 ppm and a temperature dependent deviation of ±10 ppm over the specified range from -25 to 75°C. System time is stored in a 16-bit counter value and extended, on overflow, by incrementing an additional integer variable to a 64-bit timestamp.

Message timestamps. For propagation delay measurements, timestamps are taken on the sending and on the receiving nodes. Packet based radios like the CC1101 generate interrupts when a synchronization symbol is detected. These interrupts occur both on the sender and on the receiver. The chain of events involved in message timestamping is shown in Fig. 5. As soon as the sender has transmitted the synchronization symbol, an interrupt signal is generated. At the next rising edge of the sender's clock, the value of the timer register is stored as the timestamp T. On reception of the synchronization symbol, the receiver stores its timestamp R in a similar way.

Message timestamps are affected by jitter, which is caused by asynchronously running digital clocks and conversion between digital and analog domain when generating

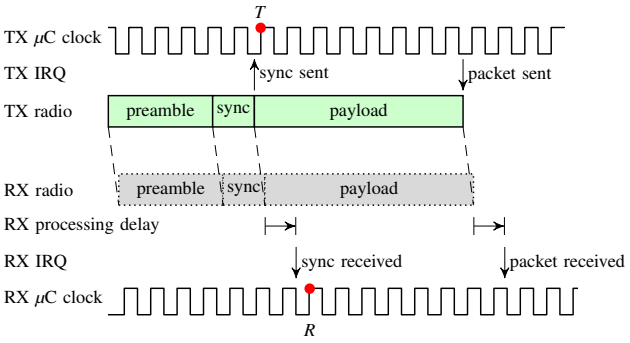


Figure 5. Timestamps for one message transmission. *Timestamps T and R are inaccurate due to asynchronous clocks and uncertainties introduced with radio modulation.*

or decoding the radio signal. The smaller the jitter, the more accurate the resulting time synchronization. Therefore a fast clock is beneficial for synchronization.

We configure the radio to use GFSK modulation and a data rate of 250 kbit/s. The distribution of the message delay over a short distance, experimentally measured using two nodes on a desk and an external logic analyzer, is shown in Fig. 6. The delay is normally distributed with a mean value of 13.68 μs and a standard deviation of 107 ns. Compared to other hardware platforms, this is a relatively low value (see Table 4). Lower jitter should potentially lead to lower synchronization error, provided the clock resolution for timestamps is sufficiently high. In our case, we can rely on a 77 ns clock resolution. We use the capability of the MSP430 to capture time values with dedicated capture registers, thus avoiding software interrupt delays when taking timestamps.

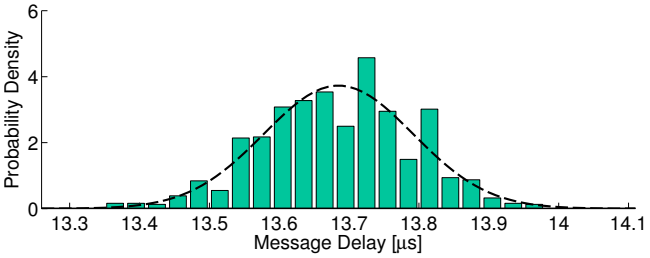


Figure 6. Distribution of message delay for the CC430 radio on a single link. *The dashed curve is a fitted normal distribution with a mean value of 13.68 μs and a standard deviation of 107 ns.*

Table 4. Standard deviations for message delays on different platforms.

Platform	Standard deviation
TelosB [23]	41 ns [25]
Opal (AT86RF231) [10]	180 ns [15]
Mica2 [9]	1.95 μs [26]
RF230 radio	370 ns [25]
MSP430-CCRF	107 ns

6 Evaluation

In this section we evaluate *TATS* in a public testbed with 31 nodes. As *TATS* does not employ explicit two-way round-trip measurements, we will evaluate how quickly delays are measured by parents and forwarded to child nodes. In a second experiment, we do a head-to-head comparison of *TATS* against *PulseSync* and *Glossy* in different network structures. Our experiments reveal the following key findings:

- *TATS* quickly acquires message delay estimates solely based on network flooding.
- Despite unidirectional links, the acquired delay estimates allow to compensate propagation delay on 95 % of all involved links.
- In a 22-hop line topology *TATS* performs up to 6.9 $\times$ better than *PulseSync* with respect to the average maximal synchronization error and 3 $\times$ better than *Glossy* and *PulseSync* on a shorter dynamic topology.
- In all settings, *TATS*'s maximal synchronization error is clearly below 1 microsecond.

6.1 Experimental Setup

A common method to evaluate the synchronization accuracy of a protocol on real hardware is to put all the sensor nodes into a single broadcast domain and enforce a logical topology in software, *i.e.*, only certain links are allowed to be used for communication. The accuracy is then measured by letting all nodes capture the time of a commonly received packet [15, 19]. Using a message as common reference is not possible in our case as this message is also affected by propagation delays and would therefore reach different nodes at different time instances. Moreover, the setting does not resemble well a real deployment, where nodes are scattered over a large area.

Therefore we choose a more realistic approach by letting nodes actually form a real multi-hop network. We run our tests on FlockLab [16], a public testbed where 31 nodes are spread over an area of 75 $\times$ 35 meters in an office environment and also outdoors. The detailed layout of the testbed is shown in Fig. 7. We have 6 nodes equipped with GPS receivers that generate an accurate reference pulse, *i.e.*, a digital signal that has a low-high transition every second. This pulse is then connected to a GPIO pin of a node and timestamped using capture registers. The computed global timestamp of this event is then used to calculate the synchronization error. The employed LEA-6T GPS receivers provide timing accuracy with a root-mean-square error of 30 ns [29].

For all experiments, we use a transmission power of 10 dBm and a radio frequency of 870 MHz.

6.2 Propagation of Message Delay Estimates

This experiment evaluates the feasibility of message delay measurements without additional packets, only based on flooding. In *TATS*, two-way delays are measured by parent nodes and then forwarded to child nodes. As described in Sec. 4.3, unidirectional links prevent two-way delay measurements, therefore *TATS* introduces a strategy to circumvent unidirectional links. If a synchronization packet is received over a link with unknown propagation delay, nodes wait for an additional waiting period for a synchronization

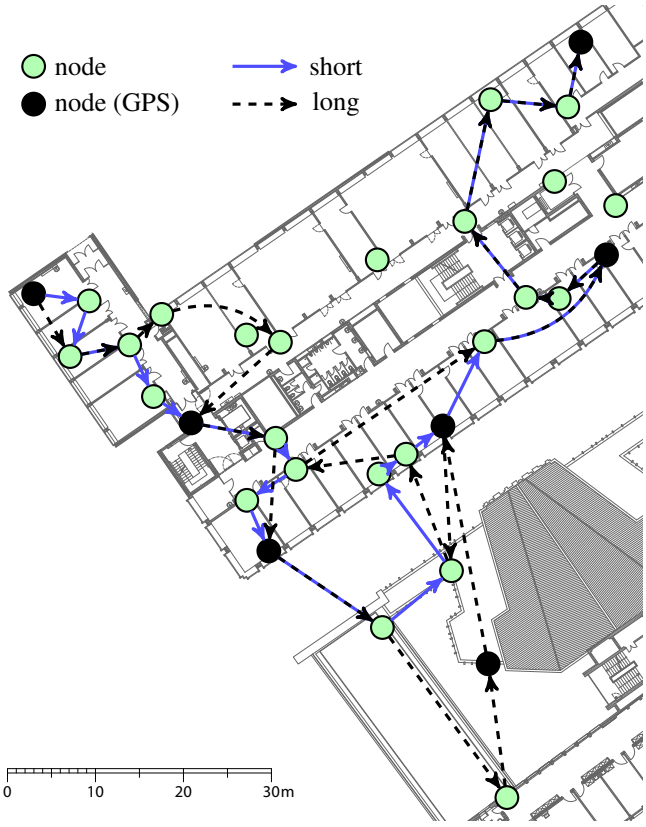


Figure 7. Testbed layout and software enforced path for the *short* and *long* line topology. *Nodes in bold are equipped with a high precision GPS receiver that generates a synchronized reference pulse. In the* dynamic *topology, all nodes participate.*

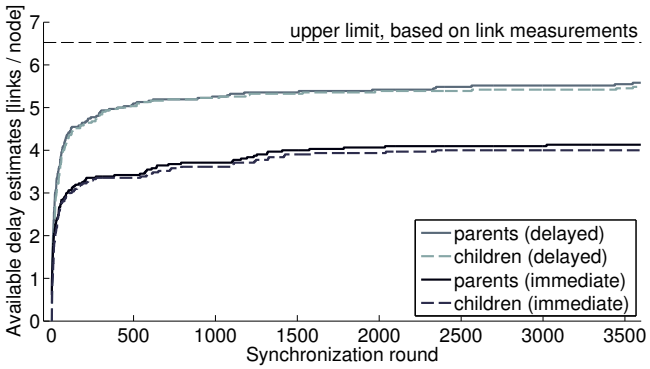


Figure 8. Available link delay estimates, on parents and on child nodes for *TATS* using immediate and delayed forwarding. *Measurements propagate quickly from parents to child nodes. Delayed forwarding achieves 37 % more measurements and covers 85.5 % of all possible links.*

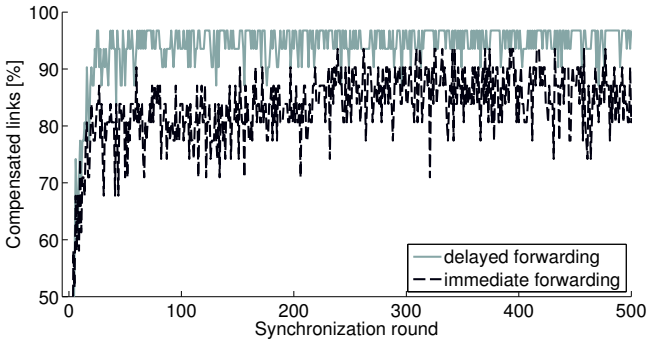


Figure 9. Compensated links during floods. *Both protocol variants quickly acquire delay measurements for relevant links, while delayed forwarding has more coverage and stabilizes at a level of 95 %.*

packet on a link with known propagation delay. In this experiment, we quantify the impact of this strategy. We run *TATS* once without additional timeout (*immediate forwarding*) and once with a waiting period of 10 ms (*delayed forwarding*).

Setup. For each configuration, we let the synchronization protocol run for one hour on all nodes in the testbed. The GPS node in the upper left corner in Fig. 7 is used as reference node. We configure *TATS* to have a synchronization period of 1 s. For every synchronization round, nodes report the number of message delays measured (as parent) and the number of delay measurements received (as child node). In addition, we count the number of unknown message delays when updating the global time in forwarded messages. From a regular link measurement on FlockLab [16], we extract the number of available communication links to put our experiment into perspective. On average, we see 101 links between 31 nodes, 29.8 % are mostly unidirectional.

Results. Fig. 8 shows the average number of estimated link delays per node, while Fig. 9 presents the ratio of links that are compensated when running *TATS*. The latter uses a subset of all estimated link delays, *i.e.*, those links that are part of the flooding tree. For both variants, measurements propagate quickly from parents to child nodes, which results in a very small difference in available delay estimates between parents and children. We find that delayed forwarding is beneficial and achieves 37 % more estimated links than immediate forwarding. Delayed forwarding leads to an increased coverage of links and also to less missing estimates while forwarding packets.

This experiment confirms the usefulness of delayed forwarding and shows that it is feasible to perform two-way delay measurements based on network flooding, even in the presence of unidirectional links.

6.3 Comparison against *PulseSync* and *Glossy*

In this experiment, we compare *TATS* against *PulseSync* and *Glossy*. To assess the synchronization accuracy, the *global synchronization error* is an important metric, *i.e.*, the maximal pairwise difference between clock values of all node in the network. Technically this is not possible with our setup, as we would need a GPS receiver next to every node. A representative coverage of the network is attained by placing the GPS receivers evenly distributed. Instead of the global synchronization error, we measure the synchronization error relative to the reference node

$$\mathcal{G}_r = \max_{v \in V}(|t_r - t_v|). \tag{6}$$

Here, the set V contains all GPS nodes except the reference node r. The clock values t_r and t_v are the locally calculated times when the GPS pulses arrived at the respective nodes. For each test run, we report the average and the maximum synchronization error $\mathcal{G}_r$ over the duration of the test.

Setup. We measure the accuracy in three different settings: a dynamically formed network and two different 22-hop line topologies. The dynamic network has a diameter of approximately 6 hops. Line topologies are enforced in software as shown in Fig. 7: line topology *short* has a length of 182 m, while line topology *long* has a length of 283 m. This way, we evaluate the impact of different propagation delays and different network structures on *TATS*'s synchronization accuracy.

PulseSync is calibrated using a single message delay parameter [15]. We averaged a total of 2014 measurements between two nodes to estimate this parameter. As we forward packets as fast as possible, we implement both *PulseSync* and *TATS* without drift compensation, as the effect of drift would be marginal. In case of larger clock drifts between nodes, caused *e.g.*, by large temperature differences or varying manufacturing processes, drift could be compensated as described in [15].

For a fair comparison, we perform the same linear regression as in *TATS* also on *Glossy* nodes. As enforcing a real 22-hop line topology is not possible for concurrent transmissions, we compare *Glossy* only on the dynamic topology.

We configure all three protocols to use a synchronization interval of 1 s and a regression table of 80 samples. In total, seven different test runs are performed, each possible combination of protocol and topology once for a duration of one hour.

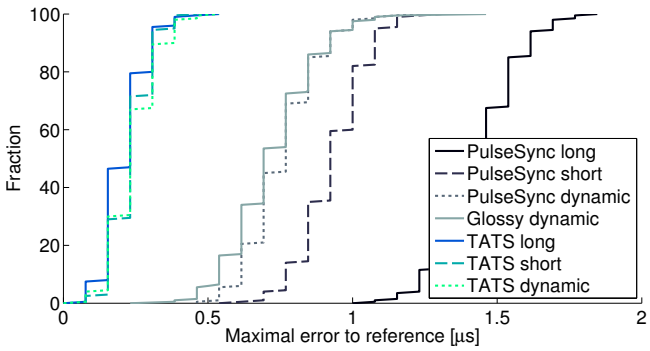


Figure 10. Cumulative distribution of synchronization errors, measured relative to the reference node. *While PulseSync performs different on the three topologies, TATS can adapt and compensate for different propagation delays.*

Results. Fig. 10 shows the distribution of the maximal absolute time difference to the reference node over all synchronization rounds. The error distribution is stable for *TATS* on all three topologies, while *PulseSync* exhibits very differing performance. In addition, the error is significantly higher than the one of *TATS*. Both effects can be attributed to the fact that a single point calibration of message delay can't sufficiently represent the conditions in the whole network. *Glossy* exhibits a similar performance as *PulseSync* on the dynamic topology. Fig. 11 shows the evolution over time. The error settles for all protocols after only a few synchronization rounds.

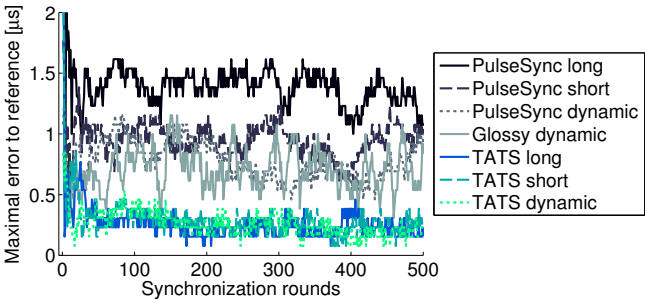
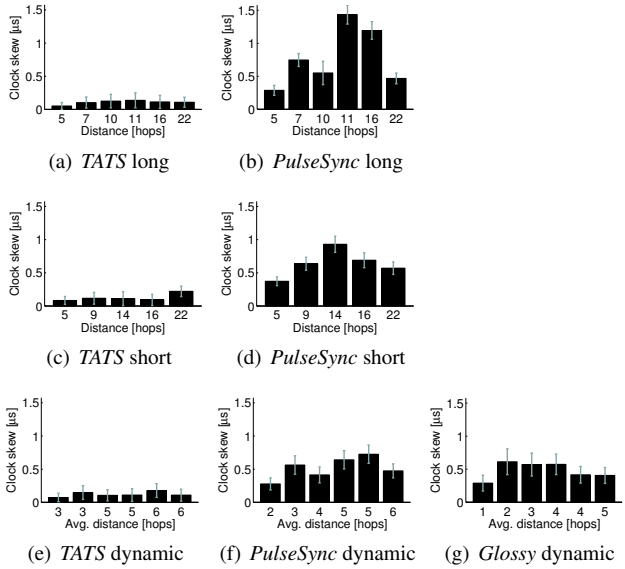


Figure 11. Maximal synchronization error over time. *For better visibility, only the first 500 rounds are shown.*

Table 5. Accuracies measured for different protocols.

Setting	avg error	max error	PRR
TATS long	0.21 μs	0.54 μs	[0.96..1.00]
TATS short	0.23 μs	0.46 μs	[0.99..1.00]
TATS dynamic	0.24 μs	0.54 μs	[0.90..1.00]
PulseSync long	1.43 μs	1.85 μs	[0.98..1.00]
PulseSync short	0.93 μs	1.31 μs	[0.99..1.00]
PulseSync dynamic	0.75 μs	1.23 μs	[0.96..1.00]
Glossy dynamic	0.72 μs	1.46 μs	[1.00..1.00]



(a) *TATS* long (b) *PulseSync* long
(c) *TATS* short (d) *PulseSync* short
(e) *TATS* dynamic (f) *PulseSync* dynamic (g) *Glossy* dynamic

Figure 12. Average synchronization errors over time per node. Bars indicate standard deviation.

If we consider the average error of individual nodes in Fig. 12, we see that the error is evenly distributed for *TATS* on all topologies (a), (c), (e), while nodes running *PulseSync* are affected by more variance (b), (d), (f). The distribution for *Glossy* (g) is similar to *PulseSync* (f).

Key figures for all test runs are summarized in Table 5. *TATS* performs up to 6.9 $\times$ better than *PulseSync* with respect to average maximal synchronization error and 3 $\times$ better than

Glossy and *PulseSync* on a shorter dynamic topology. In all settings, *TATS*'s maximal synchronization error is below 1 microsecond. The lowest packet reception rate (PRR), *i.e.*, the ratio between sent and received synchronization packets, is 0.9 over all test runs. Missing synchronization packets potentially lead to reduced accuracy, as less sampling points for the linear regression are available.

7 Conclusion

We have presented *TATS*, a new and highly precise time synchronization protocol for wireless embedded systems. *TATS* combines fast flooding and message delay compensation at similar message cost as existing protocols without delay compensation. Experiments on a testbed that resembles real deployment scenarios well with respect to node distances show that (*i*) *TATS* achieves up to 6.9 $\times$ better accuracy than state-of-the-art protocols, and (*ii*) can synchronize even networks with large diameters of up to 22 hops within sub-microsecond accuracy. This makes time synchronization using wireless sensor networks a viable option to wired or GPS-based high precision systems.

Acknowledgements

We thank Marco Zimmerling, Olga Saukh, and Jan Beutel for their valuable input. The work presented in this paper was scientifically evaluated by the SNSF, and financed by the Swiss Confederation and by nano-tera.ch.

8 References

[1] The Contiki operating system. `http://www.sics.se/contiki/`.

[2] The Permasense project. `http://www.permasense.ch`.

[3] IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages c1–269, July 2008.

[4] K. Chebrolu, B. Raman, N. Mishra, P. K. Valiveti, and R. Kumar. Brimon: A sensor network system for railway bridge monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2008.

[5] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[6] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, 2011.

[7] 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 (SenSys)*, 2003.

[8] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li. Research challenges and applications for underwater sensor networking. In *Wireless Communications and Networking Conference (WCNC). IEEE*, volume 1, 2006.

[9] J. L. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6), 2002.

[10] R. Jurdak et al. Opal: A multiradio platform for high throughput wireless sensor networks. *IEEE Embedded Systems Letters*, 3, 2011.

[11] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, 2007.

[12] 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. *Int. J. Ad Hoc Ubiquitous Comput.*, 1(4):239–251, July 2006.

[13] K. Leentvaar and J. Flint. The capture effect in FM receivers. *Communications, IEEE Transactions on*, 24(5):531–539, May 1976.

[14] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal clock synchronization in networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.

[15] C. Lenzen, P. Sommer, and R. Wattenhofer. PulseSync: An efficient and scalable clock synchronization protocol. *ACM/IEEE Transactions on Networking (TON)*, Mar 2014.

[16] 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 *Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN)*, 2013.

[17] R. Lim, B. Maag, B. Dissler, J. Beutel, and L. Thiele. A testbed for fine-grained tracing of time sensitive behavior in wireless sensor networks. In *Proceedings of the 40th Conference on Local Computer Networks Workshops (LCN Workshops)*, 2015.

[18] H. Mach, E. Grim, O. Holmeide, and C. Calley. PTP enabled network for flight test data acquisition and recording. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS)*, 2007.

[19] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[20] MEMSIC. *TelosB Mote Platform*, 2011. Rev A.

[21] OLIMEX Ltd. *MSP430-CCRF development board: User's manual*, 2013. Revision C.

[22] J. Ortiz and D. Culler. Multichannel reliability assessment in real world WSNs. In *Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN)*, 2010.

[23] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN)*, 2005.

[24] J. Robert, J.-P. Georges, T. Divoux, P. Miramont, and B. Rmili. On the observability in switched Ethernet networks in the next generation of space launchers: Problem, challenges and recommendations. In *Proceedings of the 7th International Conference on Advances in Satellite and Space Communications (SPACOMM)*, 2015.

[25] T. Schmid, P. Dutta, and M. B. Srivastava. High-resolution, low-power time synchronization an oxymoron no more. In *Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN)*, 2010.

[26] P. Sommer and R. Wattenhofer. Gradient clock synchronization in wireless sensor networks. In *Proceedings of the 8th International Conference on Information Processing in Sensor Networks (IPSN)*, 2009.

[27] A. R. Swain and R. Hansdah. A model for the classification and survey of clock synchronization protocols in WSNs. *Ad Hoc Netw.*, 27(C):219–241, Apr. 2015.

[28] A. A. Syed, J. S. Heidemann, et al. Time synchronization for high latency acoustic networks. In *Proceedings of the 25th Conference on Computer Communications (INFOCOM)*, 2006.

[29] u-blox. *LEA-6 data sheet*, 2014. R10.

[30] Z. Zhong, P. Chen, and T. He. On-demand time synchronization with predictable accuracy. In *Proceedings of the 30th International Conference on Computer Communications (INFOCOM)*, 2011.