Keep the Beat: On-The-Fly Clock Offset Compensation for Synchronous Transmissions in Low-Power Networks

Martina Brachmann
Adaptive Dynamic Systems
TU Dresden
Dresden, Germany
martina.brachmann@tu-dresden.de

Olaf Landsiedel
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
olafl@chalmers.se

Silvia Santini
Faculty of Informatics
Università della Svizzera italiana (USI)
Lugano, Switzerland
silvia.santini@usi.ch

***Abstract*—Emerging protocols for low-power wireless networks increasingly exploit constructive interference and the capture effect. The basic idea is that the synchronous transmission of identical packets by neighboring nodes leads to constructive interference – or at least do not cause destructive interference. This requires that the temporal displacement of packets at receiving nodes is lower than 0.5 μs when employing IEEE 802.15.4 radios. However, commonly used sensor nodes are equipped with cheap and imprecise clocks that show high frequency deviations across nodes, making constructive interference difficult to achieve. Such deviations further increase when individual nodes are exposed to different temperatures. In this paper we introduce Flock, a novel approach to compensate for differences in clock frequency across synchronously transmitting nodes. We implemented Flock in Contiki on the example of Glossy, a flooding protocol based on synchronous transmissions. Our results confirm that Flock can achieve constructive interference on real sensor nodes in over 98% of the cases. Overall, Flock makes protocols that exploit synchronous transmissions more robust to operate even in challenging environments.**

***Index Terms*—Synchronous transmission, clock offset compensation, constructive interference, low-power networks**

I. Introduction

In recent years, protocols for low-power wireless networking increasingly build on constructive interference (CI) and the capture effect. Dutta et al. pioneered this research direction by introducing A-MAC – an efficient, receiver-initiated Medium Access Control (MAC) protocol [1]. In A-MAC, multiple receivers concurrently send identical acknowledgments for a single data packet. Glossy [2] builds upon this idea and lets nodes synchronously transmit the same data packet. Tight synchronization ensures non-destructive interference of the transmissions and allows receivers to decode a packet. Glossy can flood a packet in a multi-hop network within few milliseconds. It thereby achieves very high reliability and also provides time synchronization with microseconds accuracy.

A-MAC and Glossy sparked a new research direction in low-power wireless communication protocols that exploit so-called concurrent or synchronous transmissions. Recently presented approaches – such as LWB [3], Splash [4], Choco [5], and others – all build upon Glossy and use individual network floods to support communication primitives like data collection. Virtus [6] also builds upon Glossy to provide virtual synchrony [7] and Crystal [8] relies on data prediction to reduce the number of Glossy floods. CXFS [9], Sparkle [10], LaneFlood [11], and others [12], [13], [14], [15] limit the number of concurrent transmitters in Glossy or LWB to improve energy efficiency.

To ensure that concurrently transmitted packets do not interfere destructively, two conditions must be fulfilled: (i) nodes must transmit the same packet, and (ii) the transmissions need to be tightly timed to ensure that their symbols overlap sufficiently. For example, in IEEE 802.15.4, the synchronization of transmissions within at least 0.5 μs is necessary to prevent packets from interfering destructively [2]. Glossy demonstrated that it is possible to fulfill this "0.5 μs condition" by accounting for interrupt latency and ensuring that there is a fixed number of microcontroller (MCU) cycles between the end of the reception of a packet and the triggering of its retransmission. This number is set to roughly one hundred MCU cycles in the standard implementation of Glossy [2].

In this paper, we show that due to differences in clock frequency across different nodes, even the execution of just these one hundred clock cycles can result in a violation of the "0.5 μs condition". This is due to the fact that the MCUs of common low-power hardware platforms rely on unstable digital crystal oscillators (DCOs). DCOs are hard to calibrate, show strong drifts over time and even larger drifts when the temperature changes. Recent work shows that errors of up to 20% [2], [16] are common in real-world scenarios. This problem is further exacerbated when a larger delay between the reception and retransmission of a packet occurs, for example to allow for data processing as in Chaos [17] or to switch channels to increase resilience [18], [19], [20].

This paper introduces a novel approach to address these challenges: Flock – On-the-Fly Clock Offset Compensation. Instead of relying on the unstable DCO clock only, Flock also exploits the fine-grained, highly accurate clock driving the radio. This clock runs at 8 MHz and drives packet transmissions and base-band modulation in the radio chip. It has a low-error of at most ±40 ppm, as required by the IEEE 802.15.4 standard [21]. Its stability ensures that when a packet of known length is sent, its duration is very predictable (up to an error of few nanoseconds). Flock, thus, timestamps the start and the end of each packet reception. Next, it correlates these two timestamps with the local DCO time. Receiving a packet of 9 bytes should for instance require 1210 DCO cycles. The difference between this value and the actual number of DCO cycles it took to receive the packet gives a measure of the DCO frequency offset. Flock can then compensate on-the-fly for this offset and thus, makes it possible to fulfill the "0.5 μs condition" even in challenging environments.

This paper makes three main contributions: (1) We introduce Flock, a new approach to estimate and compensate on-the-fly the frequency deviations of the free-running DCOs – common on today's sensor nodes – utilizing the accurate and high-resolution radio clock; (2) We employ Flock on the example of Glossy to enable synchronous transmissions even in the presence of strong temperature changes or other adversary conditions; and (3) We evaluate Flock both in simulation and lab experiments on real sensor nodes and show its effectiveness in several real scenarios.

The remainder of this paper is structured as follows. Sec. II provides the required background on synchronous transmissions, Glossy and DCO-related issues. Next, Sec. III introduces the design of Flock. We discuss the evaluation of Flock in Sec. IV and discuss related work in Sec. V. Sec. VI concludes the paper and discusses future work.

II. Background and Motivation

Before introducing the design of Flock, this section provides the required fundamental notions necessary to understand the remainder of the paper. Furthermore, we provide a quantitative example to motivate our work.

A. *Enabling synchronous transmissions with Glossy*

The work of Glossy by Ferrari et al. [2], published in 2011, has spawned an entire new family of communication protocols utilizing so-called synchronous transmissions. These exploit the basic idea that the synchronous transmission of identical packets by neighboring nodes leads to constructive interference – or at least non-destructive interference. To achieve this, the temporal displacement between transmissions at a common receiver must be lower than a given threshold. For IEEE 802.15.4-compliant radios this threshold is known to be 0.5 μs [2]. When neighboring nodes transmit packets concurrently, they must do so within the bounds of 0.5 μs.

If nodes were synchronized at sub-microseconds scale, it would be possible to coordinate these transmissions using the common time reference. The vast majority of existing synchronization protocols for low-power networks, however, are only able to achieve synchronization accuracy of the order of microseconds [22]. This is not sufficient to ensure constructive interference. Recently presented synchronization mechanisms can achieve sub-microseconds accuracy [23], [24]. Their ability to support synchronous transmissions is

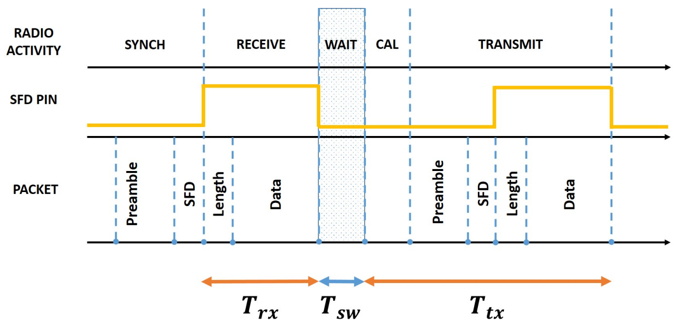


Fig. 1. Timeline of the radio activity during the reception and retransmission of a packet in Glossy. The time to receive a packet T_{rx}, the software delay T_{sw} and the time to transmit a packet T_{tx} are highlighted at the bottom of the image. (This illustration is a slightly modified version of Fig. 7 in [2]).

however still limited. The approach by König et al., for instance, can achieve synchronous transmissions in slightly more than 30% of the cases only [23].

Instead of relying on explicit synchronization, Glossy makes nodes implicitly align their transmissions by relying on radio events only. Glossy implements a flooding primitive that propagates a packet quickly throughout the network. The flood is started by a dedicated node – called the *initiator* – that sends the packet to propagate. All one-hop neighbors of the initiator receive this packet at almost exactly the same time instant. This is because time differences due to propagation delays are negligible (of the order of nanoseconds) in practical cases. In Glossy these nodes then retransmit the packet immediately to ensure that the time interval elapsed between the reception and the retransmission of the packet is as short as possible. Furthermore, Glossy eliminates almost entirely the sources of randomness that can influence the length of this time interval. This way, it ensures that nodes retransmit an incoming packet within a delay that is (almost) equal on all nodes and thus it manages to fulfill the "0.5 μs condition". The flood terminates once all nodes have retransmitted the packet for a pre-defined number of times (e.g., 3 times).

B. *The software delay in Glossy*

To ensure that the time needed to retransmit a packet after receiving it is the same on all nodes, Glossy must deal with the existence of the so-called *software delay* T_{sw} [2]. Fig. 1 helps explaining what the software delay is by depicting the main activities of the radio during the reception and retransmission of a packet in Glossy.

The physical layer header of an IEEE 802.15.4 packet contains a *Preamble* followed by a *Start of Frame Delimiter* (SFD)-byte. Once the SFD is received, a corresponding pin in the radio chip is set, as shown by the "SFD pin" line in Fig. 1. The SFD pin remains set until the entire packet, consisting of *Length* and *Data* fields, is received.

When the reception of the *Data* field is completed the SFD pin returns unset. The changing SFD signal triggers an interrupt at the MCU indicating that the reception of the incoming packet is completed. In the corresponding interrupt service routine the main goal of Glossy is to quickly trigger the retransmission of the received packet. The time needed by the MCU to do so is called the software delay T_{sw} [2]. Glossy ensures that the MCU executes exactly the same number of instructions – corresponding to 97 clock cycles – on all nodes during T_{sw}. With this setting, the (theoretical) maximal difference between the value of T_{sw} on different nodes is 0.375 μs [2], which is sufficient to ensure the "0.5 μs condition" to be fulfilled.

This, however, only holds when the frequency of the clock that sources the MCU is essentially equal on all synchronously transmitting nodes. In practical settings, however, these clocks may and do run at a different frequency on different nodes. In the following section and in Sec. IV we show that this in turn results in non-negligible differences in the actual software delay.

C. How the MCU clock frequency affects the software delay

The MCU in low-power hardware platforms is often driven by an unstable digital crystal oscillator (DCO). While the DCO is expected to tick at a pre-specified, nominal frequency, the actual DCO frequency may deviate significantly from the nominal value. This deviation occurs despite the fact that the frequency of the DCO is calibrated at frequent, pre-defined intervals via an external, stable 32 kHz crystal.

For the MSP430 – the MCU typically found on the widely used TelosB platform [25] – the nominal frequency of the DCO corresponds to 4,194,304 Hz [26]. The actual frequency may, however, vary significantly from node to node due to manufacturing issues or depending on the internal temperature or supply voltage of the node [26], [16]. In particular, the sensitivity of the DCO to temperature is quantified in $-0.38\%/^{\circ}$C [26]. This implies that if two nodes are exposed to a temperature difference of, e.g., 20°C, the difference between the frequency at which their MCUs run will differ of 7.6 percentage points. Thus, while the clock of one node can be assumed to run at 4,194,304 Hz (i.e., $\sim$238 ns per tick), the other node would have a clock running at 3,875,537 Hz (i.e., $\sim$258 ns per tick). While 97 ticks at the first frequency correspond to 23.13 μs, they instead result in 25.03 μs if the second frequency is considered. The difference between these two delays is far higher than the maximum allowed temporal displacement of 0.5 μs, which is required to ensure constructive interference to occur.

We introduce Flock to compensate for these differences in the DCO clock frequency. Flock contributes in making Glossy, and many other protocols building upon it, more robust to operate in challenging environments, like those described in [16]. Furthermore, Flock allows to cope with a further downside of Glossy's approach to ensure synchronous transmissions, i.e., the fact that it does not allow for intermediate operations to be executed between the reception and retransmission of a packet. Indeed, Glossy minimizes the number of instructions executed during the software delay to mitigate "*the impact of DCO instability*" [2]. Thus, approaches that need to process an incoming packet or delay its retransmission can adopt Flock to compensate for DCO clock frequency misalignments and thus lift an inherent limitation of Glossy's design.

III. Design

As illustrated in the previous section the actual duration of the software delay in Glossy may differ across different nodes because the frequency of the DCO may deviate from the nominal frequency in an unpredictable way. This causes a *clock offset* to exist between nodes, which Flock can compensate for.

Flock does not cause any additional communication among nodes and its principle of operation is rather simple. In a nutshell, it adapts on-the-fly the number of MCU clock cycles that must elapse during T_{sw}. While in Glossy this number is fixed to 97, Flock varies it on each node individually to account for deviations of the MCU clock frequency from the nominal value.

A. How Flock works

To determine the actual number of MCU clock cycles that must elapse during T_{sw} on each node, Flock must obtain a reliable estimation of the actual DCO frequency – or at least of its deviation from the nominal value. To this end, it needs a reliable time reference. The key intuition behind the design of Flock is to use the time between the transitions of the SFD pin during the reception of a packet, indicated as T_{rx} in Fig. 1, as such a time reference. Indeed, the time T_{rx} needed by the radio to receive a packet has two crucial characteristics: (i) it is known, and (ii) it can be assumed to have a stable value across different nodes.

The length of T_{rx} is known because it depends only on the length of the packet, which can be fixed a priori or be read from the packet itself in each round (see field *Length* in Fig. 1). The length of T_{rx} is stable because it depends on the clock of the radio only. This clock is typically highly accurate and more fine-grained than the DCO clock. In particular, the IEEE 802.15.4 standard requires that this clock can drift at a rate of at most $\pm$40 ppm [21].

If Glossy runs on a platform that relies on the CC2420 radio chip [27] – like our reference platform, the TelosB mote, does – then T_{rx} is equal to 288 μs. This is because the CC2420 transmits at 250 kbit/s, the *Data* field of a Glossy packet is of 8-byte length, and T_{rx} is the time needed to receive both the 1-byte long *Length* field and the *Data* field. A total of 9 bytes at 250 kbit/s thus results in 288 μs[1]. Assuming that the clock of the radio suffers from the maximum drift of $\pm$40 ppm, the resulting error on the value of T_{rx} would be of $\pm$11.52 ns (288 μs $\cdot$ $\pm 40 \cdot 10^{-6}$). Thus, we can assume the value of T_{rx} to be the same – up to few nanoseconds – across all nodes, which is a factor 23 below the required bound of 0.5 μs for synchronous transmissions. Moreover, considering a drift of $\pm$40 ppm for the radio crystal is a worst case scenario. The crystal needs to fulfill this requirement over its complete lifetime and operation parameters including voltage

[1]Our measurements – reported in Section IV – show that the value of T_{rx} is indeed very stable but in reality amounts to 288.6 μs. This must be considered when the value of T_{rx} is used but has no effect on the design of Flock.

and temperature changes. Thus, in practice, a much smaller error than ±40 ppm can be expected, as also confirmed by our experiments. Furthermore, we observe that the DCO frequency can be assumed to remain stable in the interval $T_{rx}+T_{sw}$, thus eliminating a further potential source of error.

Let us now indicate with C_{rx} the number of clock cycles that would elapse during T_{rx}, if the DCO would run at the nominal frequency f_{DCO}. Accordingly, C^*_{rx} indicates the corresponding number of DCO clock ticks when the DCO frequency is the actual one, indicated as f^*_{DCO}. Let us further assume that I indicates the number of clock cycles that must elapse during T_{sw} under the assumption that the DCO frequency is f_{DCO}.

If the three values defined above were known, the number of clock cycles I^* that must elapse during T_{sw} when the DCO frequency is f^*_{DCO} can be determined through the equation:

$$I^* = \lfloor \frac{C^*_{rx}}{C_{rx}} \cdot I \rceil \tag{1}$$

The function $\lfloor \rceil$ in Eq. 1 indicates the nearest integer function (i.e., the right side of the equation must be rounded to the closest integer). Using Eq. 1, Flock can compute the desired value of I^* without the need of knowing the actual value of f^*_{DCO}. We detail below how Flock determines the values of the three factors on the right side of Eq. 1.

B. Computing C^*_{rx}, C_{rx}, and I

We determine the value of C^*_{rx} by letting the MCU count the number of MCU clock ticks that elapse during T_{rx}. This can be easily implemented on the TelosB platform because the MCU supports the time-stamping of transitions of the SFD pin. Flock captures the two timestamps corresponding to the instances at which the MCU detects when the SFD pin rises and subsequently drops. The difference between these two timestamps corresponds to C^*_{rx}. Using this procedure, the MCU always returns an integer value.

Fig. 2(a) helps illustrating how the value of C^*_{rx} is actually determined on the MCU of a TelosB mote. The rising of the SFD pin is detected by the MCU with a variable and unpredictable delay. This is due to the fact that the SFD pin is set at the rising edge of the clock of the radio, while the MCU detects it at the rising edge of its own clock. Since these two clocks run at different frequency, the delay mentioned above originates. We indicate the ratio between this delay and the length of an MCU clock cycle as the factor k_p. This same parameter is defined in Glossy as "*the fraction of the DCO period [...] required at the MCU to sample the SFD transition*" and it is a uniformly distributed random variable between 0 and 1.

Depending on the value of k_p, the actual number of clock cycles counted by the MCU during T_{rx} may differ. Furthermore, the MCU detects the transition of the falling SFD (reception end) on the next rising edge after the actual transition. This is due to the fact that the MCU verifies the status of the SFD pin (and sets timestamps) only in correspondence of the rising edge of its clock. This causes

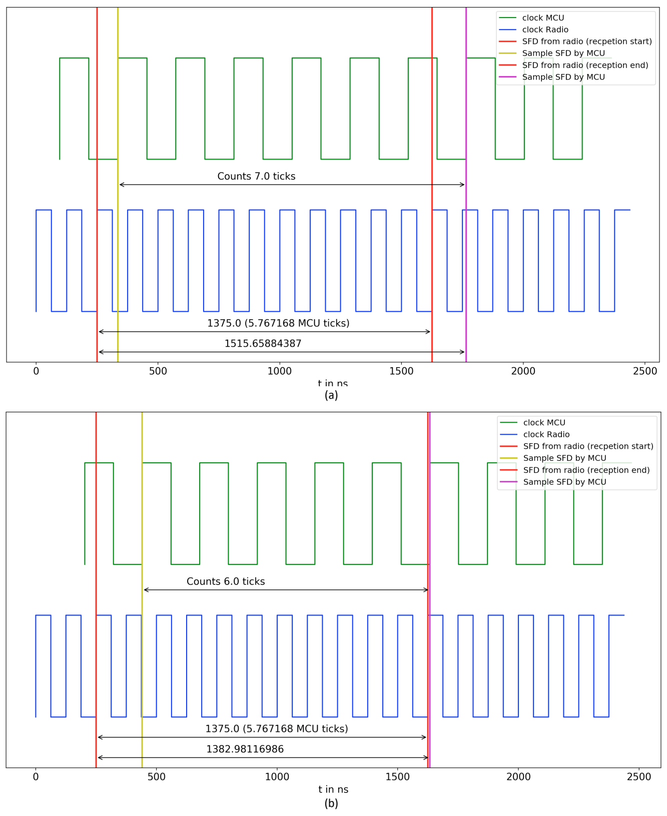


Fig. 2. The MCU clock driven by the DCO and the radio clock run at different frequency. This introduces a delay between the time at which a SFD pin transition is signalled by the radio (at the rising edge of the radio clock) and the time instant at which the MCU detects this transition (at the rising edge of the MCU clock). The length of this delay varies between 0 and the length of one DCO clock cycle and affects the number of clock cycles counted by the MCU to elapse during T_{rx}.

the clock tick count to be further increased by 1. The issues become visible when Fig. 2(a) and Fig. 2(b) are compared. In the first case, the MCU counts 6 clock ticks during T_{rx} and an additional tick after the SFD transition. Thus, the MCU counts 7 ticks in total. In the second case, it counts in total 6 ticks only. Depending on the value of k_p, the MCU may thus count either 6 or 7 ticks for T_{rx}. Using the timestamping procedure described above and assuming a nominal frequency f_{DCO}, we can compute C_{rx} as follows:

$$C_{rx} = \lceil (T_{rx} \cdot f_{DCO}) + k_p \rceil + 1. \tag{2}$$

Depending on the values of k_p, T_{rx}, and f_{DCO}, C_{rx} may in general take one of several values (e.g., 6 or 7 as in the example above). Flock selects the value with higher probability C^{high}_{rx} as the one used to compute I^* in Eq. 1. Because k_p is random and uniformly distributed between 0 and 1, we can calculate the probability of the bigger value C^{max}_{rx} (e.g., 7 in our example) with

$$P(C^{max}_{rx}) = (T_{rx} \cdot f_{DCO}) - \lfloor (T_{rx} \cdot f_{DCO}) \rfloor \tag{3}$$

Depending on the occurrence probability of C^{max}_{rx}, we can determine C_{rx}^{high} as follows:

$$C_{rx}^{high} = \begin{cases} \lceil (T_{rx} \cdot f_{DCO}) \rceil + 1 & \text{if } P(C_{rx}^{max}) < 0.5 \\ \lceil (T_{rx} \cdot f_{DCO}) \rceil + 2 & \text{if } P(C_{rx}^{max}) \geq 0.5 \end{cases} \quad (4)$$

The last parameter from Eq. 1 to describe is I. As mentioned in Sec. II-C, Flock allows for intermediate operations to be executed during T_{sw}. This requires the application designer to set the value of I.

C. Statistical uncertainties on the software delay

The authors of Glossy showed that – even in the absence of clock offsets – the value of the software delay T_{sw} is not deterministic and depends on the frequency of the radio clock f_r [2]. In particular, they demonstrate that "*T_{sw} is a discrete random variable with granularity* $1/f_r$" and that this is due to the fact that the radio clock and the DCO are not synchronized. To understand how Flock copes with this uncertainty, we remind that while Glossy uses a constant number of software instructions I to make the software delay as deterministic as possible, Flock changes the number of software instructions I^* at runtime to mitigate the clock offset.

If there is no DCO offset, I^* and I have the same value. In this case, C_{rx} can assume one or two values. If the value $T_{rx} \cdot f_{DCO}$ in an integer number, C_{rx} assumes a single value with probability $P(C_{rx}^{high}) = 1$. Otherwise, C_{rx} assumes one of two possible values that occur with different probabilities. Eq. 4 allows us to compute the value of C_{rx} that occurs with higher probability. The other value can however still occur with probability $1 - P(C_{rx}^{high})$. The possible values for C_{rx} differ by 1 DCO tick. Assuming that $C_{rx}^{high} = C_{rx}^{max}$ then C_{rx}^{*} becomes $C_{rx} - 1$. After transformation, Eq. 1 becomes $I^* \cdot C_{rx} = I \cdot (C_{rx} - 1)$. This latter equality holds if $I^* < I$, which however contradicts our initial assumption that I^* and I have the same value. The resulting uncertainty for I^* is $\frac{I}{C_{rx}}$ DCO ticks. As a consequence, the uncertainty increases with large I. Increasing C_{rx} by increasing the packet size mitigates these effects. Thus, large packets increase the probability that the "0.5 μs condition" is fulfilled even with large processing delays between the reception and re-transmissions of a packet.

Glossy sets $I = 97$ to achieve "*the theoretical lower bound of only two possible values for T_{sw}*" [2]. Flock achieves a lower bound of two values only when the product $T_{rx} \cdot f_{DCO}$ results in an integer value. Otherwise, the theoretical lower bound is three possible values for T_{sw} with granularity $1/f_r$, due to Flock's additional uncertainty of $\frac{I}{C_{rx}}$ DCO ticks for I^*.

IV. Evaluation

To demonstrate the performance of Flock, we first present simulation results that show the distribution of T_{sw} with and without Flock for different packet sizes and software delays. Afterwards, we report results from real sensor nodes and show how Flock performs under different conditions.

A. Flock in simulations

We first simulate the distribution of T_{sw} with and without Flock. Our results confirm that the small amount of software instructions in Glossy is still affected by DCO frequency offsets. We further show that Flock efficiently compensates for frequency deviations of the DCO even with large T_{sw}. A larger T_{sw} allows processing operations between packet reception and retransmission. We use Eq. (3) from [2] for calculating T_{sw}. We set I = 97 as in Glossy and vary the DCO frequency to simulate a deviation from the nominal value between 0% and 7.6%. As mentioned in Sec. II, a temperature difference of 20°C corresponds to a DCO frequency difference of 7.6%.

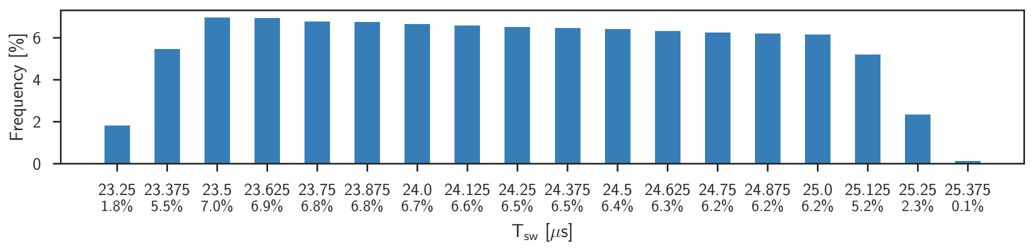
Fig. 3a shows the resulting distribution of T_{sw} for 10^6 samples for Glossy without Flock. We find that T_{sw} spreads over $\Delta 2.125$ μs, distributed over 18 values with a distance of 125 ns to each other. As mentioned in Sec. III-C is T_{sw} randomly distributed with granularity of $1/f_r$. The radio clock runs with 8 MHz, resulting in a resolution of 125 ns. Our simulation result confirms that the small amount of software instructions that Glossy requires are still affected by frequency deviations. With a constant number of instructions I, the temporal displacement can be far above the required 0.5 μs for constructive interference. We now apply Flock. We consider a packet size of 9 bytes and simulate the distribution of T_{sw} again. As depicted in Fig. 3b, T_{sw} distributes now only over $\Delta 0.625$ μs. Further, over 99% of the values are within 0.375 μs, which is sufficient to ensure constructive interference.

We now evaluate the influence of the packet length and the number of instructions I on Flock. We set the packet length to 128 bytes while I is still set to 97 and simulate again for 10^6 samples. Fig. 3c illustrates that the packet length in this configuration only has marginal effects on the distribution of T_{sw}. This is expected as $(\frac{I}{C_{rx}} = \frac{97}{1210}) \ll 1$. In a next step, we set I to 2000 ticks, which is the default value for the processing time in Chaos [17] and set the packet length to 9 bytes. Fig. 3d shows a larger distribution of T_{sw}. The calculated uncertainty for I^* is around 2 ticks, resulting in a larger distribution of T_{sw}. Constructive interference can be achieved in 87% of the transmissions. In order to verify that larger packets mitigate the effect of the uncertainty, we set the packet length to 128 byte. Fig. 3e confirms that the distribution of T_{sw} reduces to $\Delta 0.5$ μs, fulfilling the "0.5 μs condition" for constructive interference. Fig. 3f shows the distribution of T_{sw} without Flock for I = 97 for reference. The values spread over $\Delta 39.5$ μs.

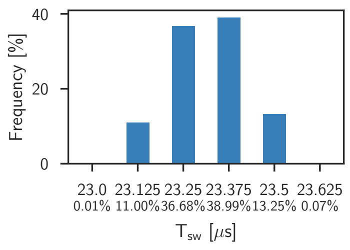
B. The effect of Flock on the software delay

We now evaluate the effect of Flock on T_{sw} in a controlled environment in our lab. We find that Flock can achieve constructive interference in 98% of the cases, even in challenging environments. The variable delay of T_{sw} is reduced from $\Delta 2.124$ μs without Flock to maximum $\Delta 0.75$ μs when enabling Flock. Thus, Flock makes synchronously transmitted packets overlap with high probability.

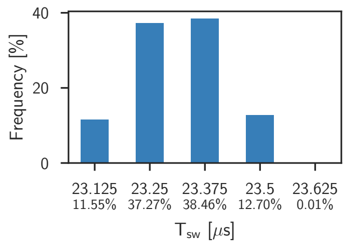
We implement Flock in the publicly available source code of Glossy and run it on TelosB motes. In order to reach I^*, nodes



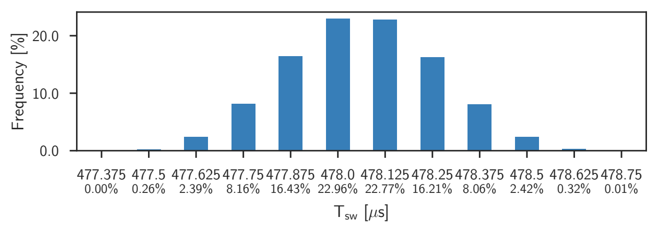
(a) Without Flock, I = 97, packet length = 9 byte.



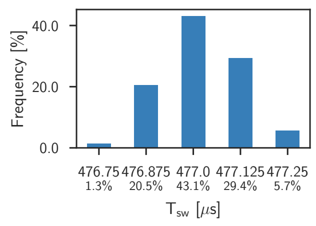
(b) Like (a) but with Flock.



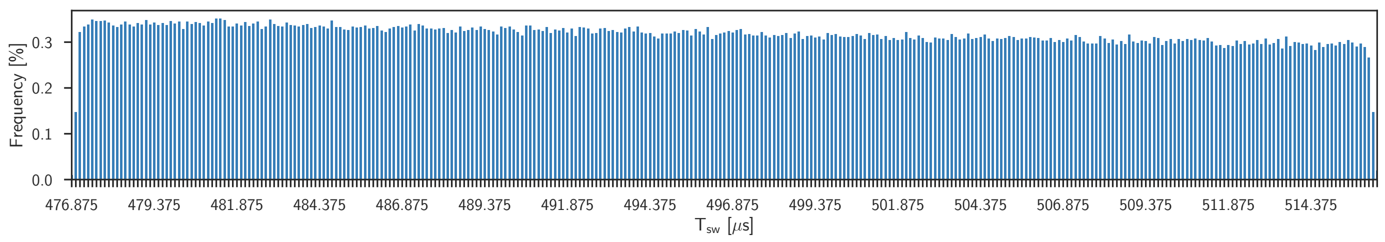
(c) With Flock, I = 97, packet length = 128 byte.



(d) With Flock, I = 2000, packet length = 9 byte.



(e) With Flock, I = 2000, packet length = 128 byte.



(f) Like (d), but without Flock.

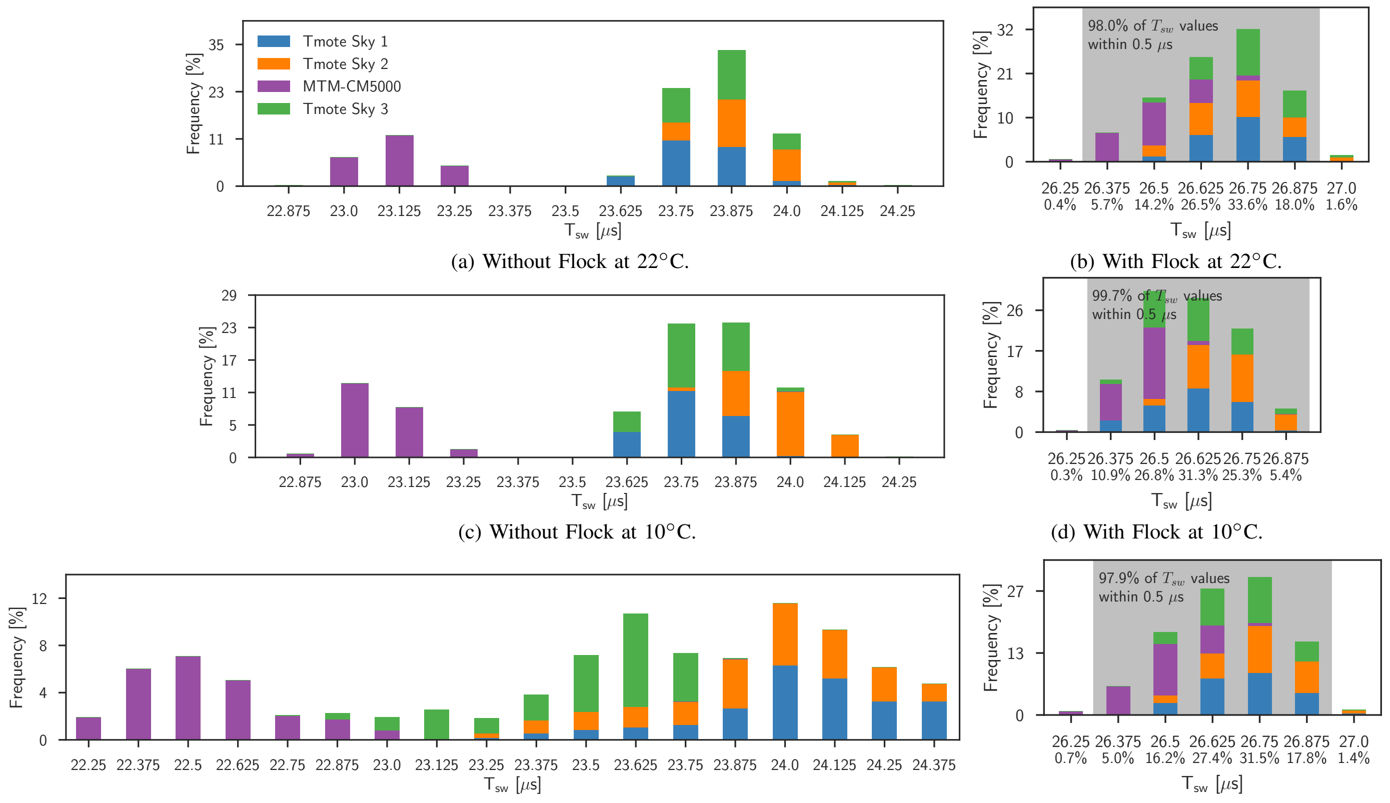
Fig. 3. Simulation results for the distribution of T_{sw}. Flock reduces DCO frequency offsets efficiently, even for large delays between packet reception and retransmission requests. It is independent of packet lengths and achieves constructive interference in over 99% of the transmissions for small packet lengths and a small I.

execute a corresponding number of no operations (NOPs). The consequence is an increase of T_{sw} in dependence of the maximum frequency deviation of the nodes in the network. In our implementation I is set to 112. We use a lookup-table to keep the processing overhead for calculating I^* to a minimum. The nodes select the number of NOPs to add to reach I^* from the look-up table depending on C^*_{rx}.

Using an oscilloscope we first measure T_{rx} for 9 byte packets. We find that T_{rx} is 288.6 μs. We assume that this discrepancy between measurements and theory is due to the fact that the CC2420 adds a processing delay when receiving a packet. At the sender side, we observed no such delay. Nonetheless, this has no effect on the design of Flock but it must be considered when the value of T_{rx} is calculated.

We measure T_{sw} on four receivers by connecting their SFD pins to an oscilloscope. Three receivers are Tmote Sky motes and one is a MTM-CM5000 mote. Both, Tmote Sky and MTM-CM5000 are variants of the well-known TelosB platform. To enforce different frequency drifts of the DCO, we repeat our experiments for three different temperature environments, namely, room temperature (22°), cold (10°C) and hot (40°C). These temperatures are within the operating conditions of the motes that are specified with -40°C and +85°C [28]. The cold and hot environments are generated by a refrigerator that can be switched to either cooling or heating. We carefully place the four receivers into the refrigerator and wait a few minutes before starting the experiments, so that the nodes can acclimatize. The initiator – the node that starts the communication and sends the packets to propagate – still operates at room temperature. We set the transmission power to maximum, which is 0 dBm, and make sure that the reception reliability is approximately 100%. We further make sure that the DCO calibrates with the stable 32 kHz clock before each packet transmission to ensure a fair evaluation. We first run Glossy at room temperature without Flock. After collecting over 2,000 samples, we repeat the experiment, but place all four receivers in a cold and later in a hot environment. We repeat the experiments afterwards with Flock enabled.

The plots on the left side of Fig. 4 show the distribution of T_{sw} in different temperature environments for the four receivers without Flock. As shown in Fig 4a, while the distribution of T_{sw} for the Tmote Sky motes is rather close to each other, the distribution of T_{sw} for the MTM-CM5000 strongly differs from the distribution of the Tmote Sky nodes. The resulting variable delay for T_{sw} spreads over Δ1.375 μs. A similar distribution of T_{sw} is achieved in the cold environment, shown in Fig. 4c. In the hot environment the distribution of T_{sw} expands to Δ2.125 μs, distributed over 18 values with a



(a) Without Flock at 22°C.

(b) With Flock at 22°C.

(c) Without Flock at 10°C.

(d) With Flock at 10°C.

(e) Without Flock at 40°C.

(f) With Flock at 40°C.

Fig. 4. Experimental results for the distribution of T_{sw}. The variable delay without Flock spreads over $\Delta 2.125$ μs. With Flock the variable delay reduces to $\Delta 0.75$ μs. The gray area marks the range, for which values of T_{sw} constructive interference can be achieved. Constructive interference can be achieved in 98% of the case.

distance of 125 ns to each other. Constructive interference is not guaranteed in all three temperature environments.

The plots on the right side of Fig. 4 depict the distribution of T_{sw} with Flock. T_{sw} spreads over $\Delta 0.75$ μs. In 98% of the cases the distribution of T_{sw} is within 0.5 μs, the necessary requirement to achieve constructive interference. The gray area marks the range, for which values of T_{sw}, constructive interference can be achieved. In all temperature environments, this area spreads from 26.375 μs to 26.875 μs. That means that constructive interference can be achieved when the nodes are located in mixed temperature environments.

As mentioned above, the code of Flock requires a few software instructions by itself. The average T_{sw} increases from 23.56 μs for Glossy without Flock to 26.63 μs with Flock. However, we show in the following experiments that Glossy with Flock can still achieve a lower latency and also reduces the time the radio is active during a Glossy flood.

C. The performance of Flock in controlled lab environments

In this set of experiments, we want to investigate the effect of clock offset and the influence of Flock on the performance of Glossy. We can report that Flock increases the reliability of Glossy by 3% and also reduces latency and the time the radio is active during a Glossy flood.

We use the code from the previous experiment and deploy it on three Tmote Sky motes – two senders and one receiver. We assign sender 1 and the receiver a nominal MCU frequency of 4,194,304 Hz. To simulate a frequency offset of 1.5% we set the nominal frequency of sender 2 to 4,131,389 Hz. We ensure that we only measure the effects of clock offset by eliminating external factors like multipath propagation that may also influence the temporal displacement of signals and thus, the performance of Glossy. We connect the motes at the SMA connection sleeves of the antennas via coaxial cables and a tee coaxial adapter as shown in Fig. 5. Triggered by the receiver, the two senders synchronously transmit a packet. At the receiver we measure three key metrics: The *reliability (R)* is the ratio of missed and total packets. The *latency (L)* indicates the time between the transmission of a packet and its first reception. And the *radio on-time (T)* is the time the radio is active during a Glossy flood. The receiver further collects information about dropped packets due to bit-flips caused by destructive interference: The parameter *wrong length (WL)* is the fraction of packets that have a length field which is either smaller than 2 byte (the size of the field containing, among others, the Cyclic Redundancy Check (CRC)) or greater than the maximum packet length of 127 byte, specified by the IEEE 802.15.4 standard [21]. Glossy distinguishes itself from other coexisting protocols by a header field with value `0xa0`. The parameter *wrong header (WH)* indicates the fraction packets that are dropped due to a value that differs from

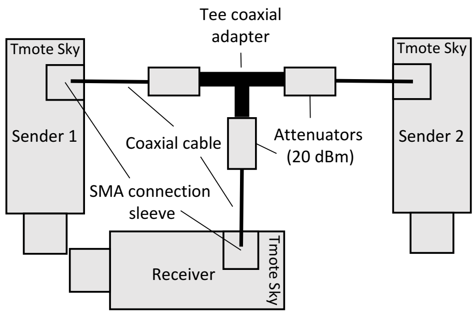


Fig. 5. Experiment setup.

TABLE I
EXPERIMENTAL RESULTS FOR THE PERFORMANCE OF FLOCK.

	R [%]	L [ms]	T [ms]	WL [%]	WH [%]	WC [%]
with Flock	98.74	2.464	6.986	13.21	5.98	80.81
without Flock	95.18	2.834	10.184	11.88	5.85	82.27

the expected one. Packet drops due to failed CRC checks are indicated with the *wrong CRC (WC)* parameter. We perform this set of experiments at room temperature and repeat each experiment three times, with 2,100 Glossy floods for each experiment. Before starting, we set the transmit power level of the nodes to 2 and ensure that all nodes receive packets with approximately the same power level (in our case -85 dBm).

Table I summarizes the results for this set of experiment. Our experiments reveal that Flock increases Glossy's reliability by around 3%. Latency and radio-on time are decreased by 13% and 31%, respectively. This means that even with a longer T_{sw}, compared to standard Glossy, Flock decreases latency. The reason is that the first packet is correctly received in an earlier Glossy slot. As a consequence the radio on-time is also reduced. The respective fractions of dropped packets are roughly the same with and without Flock.

V. RELATED WORK

Clock offset or clock drift compensation has been largely studied in the context of time synchronization. Examples of protocols used to establish and maintain synchronization among low-power nodes include RBS [29], TPSN [30], FTSP [31], RITS [32], RATS [33], and PulseSynch [34].

The time synchronization protocols mentioned above managed to achieve accuracy of the order of microseconds. For instance, it is reported that the average synchronization error between two nodes (for a single hop) for TPSN and RBS are 16.9 μs and 29.1 μs, respectively [30]. FTSP can reduce this error to 1.48 μs [31]. RITS, RATS, and PulseSynch also obtain synchronization errors of the order of a few microseconds.

To enable constructive interferences, however, sub-microseconds timing accuracy is necessary. In particular, to allow for constructive interference to occur with high probability, identical transmissions from separate nodes should be triggered with a temporal displacement of at most 0.5 μs [2]. This is however an upper limit. To achieve actual gains from concurrent transmissions in practical settings, the temporal displacement should be even lower, e.g., 0.2 μs [23].

Glossy [2] ensures that the temporal displacement between transmissions from concurrently transmitting nodes is below 0.5 μs with high probability. This goal is achieved by making the time elapsed between the reception of a message and the start of its retransmission to be very low. In particular, Glossy makes this time to be equal to 97 ticks of the DCO clock of the MSP430 microcontroller (MCU).[2] Because the clocks of different nodes can run at different frequency, however, the time required to execute the 97 cycles may also differ. Flock addresses this problem inherent in the design of Glossy and provides a practical solution to compensate on-the-fly for DCO clock errors. Furthermore, Flock allows to lift the constraint introduced by Glossy that nodes must retransmit incoming packets immediately.

König et al. [23] also recently proposed a method to achieve sub-microseconds synchronization among nodes and, thus, allow for synchronous transmission to be successfully achieved. Their approach, however, induces additional communication overhead and is less effective than Flock in ensuring the fulfillment of the "0.5 μs condition".

A common way to limit the impact of DCO errors is to rely on the Virtual High Resolution Timer (VHT) [24]. VHT combines the different clocks found on typical MCUs of low-power platforms. The common TelosB mote, for example, is equipped with a MSP430 processor featuring both a 4.1 MHz and a 32 kHz clock. While the latter is a low-resolution, but accurate clock, the first is a high-resolution, but less accurate one. VHT combines them both into a virtual timer, called VHT. For example, an event is time-stamped as time x on the low-frequency clock followed by y ticks on the high frequency clock. Recent work [23], [18], [20] utilize VHT to time synchronous transmissions, but also denote limitations [18], [20]. In particular, the accuracy obtained using VHT is mostly insufficient to ensure the fulfillment of the "0.5 μs condition".

VI. CONCLUSION

The Glossy protocol and subsequent work exploit the idea that the synchronous transmission of identical packets by neighboring nodes leads to constructive or non-destructive interference. This occurs, however, only if the temporal displacement between transmissions is less than 0.5 μs when employing IEEE 802.15.4 radios.

In this paper, we show that – due to the unstable DCO clocks used on common hardware platforms for low-power networks – this "0.5 μs condition" is challenging to reach. We thus introduce Flock, a novel approach to compensate for differences in DCO clock frequency across nodes. Furthermore, Flock allows for intermediate operations to be executed between the reception and retransmission of a message.

With respect to the existing literature, Flock obtains significantly better accuracy than previous approaches and can be seamlessly integrated into the code base of Glossy. This

[2]See Section 5.3 in [2], parameter I.

way, Flock ensures Glossy and other protocols that build upon it can operate reliably even in challenging environments. The source code of Flock is publicly available at `http://github.com/martinabr/flock`.

Acknowledgments

The authors would like to thank the anonymous reviewers for their very valuable comments. This work was partially supported by: The Collaborative Research Center SFB 921 Highly Adaptive Energy-Efficient Computing (HAEC) funded by the German Research Foundation; the Swedish Research Council (VR) through the project ChaosNet; and the Swedish Foundation for Strategic Research (SSF) through the project LoWi.

References

[1] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis, "A-mac: A versatile and efficient receiver-initiated link layer for low-power wireless," *ACM Trans. Sen. Netw.*, vol. 8, no. 4, pp. 30:1–30:29, Sep. 2012.

[2] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *ACM/IEEE IPSN*, apr 2011.

[3] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-Power Wireless Bus," in *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2012.

[4] M. Doddavenkatappa, M. C. Chan, and B. Leong, "Splash: Fast data dissemination with constructive interference in wireless sensor networks," in *Proceedings of the Symposium on Networked Systems Design & Implementation (USENIX NSDI)*, 2013.

[5] M. Suzuki, Y. Yamashita, and H. Morikawa, "Low-power, end-to-end reliable collection using glossy for wireless sensor networks," in *IEEE 77th Vehicular Technology Conference (VTC Spring)*, 2013.

[6] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Virtual synchrony guarantees for cyber-physical systems," in *Proceedings of the IEEE International Symposium on Reliable Distributed Systems (IEEE SRDS)*, 2013.

[7] K. P. Birman, "The process group approach to reliable distributed computing," *Commun. ACM*, vol. 36, no. 12, 1993.

[8] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks," in *ACM SenSys*, November 2016.

[9] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali, "Forwarder selection in multi-transmitter networks," in *Proceedings of the Conference Distributed Computing in Sensor Systems (DCOSS)*, 2013.

[10] D. Yuan, M. Riecker, and M. Hollick, "Making 'Glossy' Networks Sparkle: Exploiting Concurrent Transmissions for Energy Efficient, Reliable, Ultra-Low Latency Communication in Wireless Control Networks," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2014.

[11] M. Brachmann, O. Landsiedel, and S. Santini, "Concurrent Transmissions for Communication Protocols in the Internet of Things," in *IEEE LCN*, November 2016.

[12] J. Jeong, J. Park, H. Jeong, J. Jun, C. J. M. Liang, and J. Ko, "Low-power and topology-free data transfer protocol with synchronous packet transmissions," in *Proceedings of the Conference on Sensor, Mesh and Ad Hoc Communications and Networks (IEEE SECON)*, 2014.

[13] C. Sarkar, R. V. Prasad, R. T. Rajan, and K. Langendoen, "Sleeping beauty: Efficient communication for node scheduling," in *Proceedings of the IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2016.

[14] J. Zhang, A. Reinhardt, W. Hu, and S. S. Kanhere, "RFT: Identifying Suitable Neighbors for Concurrent Transmissions in Point-to-Point Communications," in *Proceddings of the ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 2015.

[15] D. Yuan and M. Hollick, "Ripple: High-throughput, reliable and energy-efficient network flooding in wireless sensor networks," in *Proceedings of the Symposium on a World of Wireless Mobile and Multimedia Networks (IEEE WoWMoM)*, 2015.

[16] C. A. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer, "TempLab : A Testbed Infrastructure to Study the Impact of Temperature on Wireless Sensor Networks," in *ACM/IEEE IPSN*, apr 2014.

[17] 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 Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2013.

[18] R. Lim, R. D. Forno, F. Sutton, and L. Thiele, "Competition: Robust flooding using back-to-back synchronous transmissions with channel-hopping," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2017.

[19] A. Escobar, J. Garcia, F. Cruz, J. Klaue, A. Corona, and D. Tati, "Competition: Redfixhop with channel hopping," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2017.

[20] B. A. Nahas and O. Landsiedel, "Competition: Towards low-power wireless networking that survives interference with minimal latency," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2017.

[21] IEEE standard 802.15.4 - 2011, "Wireless medium access control (mac) and physical layer (phy) specifications for low rate wireless personal area networks (lr-wpans)," Available at: http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf, 2011.

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

[23] M. König and R. Wattenhofer, "Maintaining Constructive Interference Using Well-Synchronized Sensor Nodes," in *DCOSS*, 2016, pp. 206–215.

[24] 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.

[25] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '05. Piscataway, NJ, USA: IEEE Press, 2005.

[26] T. Instruments, "Msp430f1611 datasheet," March 2011. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f1611.pdf

[27] ——, "Cc2420 datasheet," 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf

[28] Motiv, "Tmote Sky datasheet," Tech. Rep., 2006.

[29] 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.

[30] 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.

[31] 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.

[32] J. Sallai, B. Kusý, A. Lédeczi, and P. Dutta, "On the scalability of routing integrated time synchronization," in *Proceedings of the Third European Conference on Wireless Sensor Networks*, ser. EWSN'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 115–131.

[33] 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.*, vol. 1, no. 4, pp. 239–251, Jul. 2006.

[34] 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.