X-Sync: Cross-Technology Clock Synchronization Among Off-the-Shelf Wireless IoT Devices

Rainer Hofmann, David Grubmair, Carlo Alberto Boano, and Kay Römer
Institute of Technical Informatics, Graz University of Technology, Austria
Email: rainer.hofmann@tugraz.at, david.grubmair@gmail.com, cboano@tugraz.at, roemer@tugraz.at

***Abstract*—Clock synchronization in distributed IoT systems is a necessary feature to allow a coherent data collection and event detection. This task is challenging, as today's IoT systems often consist of heterogeneous wireless devices using incompatible technologies. Because of this, existing solutions often make use of multi-radio gateways, which allow an indirect synchronization across heterogeneous devices, but increase end-to-end delays and suffer from an increased overhead. In this work, we present X-Sync, a novel approach allowing a *direct* and *bidirectional* clock synchronization among *off-the-shelf* wireless IoT devices with incompatible physical layer. X-Sync leverages cross-technology communication to convey timing information among heterogeneous devices and presents novel techniques to compensate for the inaccuracies in reliably detecting the start of a cross-technology frame. We seamlessly integrate X-Sync into the Contiki-NG operating system and evaluate its performance experimentally on off-the-shelf Bluetooth Low Energy and IEEE 802.15.4 devices, showing that X-Sync achieves a *µs-level* synchronization accuracy.**

***Index Terms*—Cross-Technology Communication; Clock Synchronization; Contiki-NG; BLE; IEEE 802.15.4; X-Sync.**

I. Introduction

With the rise of more complex Internet of Things (IoT) solutions, heterogeneous networks are often needed to meet requirements regarding throughput, low-power, and range. Especially in industrial measurement and data acquisition systems, it is often necessary to use heterogeneous technologies to measure the performance of a system or to observe the occurrence of events. Without a proper synchronization between the devices, the timestamps referring to the same event would be inconsistent, thus making the collected data unusable. For this reason, an accurate clock synchronization is crucial to ensure a coherent data collection and event detection.

In conventional synchronization schemes, i.e., with devices employing the same physical layer (PHY), a global notion of time is established by periodically exchanging timing information, i.e., by broadcasting a global event or timestamps throughout the network [1]–[3]. Whenever a device receives such a message, it calibrates its local clock accordingly, achieving an accurate synchronization in the order of a few µs [4]. In heterogeneous networks, however, synchronization becomes more challenging, as devices employ incompatible PHYs, and are hence using diverse modulation schemes and channel bandwidths which makes an exchange of timing information via the usual communication channels infeasible.

Because of this, state-of-the-art solutions to synchronize heterogeneous devices rely on multi-radio gateways to forward data packets among devices with different PHYs [5], thus enabling an indirect exchange of timing information. The use of multi-radio gateways, however, introduces extra costs, increases traffic, and leads to variable end-to-end delays lowering the synchronization accuracy. Furthermore, multi-radio gateways constitute a bottleneck as well as a single point of failure, and their management is often complex and time-consuming, which limits their use on a large scale [6].

Cross-technology communication (CTC) has emerged as a suitable alternative to enable a direct data exchange among devices with incompatible PHY without the need of gateways. On a high level, there are two different approaches to carry out CTC: packet-level modulation, where a mutually-available side channel is established, and PHY emulation, which allows to exchange data at a high throughput. In *packet-level modulation*, data can be encoded into different frame lengths [7]–[9], gap durations [10], beacon intervals [11], or power levels [12], [13]; and decoded using energy detection, i.e., by letting the radio perform a high-frequency sampling of the received signal strength (RSS). In *PHY emulation*, instead, the payload of a data packet is adjusted such that a portion of it can be recognized by a device using another technology as a legitimate packet, thereby achieving high data rates [14]–[18].

Lack of CTCS solutions. However, existing work on CTC has mostly focused on demonstrating its feasibility on various IoT platforms and on achieving high data rates, leaving aside its use for time synchronization among heterogeneous devices, i.e., for cross-technology clock synchronization (CTCS). Tan et al. [19] have proposed a time synchronization approach that leverages PHY emulation to encapsulate an IEEE 802.15.4 packet containing a timestamp within the payload of a Wi-Fi packet. Based on the authors' experimental evaluation, this allows to achieve a synchronization accuracy of around 30 µs. Another approach, named Crocs [20], splits the synchronization process into a mutually-detectable event and the transmission of a timestamp via packet-level CTC, which achieves a synchronization accuracy of around 1 ms. However, both approaches only allow an unidirectional synchronization from Wi-Fi to IEEE 802.15.4 devices, and do not target other popular IoT technologies such as Bluetooth Low Energy (BLE).

CTC-specific challenges. To improve the synchronization accuracy when using packet-level CTC to convey timing information, several challenges need to be tackled. In conventional synchronization schemes, the time between the transmission and the detection of a message is only defined by the physical

propagation time of the message through the media, which can easily be compensated or even neglected as it is usually below 1 µs, deterministic, and constant (assuming a one hop scenario) [3]. However, due to the limited RSS sampling frequency of commercial wireless devices and the RSS averaging defined in the IEEE 802.15.4 standard, the start of a CTC frame cannot be accurately detected on the receiving device. This significantly increases the time between the transmission and the detection of a message by a non-deterministic, highly variable factor depending on the hardware characteristic of the device [21]. Because of this, new compensation methods have to be developed to improve the accuracy of CTCS. Furthermore, as a single CTC frame involves the transmission of multiple legitimate data packets, packet-level modulation results in very long transmission times (e.g., ≈ 63 ms for a 20-byte frame [22]), making it challenging to frequently exchange timing information without congesting the wireless channel or draining the battery of the device. While PHY emulation CTC is not affected by the limited RSS sampling frequency or long transmission times, it only allows a unidirectional data exchange between two specific technologies. This makes the use of round-trip algorithms more complicated, as different CTCS schemes are needed for each direction, and it does not allow a simultaneous synchronization of devices employing diverse technologies, i.e., it does not allow an exchange of timing information using broadcast transmissions.

Our contributions. In this paper, we address the lack of CTCS schemes and present *X-Sync*, a novel approach that allows to synchronize the clock of heterogeneous devices without the use of multi-radio gateways. In particular, X-Sync enables a *bidirectional* clock synchronization among *off-the-shelf* wireless devices achieving a *µs-level accuracy*. It leverages packet-level CTC to convey timing information in a generic way, which allows to synchronize several heterogeneous devices simultaneously, regardless of the used technology, by transmitting *cross-technology broadcast* frames. We have implemented X-Sync on off-the-shelf BLE and IEEE 802.15.4 devices, where the latter are especially challenging as they introduce an additional source of error due to the averaging of the RSS values. However, due to its generic design, X-Sync can be used by any device capable of sampling the RSS, e.g., off-the-shelf Wi-Fi devices such as Raspberry Pis [23], [24]. To account for the non-deterministic delay in the communication process, caused by the limited RSS sampling capability on off-the-shelf wireless devices, X-Sync performs a *repetitive sampling* of the RSS in combination with a *binary search algorithm* to accurately determine the start of a CTC frame, which is crucial to achieve an accurate synchronization. To keep the number of synchronization messages low, X-Sync *estimates the clock skew* between the devices and calibrates the local clock accordingly, achieving a high accuracy even when using long synchronization intervals. Due to its bi-directionality, X-Sync can also be used as a building block to allow the use of existing multi-hop synchronization schemes (such as RBS [1], TPSN [2], or FTSP [3]) among heterogeneous devices.

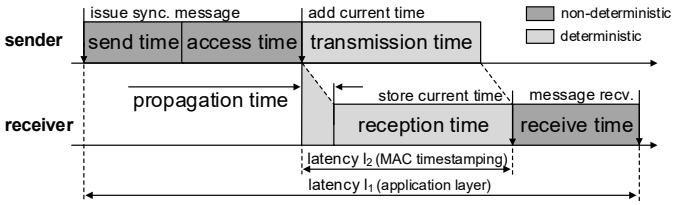


Fig. 1: Delays of exchanging a timestamp over a wireless link.

In summary, this paper makes the following contributions:

- We present X-Sync, a generic and novel cross-technology clock synchronization scheme for heterogeneous wireless devices, highlighting its key design principles (Sect. II);
- We illustrate in detail the working principle of X-Sync, describing how it enables a bidirectional and accurate cross-technology clock synchronization, despite the use of packet-level CTC to convey information (Sect. III);
- We describe X-Sync's seamless integration into the Contiki-NG operating system and its implementation on off-the-shelf BLE and IEEE 802.15.4 devices (Sect. IV);
- We evaluate X-Sync experimentally on off-the-shelf devices, showing that a synchronization error in the µs-range can be achieved; we further evaluate the impact of various parameters on X-Sync's synchronization accuracy and analyse its memory footprint (Sect. V).

After describing related work in Sect. VI, we conclude our paper in Sect. VII along with an outlook on future work.

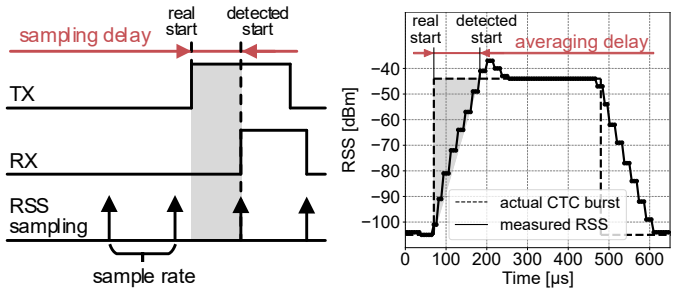
II. X-SYNC: DESIGN RATIONALE

In this section, we describe the rationale behind the design of X-Sync and the challenges in devising an accurate clock synchronization scheme for heterogeneous wireless devices.

Removing higher-layer delays. The accuracy of a synchronization scheme is primarily limited by the delays added during the communication process, i.e., by the delay of sending and receiving timestamps, and should be eliminated or compensated to allow a precise synchronization. Those delays were already extensively studied and analysed in the literature [2], [3], [25] and are illustrated in Fig. 1 in more detail.

As can be seen in Fig. 1, the end-to-end latency when exchanging a timestamp traditionally equals l_1, i.e., it lasts from the instant in which the application issues the message to the instant in which it is decoded by the receiver. To get rid of *non-deterministic delays* introduced by the operating system and network stack (send and receive time in Fig. 1), as well as by the medium access control scheme (access time), X-Sync uses *MAC timestamping* [2], where timestamps are generated by reading the local clocks immediately when a message is sent or received. This significantly reduces the end-to-end latency when exchanging a timestamp (l_2 in Fig. 1) to the physical propagation time of the message through the media (propagation time) and to the time it takes to transmit (transmission time) or receive (reception time) the timestamp.

Building a generic CTCS scheme. However, to allow the synchronization of multiple heterogeneous devices, independently of the employed technology, a generic way of exchang-



(a) sampling delay (b) averaging delay

Fig. 2: Due to the limited resolution of RSS sampling capabilities in common IoT platforms, one needs to account for specific delays that may increase the inaccuracy in detecting the start of a cross-technology frame.

ing the timestamps is needed. Furthermore, in conventional synchronization schemes it is often necessary to exchange data bidirectionally to improve the synchronization accuracy, e.g., by measuring the round-trip time [2]. Because of this, we base X-Sync on *packet-level modulation* instead of PHY emulation, allowing X-Sync to be used as a building block for other synchronization schemes among different wireless technologies. To this end, we have extended *X-Burst* [21] – a generic and portable CTC framework allowing constrained IoT platforms with incompatible physical layer to seamlessly interact using packet-level modulation – to enable the transmission and reception of timestamps. Thus, X-Sync uses precisely-timed packets to synchronize devices with different PHY.

Overcoming packet-level CTC limitations. To detect and decode the information within such packets, a receiver performs a high frequency sampling of the RSS. However, due to the limited RSS sampling rate and processing speed of wireless devices, which strongly vary among platforms [21], the instant in which a packet's presence is detected may be delayed. As a result, the start of a CTC frame cannot be accurately detected, as the RSS can only be sampled at specific points in time. Fig. 2a illustrates this problem: the start of a packet was shortly missed, and an additional delay, which we call the *sampling delay*, is added to the communication process. This sampling delay has a negative impact on the synchronization accuracy. However, this becomes even worse on IEEE 802.15.4 devices, as those additionally experience a constant, device-dependent delay, caused by the averaging of the RSS, as can be seen in Fig. 2b [21]. This second delay, which we call the *averaging delay*, strongly depends on the setup and the threshold used to distinguish between data packets and the RSS noise floor.

To compensate for the sampling delay (and hence to minimize the variability of the end-to-end latency), X-Sync performs a *repetitive sampling* of the RSS, i.e., instead of continuously sampling the RSS, X-Sync only samples at specific points in time. This allows to accurately detect a well-known sequence of energy bursts (i.e., the synchronization preamble, consisting of N_B bursts), which is contained within each CTC frame, and thus, to refine the detected start time of the CTC frame. For the remaining synchronization error (caused by the

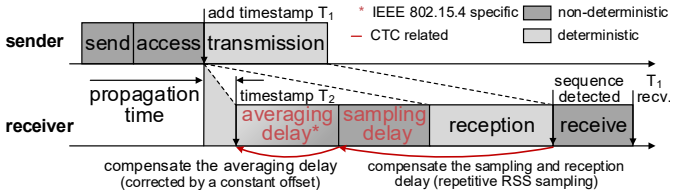


Fig. 3: Communication process (delays) over a wireless link when packet-level CTC is used to convey the timestamp.

propagation time and by the averaging delay on IEEE 802.15.4 devices), the detected start time of the frame can be further corrected by a constant, measurable offset. More information about how X-Sync refines the start time of a CTC frame can be found in Sect. III-B. Fig. 3 shows the communication process when packet-level CTC is used to convey the timestamp, including the sampling delay and the averaging delay.

Minimizing the number of synchronization messages. Another important aspect when using packet-level CTC to convey timing information is its very long transmission times (e.g., ≈ 63 ms for a 20-byte frame [22]). However, as the time offset between two devices changes in a linear fashion (assuming a good short-term stability of both clocks and a constant environmental temperature), X-Sync estimates the clock drift between two devices using *linear regression* [1]. This way, after sufficient data points are collected, the synchronization interval I_S can be increased, and thus, the number of synchronization messages and the energy consumption be reduced.

III. X-Sync: Working Principle

To synchronize the clocks of heterogeneous devices, X-Sync uses precisely-timed energy bursts that are generated by sending legitimate data packets of variable length. An overview of X-Sync is shown in Fig. 4 and discussed in more detail below.

A. Accurate Timestamp Transmission

As can be seen in Fig. 4, a short preamble (CTC preamble) is sent at the beginning of each CTC frame. This allows other devices to distinguish between noise and actual CTC frames on the wireless channel. Moreover, by using a different preamble for synchronization messages, a receiver can further distinguish between usual CTC messages and synchronization ones. Thus, a device immediately knows if the following energy bursts will contain timing information and, if this is the case, starts compensating for the sampling delay. Therefore, another preamble (synchronization preamble) is transmitted right after the first one, which allows the receiver to accurately determine the start of the CTC frame. We discuss the synchronization preamble and how it can be used to compensate the sampling delay in Sect. III-B in more detail. After both preambles are sent, the actual timestamp T_1, which is measured right before the transmission of the CTC message, is transmitted, followed by a checksum to detected possible transmission errors.

B. Accurate Timestamp Reception

When the CTC preamble was detected by a receiver, it immediately starts performing a *repetitive sampling* on the

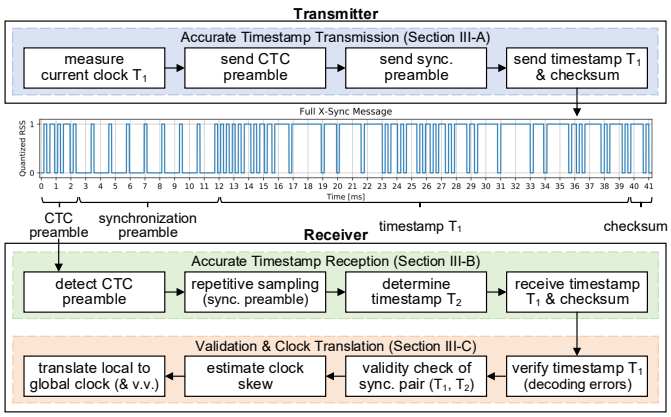


Fig. 4: Overview of the working principle of X-Sync. The timestamp T_1 is measured shortly before the transmission. Each frame consists of a CTC preamble, synchronization preamble, timestamp T_1, and a checksum. The receiver refines the detected start time of the frame T_2, validates the timing information, estimates the clock skew, and translates its clock.

following N_B energy bursts, i.e., on the synchronization preamble. In particular, as the RSS sampling frequency is defined by the underlying hardware, it cannot be changed without hardware modifications, but the specific point in time when the RSS is sampled can be chosen freely. Thus, instead of continuously sampling the RSS, the instant of the next sampling point can be delayed to a certain point in time. X-Sync uses this possibility, which is called repetitive sampling, in combination with a *binary search algorithm* to accurately determine the start of the last energy burst within the synchronization preamble. As the nominal durations of (and the gaps in-between) both preambles are known, the receiver can accurately refine the start time of the CTC frame, i.e., determine timestamp T_2. This procedure is illustrated in Fig. 5.

As can be seen in Fig. 5, an *uncertainty region* ϵ is specified, defined by the RSS sampling frequency (f_{RSS}), in the middle of which the start of the energy burst is expected. Therefore, the start of the first energy burst t_b^1 will be within:

$$t_e^1 - \frac{\epsilon^1}{2} \leq t_b^1 \leq t_e^1 + \frac{\epsilon^1}{2}, \tag{1}$$

where t_e^1 is the estimated start of the first energy burst and ϵ^1 is the first uncertainty region, defined by $1/f_{RSS}$.

After each energy burst, ϵ gets halved ($\epsilon^{i+1} = \epsilon^i/2$) and depending on the sampled value, i.e., if a burst was detected (RSS $\geq$ threshold) or not (RSS $<$ threshold), the estimated start of the next energy burst t_e^{i+1} (i.e., the instant of time in which the next RSS is sampled) is adjusted accordingly. If no packet was detected, the sampling point was chosen too early and the next sampling point will be slightly postponed and vice-versa:

$$t_e^{i+1} = \begin{cases} t_e^{i+1} + \frac{\epsilon^{i+1}}{2}, & \text{if } RSS < threshold \\ t_e^{i+1} - \frac{\epsilon^{i+1}}{2}, & \text{if } RSS \geq threshold \end{cases} \tag{2}$$

However, when using IEEE 802.15.4 devices, also the averaging delay has to be compensated to achieve a high synchronization accuracy. As the averaging delay also depends on the environment, we distinguish between static and dynamic setups. In the former case, where all nodes are placed at fixed positions and transmit with constant power, the averaging delay is fully deterministic. Thus, it can be measured (including the propagation delay to further increase the accuracy) by analysing the synchronization error among the first few synchronization messages, and be considered in the future when refining the detected start time of the CTC frame, i.e., when determining timestamp T_2. For dynamic setups, the delay becomes non-deterministic, as it strongly depends on the distance between sender and receiver, on the used transmission power, and on changes in the environment. To also compensate for the averaging delay in dynamic environments, the delay has to be determined for each synchronization message separately. This can either be done by comparing the measured durations of the preamble with its nominal ones, or by the use of round trip algorithms such as TPSN [2]. However, the compensation of the averaging delay in dynamic environments is beyond the scope of this paper and will be addressed in future work.

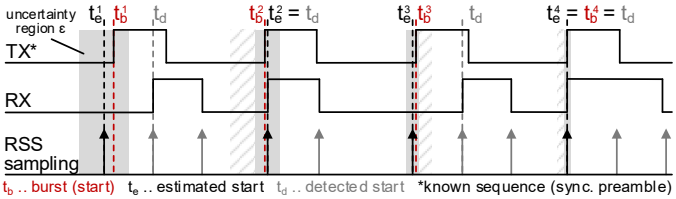


Fig. 5: X-Sync uses repetitive sampling and a binary search algorithm to determine the start of the last burst within the synchronization preamble, compensating for the sampling delay. Note that X-Sync only samples at the black arrows, the gray ones only illustrate the error in detecting the bursts.

C. Validation & Clock Translation

Due to the nature of packet-level CTC (encoding information into properties of legitimate data packets), transmissions are more vulnerable to external RF interference than traditional ones. Thus, X-Sync includes a checksum to verify the integrity of each received timestamp. Once the integrity is verified, X-Sync builds a synchronization pair consisting of timestamp T_1 (the received timestamp), and timestamp T_2 (the corrected timestamp referring to the start of the frame) allowing the device to translate its local clock to the global clock as follows:

$$t_{global} = t_{local} + t_{offset}, \quad \text{with} \quad t_{offset} = T_1 - T_2 \tag{3}$$

However, as clocks are running at slightly different frequencies due to differences in their oscillators, the synchronized clocks immediately start to drift apart again. This drift factor between two clocks is called *clock skew* and correcting it becomes especially important for long-term synchronization. Assuming a good short-term stability of the clocks and constant environmental temperature conditions, the clock skew can be assumed to be linear [3]. Thus, X-Sync performs a *least-squares linear regression* to find the best fit line through the clock offsets observed over time, allowing to simply

convert the local clock of a device to the global clock from the slope ($skew$) and the intercept ($offset$) of the line:

$$t_{global} = t_{local} * skew + offset \tag{4}$$

As the amount of memory is usually constrained on IoT devices, not all synchronization pairs can be stored for the linear regression. Therefore, X-Sync uses a moving window buffer where the last N_P pairs are stored in a first-in-first-out manner. As also the synchronization preamble is vulnerable to external RF interference, leading to a wrong correction of timestamp T_2, X-Sync performs a validity check of each synchronization pair before it is added to the buffer. In particular, random sample consensus (RANSAC) [26] is used to differentiate between inliers and outliers. As devices often only have limited resources, a simple linear model is used:

$$m = \frac{y_2 - y_1}{x_2 - x_1}, \quad b = \frac{y_1 \cdot x_2 - y_2 \cdot x_1}{x_2 - x_1}, \tag{5}$$

where (x_1, y_1) and (x_2, y_2) are two different synchronization pairs. In particular, until the moving window buffer is not fully filled, X-Sync stores all received synchronization pairs; once the buffer is full, two hypothetical inliers are randomly selected and the model from Eq. 5 is derived. All other synchronization pairs of the buffer are then tested against this model using a specific loss function and the number of valid samples is stored for comparison. This procedure is repeated with new hypothetical inliers and after a pre-defined number of repetitions is reached, the model with the most valid samples is used to validate new synchronization pairs.

IV. Implementation

We have seamlessly implemented X-Sync into the Contiki-NG operating system, making sure that no changes to its core functions and network stack are necessary. To this end, we have extended *X-Burst* [21] – a generic and portable CTC framework allowing constrained IoT platforms with incompatible PHY to seamlessly interact using packet-level modulation – to enable the transmission and reception of timestamps and ported it to Contiki-NG. To maximize code-reuse and portability to other platforms, X-Sync was implemented as a modular extension of X-Burst, as can be seen in Fig. 6. We describe next the main differences to X-Burst in more detail.

Frame management. We have extended the frame management module to assemble and disassemble synchronization messages, which consist of a CTC preamble (5 bursts), a synchronization preamble (N_B bursts), the timestamp T_1 (64 bit), and a checksum (1 byte). When a synchronization message is received, the contained timestamp T_1 is passed to the estimation & filtering module for further processing.

Hardware abstraction layer. As the provided timers of Contiki-NG are rather limited in their resolution (i.e., 32 kHz), we have extended the hardware abstraction layer (HAL) of X-Burst by providing a 64-*bit high-resolution timer*, running at the system clock of the used platform, allowing to precisely schedule the operations of X-Sync on a sub-µs level. However, as the accurate transmission of energy bursts is crucial

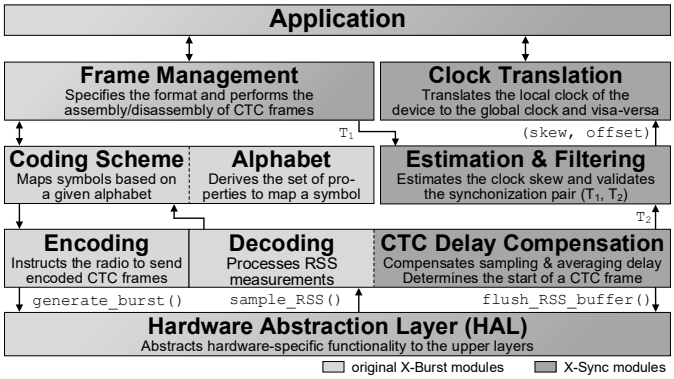


Fig. 6: Extension of the modular X-Burst framework with X-Sync's modules (highlighted in dark gray).

to enable a high synchronization accuracy, i.e., an accurate transmission of the preambles is needed to compensate for the sampling delay, we further extended the HAL to allow the use of *radio queues*. This way, the radio has full control over the transmission timings and can use a constant gap between two consecutive bursts, regardless of the operations carried out on the CPU. In case radio queues are not supported, the bursts are sent one by one, with a precise delay in-between, and the *interrupt priorities* are updated to avoid delays caused by unexpected interrupts (if interrupt priorities are not supported, interrupts are disabled during transmissions). We further extended the HAL to enable *MAC timestamping*, i.e., to generate and add timestamp T_1 immediately before the message is sent.

CTC delay compensation. To compensate for the delays introduced by the use of packet-level CTC, we extended the decoding module of X-Burst accordingly. In particular, when the CTC preamble was detected, the *repetitive sampling* of the RSS is started to refine the detected start time of the frame, i.e., to determine timestamp T_2, as explained in Sect. III-B. As IEEE 802.15.4 devices are averaging the RSS over the last 8 measurements, we reset the RSS buffer to improve the accuracy in detecting the start of the next energy burst within the synchronization preamble (if resetting the buffer is not possible, turning the radio on and off will also reset the buffer).

Estimation & filtering, and clock translation. The estimation and filtering module validates a synchronization pair (T_1, T_2), i.e., it detects and removes outliers using RANSAC, and estimates the clock skew using linear regression, as described in Section III-C. To speed-up the calculations and to save memory, the number of stored synchronization pairs (N_P) can be adjusted. The skew and the clock offset is then passed to the clock translation module, which allows to convert timestamps between devices, instead of correcting the devices' clock.

Hardware platforms. We implement X-Sync on three *off-the-shelf* IoT devices supported by Contiki-NG, namely the TI CC2650 LaunchPad (supporting BLE and IEEE 802.15.4), the Zolertia Firefly (employing an IEEE 802.15.4-compatible TI CC2538 transceiver), as well as the TelosB mote (employing an IEEE 802.15.4-compatible TI CC2420 radio).

V. Evaluation

We evaluate X-Sync experimentally. We first showcase the synchronization accuracy among several off-the-shelf BLE and IEEE 802.15.4 devices and analyse the impact of different parameters to the achievable accuracy (Sect. V-A). We then show a long-term evaluation of X-Sync (Sect. V-B) and finish our evaluation by analysing its memory footprint (Sect. V-C).

Experimental setup. We make use of two TI CC2650 LaunchPads (one configured in BLE mode and one configured in IEEE 802.15.4 mode), one TelosB mote, as well as one Zolertia Firefly. Unless specified differently, we carry out all experiments in a vacant room with devices placed at 1 meter distance and using a transmission power of 0 dBm. To evaluate the synchronization accuracy, we make use of a function generator that generates a signal clocked at 1 Hz (f_G), which is connected using short BNC cables to the GPIO pins of each device. Whenever a rising edge is detected on a GPIO pin, the device samples the current time and transmits it via a UART connection to a database. The setup is illustrated in Fig. 7.

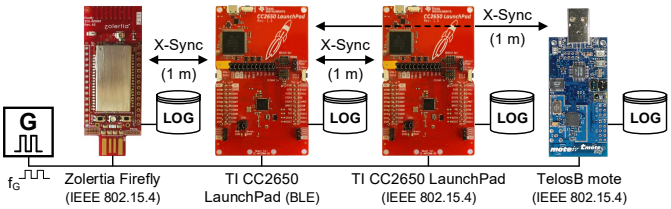


Fig. 7: Experimental setup used in the evaluation. Each device is connected to a synchronized signal (1 Hz) generated by a function generator. This signal accurately triggers the measurement of the current time, which is then stored into a database.

A. Synchronization Accuracy

Depending on device capabilities, such as the computational power or available memory, X-Sync can be configured to always achieve the best possible performance. In particular, there are three different parameters with a direct impact on the achievable synchronization accuracy: the used *synchronization interval* I_S, the number of *synchronization preamble* bursts N_B, and the number of stored *synchronization pairs* N_P. To evaluate the impact of each of these parameters on the synchronization accuracy, the remaining two parameters are set to their optimal values to not affect the evaluation. Each experiment is repeated 3 times and 200 synchronization messages are exchanged within each test run. Due to space constraints, we only show the evaluation of the parameters for the BLE $\rightarrow$ IEEE 802.15.4 direction, but the results apply also for the other direction. In every experiment, the TI CC2650 LaunchPad in BLE mode transmits a synchronization message to all other IEEE 802.15.4 devices simultaneously and acts as the reference clock. To compensate for the static *propagation delay*, and for the *averaging delay* on IEEE 802.15.4 devices, we measure the synchronization error (i.e., the remaining delay in detecting bursts) of the first 10 synchronization pairs and use the average error for future corrections of timestamp T_2.

Synchronization interval (I_S). The synchronization interval has the highest impact on the synchronization accuracy, as it defines the time between two synchronization messages sent consecutively. While a large interval leads to a lower energy consumption, the accuracy decreases due to the limited compensation of the clock skew (non-linear in the long term). Fig. 8a shows the impact of I_S on the achievable accuracy.

As can be seen in Fig. 8a, the clock of the TI CC2650 LaunchPad in IEEE 802.15.4 mode and the Zolertia Firefly could maintain a very high synchronization accuracy throughout the evaluation, reaching up to a *sub-µs level* accuracy when $I_S = 1$. Due to the very constrained hardware of the TelosB mote, i.e., due to its slow crystal oscillator (32 kHz) and clock speed (3.9 MHz), as well as the lack of preemptive interrupts, the reached accuracy is much lower compared to the other two platforms. However, even with its limited capabilities, the TelosB mote could still decrease the synchronization error down to below ± 100 µs, outperforming packet-level CTCS state-of-the-art by a factor of 10 (Crocs [20] reached an accuracy of 1 ms on the TelosB mote using an interval of 7 s).

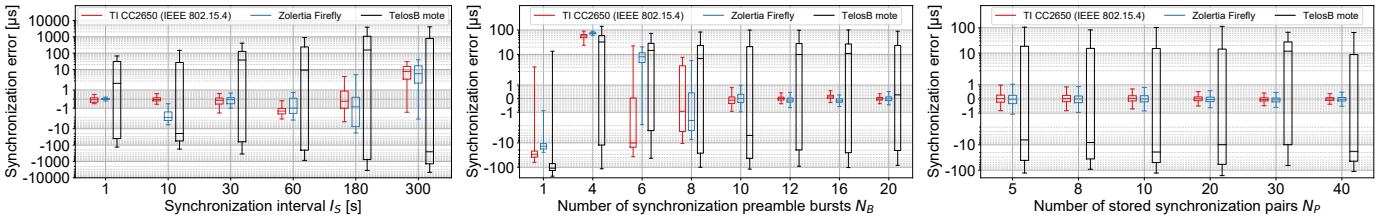
Synchronization preamble bursts (N_B). Another important parameter affecting the synchronization accuracy is the number of synchronization preamble bursts N_B that are used to compensate for the sampling delay. The more energy bursts are used, the more iterations are available for the binary search algorithm, and thus, the detected start time of the message can be refined more precisely. However, while using more energy bursts will increase the accuracy, it also increases the energy consumption and the length of a synchronization message. Fig. 8b shows the impact of N_B on the achievable accuracy.

As can be seen in Fig. 8b, setting $N_B = 1$ completely cancels the binary search, resulting in a delayed detection of the burst and thus, in a negative synchronization error. The exact opposite happens when $N_B = 4$, where the start of the fourth burst is expected too early, resulting in a positive synchronization error. By increasing N_B further, the binary search algorithm starts working as expected, reaching a good and stable performance when 10 or more bursts are used.

Stored synchronization pairs (N_P). The last parameter that influences the accuracy of X-Sync is the number of stored synchronization pairs N_P. The more pairs are stored, the better the clock skew estimation will be, but at the cost of a higher memory usage. Fig. 8c shows the impact of N_P on the achievable accuracy. The number of used synchronization pairs N_P does not have a major impact on the achievable accuracy, showing that the filtering and the clock skew estimation work even on a very small set of synchronization pairs.

B. Long Term Evaluation

Based on our findings from Sect. V-A, we evaluate the performance of X-Sync over a duration of 35 hours. In particular, we set the synchronization interval $I_S = 60$ s, as we consider this as a reasonable interval and good trade-off between accuracy and the number of required synchronization messages. We further set the number of synchronization preamble bursts $N_B = 12$, as using more would not significantly increase the

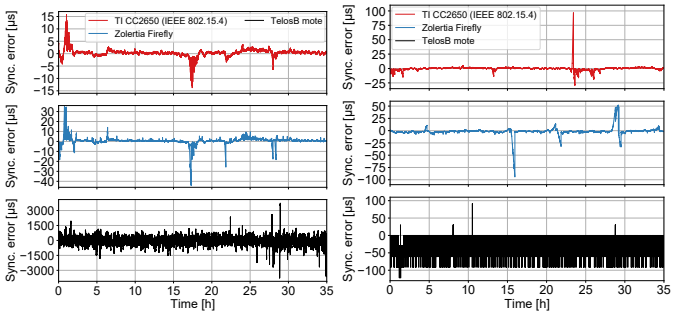


(a) Depending on the synchronization interval I_S ($N_B = 20, N_P = 40$).

(b) Depending on the number of synchronization preamble bursts N_B ($I_S = 1s, N_P = 40$).

(c) Depending on the number of stored synchronization pairs N_P ($I_S = 1s, N_B = 20$).

Fig. 8: Synchronization accuracy depending on different parameters. The TI CC2650 LaunchPad in BLE mode broadcasts a synchronization message (with N_B bursts) every I_S seconds and acts as reference clock. N_P synchronization pairs are stored.



(a) BLE → IEEE 802.15.4.

(b) IEEE 802.15.4 → BLE.

Fig. 9: Long term evaluation of the synchronization error. The TI CC2650 in BLE mode acts as (a) transmitter broadcasting to all other devices simultaneously and (b) as receiver. A message was sent once a minute ($I_S = 60s, N_B = 12, N_P = 20$).

accuracy. Last, we set the number of stored synchronization pairs $N_P = 20$ to keep the memory usage low, but still allow a good estimation of the clock skew and an increased robustness to outliers. Fig. 9a shows the long term evaluation of X-Sync when the TI CC2650 LaunchPad in BLE mode broadcasts the synchronization messages to all other platforms simultaneously. To also show the bidirectionality of X-Sync, we repeat the long term evaluation three times, where in every round another IEEE 802.15.4 device transmits the synchronization messages to the TI CC2650 LaunchPad in BLE mode and acts as the reference clock. The results are shown in Fig. 9b. As each evaluation is carried out on its own, disturbances that can be seen in Fig. 9b are not related to each other.

As can be seen in Fig. 9a, a high synchronization accuracy could be maintained throughout the evaluation. For 95 % of the time, the TI CC2650 LaunchPad in IEEE 802.15.4 mode and the Zolertia Firefly could maintain a very low synchronization error below $\pm 2.52\,\mu s$ and $\pm 6.16\,\mu s$, respectively; whereas the TelosB mote could only maintain an error below $\pm 812\,\mu s$. No precaution was taken against external RF interference, examples for their impact can be seen at $t = 1\,h$ and $t = 17\,h$ on both the TI CC2650 LaunchPad in IEEE 802.15.4 mode and the Zolertia Firefly (minor disturbances can also be seen at $t = 6\,h$, $t = 22\,h$ and $t = 28\,h$). On the TelosB mote, however, those disturbances are not significant enough to be visible, due to its worse performance compared to the other platforms.

As can be seen in Fig. 9b, also for the other communication direction (IEEE 802.15.4 → BLE) the TI CC2650 LaunchPad in BLE mode could maintain a high synchronization accuracy throughout the evaluation. When using the TI CC2650 LaunchPad in IEEE 802.15.4 mode and the Zolertia Firefly as transmitter, for 95 % of the time the synchronization error was below $\pm 4.84\,\mu s$ and $\pm 10.80\,\mu s$, respectively. The variations compared to the results shown in Fig. 9a are due to differences in the implementation as well as due to variations in the environmental conditions during the evaluations. However, when using the TelosB mote as transmitter, the error is, again, slightly higher compared to the other platforms. In this case, the error stays below $\pm 30\,\mu s$ for 95 % of the time: this is due to the limited time resolution of the TelosB mote, which uses a timer clocked at 32 kHz (1 tick corresponds to $30.52\,\mu s$).

Hardware platform	RAM / ROM (kB)		
	X-Burst	w/ X-Sync	**X-Sync only**
TI CC2650	1.17 / 7.86	9.97 / 15.44	**8.80 / 7.58**
Zolertia Firefly	1.26 / 6.97	1.76 / 12.81	**0.50 / 5.84**
TelosB mote	0.75 / 6.78	1.18 / 14.20	**0.43 / 7.42**

TABLE I: Memory footprint for different hardware platforms.

C. Memory Footprint

We conclude our evaluation by quantifying the memory footprint of X-Sync using the `gcc-size` command. Table I shows the memory footprint of X-Sync, when $N_P = 20$, in terms of RAM and ROM usage for all used platforms. The table also reports the memory footprint of X-Burst, on top of which X-Sync is built. While the memory footprint is roughly the same for all platforms ($< 1\,kB$ of RAM and $< 8\,kB$ of ROM; the tiny differences are due to slightly different implementations), the RAM usage of the TI CC2650 LaunchPad ($\approx 9\,kB$) differs significantly, due to its implementation of radio queues. Therefore, with its low memory footprint, X-Sync is well-suited for resource-constrained wireless IoT devices.

VI. Related Work

We analyse next related work on clock synchronization in wireless sensor networks, on CTC, as well as on CTCS.

Clock synchronization in wireless sensor networks. Clock synchronization has been extensively studied in sensor networks [1]–[3], [27]–[29]. While RBS [1] uses reference broadcasts to synchronize multiple receivers (removing the non-deterministic send and access time), TPSN [2] builds a spanning tree of the network and performs pair-wise synchronization with MAC-timestamping to remove the remaining delays

in the process. FTSP [3] and Glossy [29] enhance the accuracy by using network flooding. However, all of those works are infeasible for heterogeneous devices as timing information cannot be exchanged due to their incompatible PHYs.

Cross-technology communication. Because of this, a large body of work has explored how to enable a direct communication among heterogeneous devices, focusing on Wi-Fi, BLE, and IEEE 802.15.4. While early works have focused on establishing a mutually available side channel by using packet-level information such as the duration [7]–[9], interval [10], [11] and transmission power [12], [13] to encode data, later works have exploited the concept of PHY emulation to significantly increase the data rate of CTC [14]–[18].

Cross-technology clock synchronization. However, only two works have demonstrated CTCS. Tan et al. [19] proposed a time synchronization approach that leverages PHY emulation to encapsulate an IEEE 802.15.4 packet containing a timestamp within the payload of a Wi-Fi packet, enabling a synchronization accuracy of around 30 µs. Crocs [20], instead, splits the synchronization process into two parts: a mutually-detectable event, realized using a Barker-code based sequence of Wi-Fi packets, and the transmission of a timestamp via packet-level CTC, enabling a synchronization accuracy of around 1 ms. However, both works only enable an unidirectional synchronization, leaving aside the development of a more generic, bidirectional CTCS approach that can be used as a building block for existing synchronization schemes.

VII. Conclusions and Future Work

In this paper we have presented X-Sync, a novel approach allowing a direct, bidirectional clock synchronization among off-the-shelf wireless IoT devices with incompatible PHY. We identified hidden delays in the synchronization process, when CTC is used to convey timing information, and presented novel techniques to compensate for those delays. Because of this, X-Sync is able to synchronize heterogeneous devices with a (sub-) µs-level accuracy, outperforming state-of-the-art. We seamlessly integrated X-Sync into Contiki-NG and showcased its performance on three different off-the-shelf IEEE 802.15.4 and BLE devices. Due to its generic design, X-Sync offers the use of cross-technology broadcasts, which allow to synchronize several heterogeneous devices simultaneously, and due to its bidirectional data exchange, it can also be used as a building block for existing synchronization schemes.

Future work includes the development of novel techniques to also compensate for the averaging delay in dynamic environments, as well as the integration of round-trip algorithms.

Acknowledgments

This work was performed within the TU Graz LEAD project "Dependable Internet of Things in Adverse Environments". This work was also supported by the Austria Wirtschaftsservice Prototypenförderung (P1905510-WTG01) and by the SCOTT project. SCOTT (http://www.scott-project.eu) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement no. 737422. This joint undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. SCOTT is also funded by the Austrian Federal Ministry of Transport, Innovation and Technology under the program "ICT of the Future" (https://iktderzukunft.at/en/). The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

References

[1] J. Elson *et al.*, "Fine-grained Network Time Synchronization Using Reference Broadcasts," *SIGOPS Oper. Syst. Rev.*, 2002.
[2] S. Ganeriwal *et al.*, "Timing-sync Protocol for Sensor Networks," in *Proc. of the* 1^{st} *SenSys Conf.*, 2003.
[3] M. Maróti *et al.*, "The Flooding Time Synchronization Protocol," in *Proc. of the* 2^{nd} *SenSys Conf.*, 2004.
[4] S. M. Lasassmeh *et al.*, "Time Synchronization in Wireless Sensor Networks: A Survey," in *Proc. of the IEEE SoutheastCon Conf.*, 2010.
[5] G. Aloi *et al.*, "A Mobile Multi-Technology Gateway to Enable IoT Interoperability," in *Proc. of the* 1^{st} *IoTDI Conf.*, 2016.
[6] T. Zachariah *et al.*, "The Internet of Things Has a Gateway Problem," in *Proc. of the* 16^{th} *HotMobile Worksh.*, 2015.
[7] K. Chebrolu *et al.*, "Esense: Communication through Energy Sensing," in *Proc. of the* 15^{th} *ACM MobiCom Conf.*, 2009.
[8] Y. Zhang *et al.*, "HoWiES: A Holistic Approach to ZigBee Assisted WiFi Energy Savings in Mobile Devices," in *Proc. of the* 32^{nd} *INFOCOM'13*.
[9] S. Yin *et al.*, "Interconnecting WiFi Devices with IEEE 802.15.4 Devices without Using a Gateway," in *Proc. of the* 15^{th} *DCOSS Conf.*, 2015.
[10] X. Zhang *et al.*, "Gap Sense: Lightweight Coordination of Heterogeneous Wireless Devices," in *Proc. of the* 32^{nd} *INFOCOM Conf.*, 2013.
[11] S. M. Kim *et al.*, "FreeBee: Cross-Technology Communication via Free Side-Channel," in *Proc. of the* 21^{st} *ACM MobiCom Conf.*, 2015.
[12] Z. Chi *et al.*, "B2W2: N-way Concurrent Communication for IoT Devices," in *Proc. of the* 14^{th} *ACM SenSys Conf.*, 2015.
[13] X. Guo *et al.*, "Wizig: Cross-technology Energy Communication over a Noisy Channel," in *Proc. of the* 36^{th} *INFOCOM Conf.*, 2017.
[14] Z. Li *et al.*, "WEBee: Physical-Layer Cross-Technology Communication via Emulation," in *Proc. of the* 23^{rd} *ACM MobiCom Conf.*, 2017.
[15] W. Jiang *et al.*, "Achieving Receiver-Side Cross-Technology Communication with Cross-Decoding," in *Proc. of the MobiCom Conf.*, 2018.
[16] ——, "BlueBee: a 10,000x Faster Cross-Technology Communication via PHY Emulation," in *Proc. of the* 15^{th} *ACM SenSys Conf.*, 2017.
[17] Y. Chen, Z. Li, and T. He, "TwinBee: Reliable Physical-Layer CTC with Symbol-Level Coding," in *Proc. of the* 37^{th} *INFOCOM Conf.*, 2018.
[18] S. Wang *et al.*, "Networking Support For Physical-Layer Cross-Technology Communication," in *Proc. of the* 26^{th} *ICNP Conf.*, 2018.
[19] Z. Tan *et al.*, "UAV-Assisted Low-Consumption Time Synchronization Utilizing Cross-Technology Communication," *Sensors*, vol. 20, 2020.
[20] Z. Yu *et al.*, "Crocs: Cross-Technology Clock Synchronization for WiFi and ZigBee," in *Proc. of the* 15^{th} *EWSN Conf.*, 2018.
[21] R. Hofmann *et al.*, "X-Burst: Enabling Multi-Platform Cross-Technology Communication between Constrained IoT Devices," in *Proc. of the* 16^{th} *IEEE SECON Conf.*, 2019.
[22] ——, "SERVOUS: Cross-Technology Neighbour Discovery and Rendezvous for Low-Power Wireless Devices," in *Proc. of the* 18^{th} *EWSN Conf.*, 2021.
[23] H. Brunner *et al.*, "Cross-Technology Broadcast Communication between Off-The-Shelf Wi-Fi, BLE, and IEEE 802.15.4 Devices," in *Proc. of the* 17^{th} *EWSN Conf., demo session*, 2020.
[24] ——, "Leveraging Cross-Technology Broadcast Communication to build Gateway-Free Smart Homes," in *Proc. of the* 17^{th} *DCOSS Conf.*, 2021.
[25] H. Kopetz *et al.*, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Transactions on Computers*, vol. C-36, 1987.
[26] M. A. Fischler *et al.*, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, 1981.
[27] K. Römer, "Time Synchronization in Ad Hoc Networks," in *Proc. of the* 2^{nd} *MobiHoc Conf.*, 2001.
[28] J. Elson *et al.*, "Wireless Sensor Networks: A New Regime for Time Synchronization," *SIGCOMM Comput. Commun. Rev.*, vol. 33, 2003.
[29] F. Ferrari *et al.*, "Efficient Network Flooding and Time Synchronization with Glossy," in *Proc. of the* 10^{th} *IPSN Conf.*, 2011.