Paper published at the 16th IEEE International Conference on Sensing, Communication and Networking (SECON) in June 2019

X-Burst: Enabling Multi-Platform Cross-Technology Communication between Constrained IoT Devices

Rainer Hofmann, Carlo Alberto Boano, and Kay Römer
Institute of Technical Informatics, Graz University of Technology, Austria
E-mail: rainer.hofmann@tugraz.at; cboano@tugraz.at; roemer@tugraz.at

***Abstract*—Cross-technology communication (CTC) allows devices employing incompatible wireless technologies to directly exchange information without the need of expensive gateways. Existing work on CTC has showcased the ability of exchanging data between diverse wireless standards, but has not analysed the challenges nor tackled the problem of enabling CTC between multiple constrained IoT platforms with different characteristics. Indeed, CTC schemes are often hacked on very specific hardware platforms, which results in a lack of a general, portable solution. Furthermore, CTC has always been tested as a standalone piece of functionality, and its seamless integration with the classical operations of a constrained IoT device remains an open challenge. In this paper, we present *X-Burst*, a portable framework that allows multiple constrained IoT platforms with diverse characteristics to seamlessly interact using CTC. X-Burst allows to customize the CTC working principle (e.g., how information is encoded, or the alphabet used to encode a symbol) and enables the combination of different encoding and decoding strategies independently of the employed hardware platform. Thanks to its high modularity, X-Burst also simplifies the development of alternative CTC implementations and makes it easy to compare different approaches. As a proof of concept, we integrate X-Burst into the Contiki operating system without changing Contiki's core functions and allow an IoT device to seamlessly support CTC in parallel to its normal operations. We then showcase the functionality of X-Burst by enabling a bidirectional CTC between off-the-shelf heterogeneous IoT platforms based on IEEE 802.15.4 and Bluetooth Low Energy (BLE). An experimental evaluation further shows X-Burst's small memory footprint and analyses the robustness and throughput of different encoding schemes.**

***Index Terms*—Bluetooth Low Energy; IEEE 802.15.4; Contiki; Cross-technology communication; Internet of Things; X-Burst.**

I. Introduction

The massive proliferation of wireless systems and the growing density of Internet of Things (IoT) applications increases the need for a *direct* interaction between heterogeneous devices and platforms. The ability to directly share information allows, for example, co-located wireless devices and networks operating in the same frequencies to develop coexistence strategies minimizing the impact of cross-technology interference [1], [2]. Furthermore, it enables a seamless data collection in smart homes and industrial IoT systems composed of diverse wireless devices from different vendors [3].

Unfortunately, although most wireless technologies operate in the same unlicensed ISM bands, they typically have incompatible physical layers (PHY), which does not allow heterogeneous devices to directly exchange packets. Because of this, one often resorts to the use of *multi-radio gateways* in order to relay data packets and synchronize the operations between wireless devices making use of different technologies [4], [5].

The use of gateways, however, introduces extra costs, increases traffic, and leads to higher end-to-end delays. Furthermore, 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].

Recently, *cross-technology communication* (CTC) is emerging as a viable alternative to the use of multi-radio gateways and is becoming an increasingly hot research topic. CTC enables a direct data exchange across heterogeneous wireless devices by making use of a side channel that is mutually available [7], which allows channel coordination [2] as well as more advanced services such as sensor (re-)configuration [8].

Early CTC works have shown how to exchange data between various wireless technologies and decode information through energy detection, e.g., based on frame length [9], beacon interval [10], gap duration [11], and energy level [12]. Recent approaches have also shown how to carry out CTC by means of physical layer emulation [13], [14], [15], [16], which allows to transmit data at a very high throughput.

Lack of multi-platform CTC solutions. Despite the large body of work that has been produced in the last years, research on CTC is still at an early stage. So far, the community has mostly focused on building prototypes showcasing the ability of carrying out CTC between diverse wireless standards and on highlighting potential applications [2], [8], [17]. Such proof-of-concepts are typically implemented using powerful software-defined radios [11], [12], [18] or hacked on specific hardware (HW) platforms [13], [19] and hardly describe any implementation detail. As the interest in CTC grows, there is a need to move away from feasibility studies in favour of general CTC solutions *supporting multiple HW platforms by design*. Furthermore, the availability of generic and portable CTC solutions as *open-source implementations* can empower novel CTC research in the years to come.

Supporting HW platforms with diverse characteristics. The creation of portable CTC solutions is, however, very complex, due to the diversity of commercially-available HW platforms. Indeed, depending on the PCB design and on the employed components, each HW platform exhibits unique characteristics, e.g., how to access the radio, the CPU clock rate, as well as the speed at which radio instructions can be loaded and executed. These aspects drastically affect the design of CTC schemes and practically force developers to tailor and heavily optimize their implementations to the devices at hand. For example, when carrying out CTC by encoding information

into the duration of a message and by using energy sensing (i.e., by measuring the radio's received signal strength) for decoding [9], one needs to make sure that the operations of transmitter and receiver are precisely timed. To this end, aspects such as how fast one can send commands to the radio, how quickly one can sample/process the received signal strength (RSS), or the time granularity of the device, play a crucial role and affect the implementation of the CTC scheme – making it hard to build a portable solution across multiple HW platforms with different characteristics.

Supporting constrained IoT devices. Building a portable CTC solution becomes even more complex when dealing with constrained IoT devices such as environmental sensors and actuators (e.g., BLE- or IEEE 802.15.4-based smart objects). These devices are typically equipped with single-core CPUs and embed only a few kB of memory. This results in severe processing, memory, and storage constraints, which limits the complexity of the encoding and decoding schemes that can be used to carry out CTC, as well as the achievable throughput. Furthermore, these devices employ primitive radios supporting only limited features and focusing on low-power operation. In these radios, even simple functionality such as RSS sampling may be implemented differently, which has important implications on the design of portable decoding schemes based on energy sensing, as we show in Sect. II.

Lack of integration into existing systems. CTC has mostly been implemented as a standalone piece of functionality, leaving aside its integration with the classical operations of a device. Integrating CTC with existing functionality is particularly complex for constrained IoT devices, due to their severe resource limitations and the strong focus on energy-efficiency. Smart objects based on BLE or IEEE 802.15.4, for example, employ duty-cycled MAC protocols to minimize the time in which the radio is turned on. A seamless integration of CTC functionality with these protocols should make sure that the transmission and reception of CTC messages does not interfere with the regular duty-cycled communications, which requires a smart scheduling of the radio access. This is, however, hard to achieve, as constrained IoT devices typically make use of stripped-down operating systems that are not designed to handle multiple network stacks concurrently [20].

Our contribution: X-Burst. In this paper, we present X-Burst, a portable framework that allows multiple constrained IoT platforms with diverse characteristics to seamlessly interact using CTC. X-Burst is based on *energy sensing* [9], i.e., it makes only use of features available on standard-compliant off-the-shelf radios to send precisely-timed *energy bursts* and to decode information by means of RSS sampling. This keeps X-Burst's design generic and allows to broadcast CTC messages to multiple heterogeneous devices simultaneously[1].

[1]In this paper, we explicitly target constrained IoT devices embedding transceivers compliant to the BLE [21] or the IEEE 802.15.4 PHY/MAC standard [22], as these are the two most widespread wireless technologies used to build smart objects. However, note that X-Burst's design is generic, and can also be used, for example, in more powerful devices embedding Wi-Fi radios, as they also support CTC by means of energy sensing [9], [10], [23].

X-Burst allows to separate the CTC working principle (e.g., how information is encoded and decoded, or the alphabet used to encode a symbol) from low-level hardware details. To this end, X-Burst is built as a set of thin HW-agnostic modules that sit on top of a hardware abstraction layer, which simplifies the development of alternative CTC implementations and makes it easy to combine different encoding and decoding strategies.

X-Burst's modular design also allows to automatically compute the *alphabet* (i.e., the set of properties of an energy burst) that should be used to encode CTC symbols. Such an alphabet is derived according to the characteristics of the devices involved in the communication (e.g., the time granularity, the time necessary to measure, process, and store an RSS value, as well as the response time of the radio to a command).

As a proof of concept, we integrate X-Burst into the popular Contiki operating system [24] and allow to carry out CTC in parallel to classical operations. In particular, we let a scheduler learn the existing duty-cycling strategy and trigger the transmission and reception of CTC messages only when the radio would otherwise be unused. We achieve this without modifying the existing network stack and without affecting Contiki's normal communication flow.

We showcase the functionality of X-Burst by enabling a bidirectional CTC between several off-the-shelf IoT platforms embedding an IEEE 802.15.4 or BLE radio, such as the TI CC2650 LaunchPad, the Zolertia Firefly, and the TelosB mote. To the best of our knowledge, we are the first to show a *bidirectional* CTC between *multiple heterogeneous off-the-shelf* BLE and IEEE 802.15.4 devices. We further illustrate how X-Burst's modularity allows to easily compare the performance of different encoding and decoding schemes. We finally analyse X-Burst's memory footprint and evaluate experimentally the throughput and robustness of different CTC configurations.

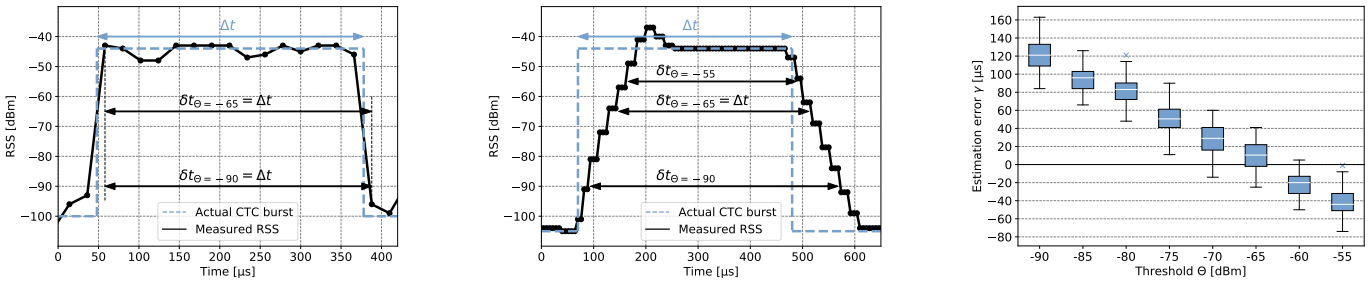
After discussing the rationale behind X-Burst's design and highlighting the challenges in devising a generic CTC framework in Sect. II, this paper makes the following contributions:

- We present X-Burst, a framework allowing diverse IoT platforms to seamlessly interact using CTC (Sect. III);
- We describe X-Burst's modular design and highlight how it keeps CTC implementations HW-agnostic and simplifies the development of new functionality (Sect. IV);
- We seamlessly integrate X-Burst into Contiki and allow to carry out CTC in parallel to classical operations (Sect. V);
- We showcase how X-Burst enables a bidirectional CTC between several diverse BLE and IEEE 802.15.4 devices, analyse its memory footprint, as well as evaluate the robustness of various encoding schemes (Sect. VI).

After describing related work in Sect. VII, we conclude our paper in Sect. VIII, along with a discussion on future work.

II. X-Burst: Design Rationale

In this section, we describe the rationale behind X-Burst's design and the challenges in devising a generic CTC framework for multiple heterogeneous and constrained IoT devices.



(a) Instantaneous RSS measurement (b) Non-instantaneous RSS measurement (c) Accuracy of the estimated durations

Fig. 1: RSS sampled with a BLE (a) and an IEEE 802.15.4 device (b). Whilst the former returns an instantaneous value, the latter returns the average over the last 128 μs, which affects the effectiveness of decoding strategies based on static thresholds (c).

Supporting a generic CTC scheme. Existing CTC approaches make use of either *physical-layer emulation* [14], [13], [15] or *packet-level modulation* [9], [10], [11], [12], [23], [25]. The former consists in adjusting the payload of a frame such that a portion of it can be recognized by a device using another technology as a legitimate frame. Such an approach allows to perform CTC at very high data rates, but it typically only supports one direction[2] and is technology-specific, i.e., one cannot transmit the same CTC message to devices employing diverse technologies simultaneously. Packet-level modulation, instead, makes use of properties such as the packet duration [9], [23], [26], the interval between packets [10], [11], and the energy-level with which they are sent [7], [12], [27] to perform CTC. This approach is more *generic* than physical-layer emulation and allows a *bidirectional* CTC between heterogeneous devices, as well as the broadcasting of a CTC message to multiple devices employing diverse technologies *simultaneously*. When following this approach, CTC messages are typically decoded by means of energy detection techniques such as RSS sampling. These are available on most off-the-shelf radios: either mandated by standards [22] or available as vendor-specific extensions [21]. To ensure generality, we base X-Burst on packet-level modulation and create a dedicated module to support the different *encoding* approaches proposed in the literature. As we show in Sect. IV, this module also simplifies their combination and the creation of new schemes.

Tackling the heterogeneity of wireless technologies. Even though the use of energy detection techniques such as RSS sampling has been widely used to decode CTC messages, a few challenges arise when creating a generic decoding functionality that should work across multiple technologies. One, indeed, needs to cope with the fact that different technologies may specify diverse ways to compute the RSS value. The IEEE 802.15.4 standard, for example, specifies that the RSS is calculated as the average signal strength of eight symbol periods, i.e., the average over the last 128 μs. Other technologies, instead, return the current signal strength at the antenna pins *instantaneously*. This may lead to different types of RSS measurements depending on the employed HW platform, which affects the effectiveness of the decoding strategies based on static thresholds [9], as shown in Fig. 1. For example, approaches making use of a single RSS threshold Θ to estimate the duration of an energy *burst* (i.e., the duration δt in which the activity in the channel is stronger than Θ), may be suitable when using a BLE device returning an instantaneous RSS measurement (Fig. 1a), but not when using an IEEE 802.15.4 device returning an averaged RSS value (Fig. 1b). In the latter case, the choice of Θ strongly affects the accuracy of the estimated δt. A wrong selection of Θ (e.g., -55 or -90 dBm) would result in large estimation errors $\gamma = \delta t - \Delta t$ (with Δt being the actual duration of the energy burst), as shown in Fig. 1c. Unfortunately, it is hard to find a suitable threshold that minimizes γ in advance, as the strength of a signal is affected, among others, by the distance between the devices, by the employed transmission power, as well as by the characteristics of the surrounding environment. To cope with these issues, we create in X-Burst a dedicated *decoding* module, which allows to separate the technique used to decode CTC messages from other functionality. As we show in Sect. IV, this module allows to easily compare the performance of different decoding strategies and find the one that works best across multiple technologies.

Tackling the heterogeneity of HW platforms. CTC schemes based on packet-level modulation typically map a data symbol into an energy burst of pre-defined duration [23], power [11], or into the length of the gap between consecutive bursts [27]. The supported set of properties used to map a symbol (i.e., the different burst durations, power levels, or gap lengths) is referred to as *alphabet*. The size of an alphabet and the supported set of values are strongly dependent on the characteristics of the employed HW. For example, when encoding information in the duration of an energy burst, the set of supported durations should be chosen such that *all* the platforms involved in the CTC communication have the ability to discern two distinct durations by means of RSS sampling. This implies that the minimum difference between two durations supported in the alphabet should be proportional to the timing characteristics of each device, i.e., to aspects such as the time granularity, the time necessary to measure, process, and store an RSS value, as well as the response time of the radio when issuing a TX command. Fig. 2 illustrates how those timing aspects can largely vary across different HW platforms. Whilst the difference between older platforms (e.g., the TelosB mote)

[2]XBee [15] has shown that physical-layer emulation is also possible from a low-end to a high-end device by means of cross-decoding, but did not show how to achieve a bidirectional communication using commodity devices.

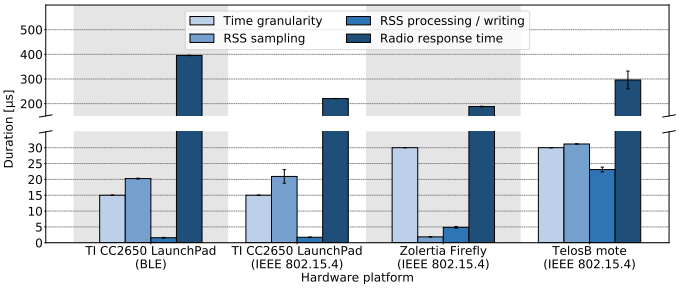


Fig. 2: Different hardware platforms can have largely diverse timing characteristics, which affects the creation of an alphabet that is compatible with multiple heterogeneous devices.

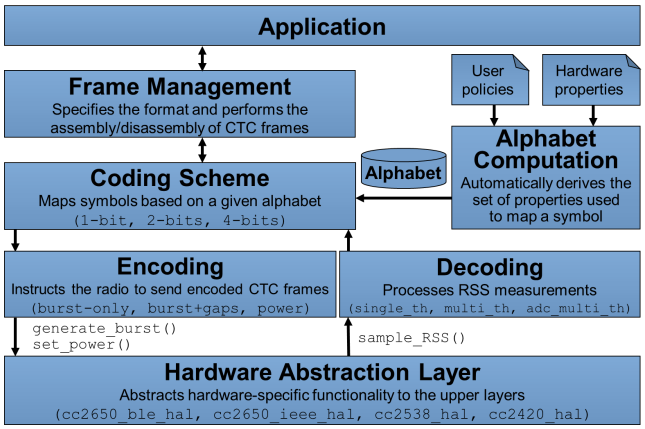


Fig. 3: X-Burst's modular architecture.

and newer platforms based on ARM Cortex-M3 CPUs (e.g., TI LaunchPad and Zolertia Firefly) is quite evident, one can still note how there is a non-negligible variability also across the timing characteristics of these newer platforms, and that the response time of the BLE radio is very long. Accounting for these variations when building an alphabet is important in order to achieve a high CTC throughput, as this requires that the operations of transmitter and receiver are precisely timed. However, the construction of such an alphabet is ideally done *automatically*, so to abstract the low-level HW details to the developer. To this end, X-Burst embeds a dedicated alphabet module that aids the automated creation of a CTC alphabet that is supported by all communicating devices.

Keeping main CTC functionality HW-agnostic. A key goal of our work is the creation of a portable framework that abstracts low-level HW-specific details from the development of the main CTC functionality. This requires the creation of a *hardware abstraction layer* that offers basic primitives such as RSS sampling or the transmission of energy bursts of predefined duration to the upper layers taking care of the actual CTC logic. This is important also in light of the heterogeneity of HW platforms: some radios only support the transmission of standard-compliant packets using payloads of different length, whilst some others also support test modes that allow the fine-grained creation of precisely-timed modulated carriers [28]. Similarly, some HW platforms return the absolute RSS in dBm, whilst some others return the number of times in which the RSS exceeded a given threshold [19]. In X-Burst all HW-dependent functionality is hence separated from the encoding/decoding strategies and from other CTC configurations, which enhances the portability to new devices.

Seamless integration with existing functionality. A device typically uses CTC to carry out additional activities on top of its *normal operations*, e.g., it exploits CTC to coordinate the frequency usage with surrounding wireless appliances [2]. Making sure that CTC works in parallel with existing functionality can be, however, quite challenging, especially in constrained IoT devices. Operating systems (OS) tailored for this class of devices typically employ optimized network stacks making use of duty-cycling strategies to periodically switch off the radio and minimize energy consumption. CTC functionality should hence only make use of the radio while the latter is not used by traditional communications. This should ideally be achieved *without* requiring modifications to the existing network stack or duty-cycling strategy, as this would incur a significant overhead and cause compatibility problems. Furthermore, the transmission of CTC messages and their reception using RSS sampling should not block the CPU for an extended amount of time, as this may affect normal operations such as periodic sensor measurements and their processing. We hence integrate X-Burst into Contiki by creating a *virtual radio* that learns the duty-cycle of a device by analysing the calls to the `radio_on()` and `radio_off()` functions. The virtual radio then schedules all CTC-related operations such that existing functionality is not affected.

III. X-Burst: Architecture

We design X-Burst to be highly-modular, in order to reduce the complexity of CTC implementations, maximize code reuse, and simplify the development of new functionality. Fig. 3 sketches X-Burst's overall architecture, which follows the design rationale illustrated in Sect. II.

A *hardware abstraction layer* (HAL) provides the minimal functions needed to transmit energy bursts with a given duration or transmission power, as well as the ability to sample the received signal strength. The HAL module is the only one containing hardware-dependent functionality and helps maintaining the main CTC functionality HW-agnostic.

An *encoding module* exploits the primitives offered by the HAL to instruct the radio transceiver to transmit energy bursts with a given length or power. This module is unaware of how an energy burst is actually sent, i.e., whether using frames or test modes. Similarly, a *decoding module* processes the RSS samples obtained from the radio and detects the duration or the power level of an energy burst as accurately as possible.

A *coding scheme* module is used to define the number of bits of information (n) that are encoded into a CTC symbol. Each CTC symbol is then mapped to a set of properties (e.g., burst durations, power levels, or lengths of the gap between bursts) depending on a given *alphabet*, whose size needs to be at least 2^n. For example, when using a symbol length of 1 bit ($n = 1$) and when encoding information into the duration of an energy burst, the alphabet is composed

of two burst durations identifying a "0" and a "1", respectively. To allow CTC broadcasting, i.e., the transmission of a CTC message to devices employing diverse technologies simultaneously, a generic alphabet that is supported by all communicating devices needs to be found. To this end, an *alphabet computation* module automatically derives the alphabet's set of values based on the *hardware properties* of the communicating devices (e.g., the time granularity, the time necessary to measure, process, and store an RSS value, as well as the radio response time). The alphabet computation module hence makes sure that all devices involved in the communication have sufficient features to encode or decode information. The computation of the alphabet can also be influenced by *user policies* defining whether to prioritize throughput or reliability. For example, a more aggressive policy can minimize the difference between two durations in the alphabet, which would shorten the transmission time and maximize throughput, at the price of a higher likelihood of decoding errors.

Finally, a *frame management* module takes care of assembling and disassembling CTC messages based on a given frame structure. This gives the flexibility to define what should be prepended or appended to a CTC payload, e.g., a synchronization preamble, a header with the address of the intended recipient, or a footer embedding a checksum.

IV. Design and Implementation Details

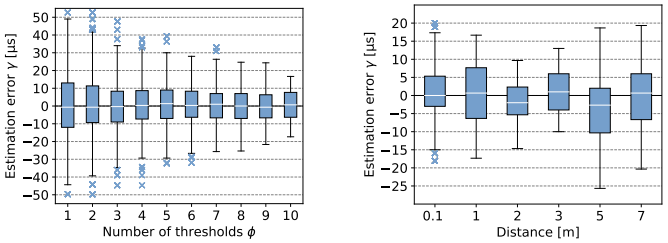
We describe next X-Burst's core modules in detail. We start by illustrating the operations of an encoding (Sect. IV-A) and decoding (Sect. IV-B) module. We then show an algorithm to automatically derive the set of values of an alphabet supported across multiple HW platforms (Sect. IV-C) and detail on the operations of the frame management module (Sect. IV-D).

A. Encoding

As discussed in Sect. II, we base X-Burst on packet-level modulation and create an encoding module to support the various approaches proposed in the literature (e.g., schemes encoding information in the duration of or gap between frames, as well as in the energy level with which they are sent).

Our basic implementation uses a `burst-only` encoding scheme where the duration of an energy burst is used to convey information. We further enrich our X-Burst implementation with a `burst+gaps` scheme that exploits also the length of the gap between two consecutive bursts. As we show in Sect. VI, the latter allows to improve the throughput by shortening the time on-air of a CTC frame.

We implement both schemes by making use of the `generate_burst()` function available from the HAL to transmit an energy burst. Off-the-shelf IEEE 802.15.4 devices can generate a burst by sending a frame and by adjusting the length in bytes of its payload. Accounting for the mandatory synchronization and PHY header (6 bytes), the minimum duration of an energy burst is 192 μs at a data rate of 250 kbit/s. Increasing the payload by one byte allows to generate longer durations in steps of 32 μs up to a maximum of 4256 μs (127 bytes). Advanced radios also allow the creation of more precisely-timed modulated IEEE 802.15.4 test carriers [28].



(a) Number of thresholds (b) Distance between devices

Fig. 4: Accuracy of a decoding scheme based on multiple threshold and adaptive duration correction (`adc_multi_th`). The estimation error γ is significantly reduced compared to classic decoding schemes making use of a single threshold (see Fig. 1c), and is not affected by the distance between nodes.

Off-the-shelf BLE devices compliant to the BLE standard v4.2 and above are required to implement a direct test mode for RF PHY conformance testing [21]. This mode allows to send test packets with payload up to 255 bytes in any of the 39 BLE channels and without the need of an active connection. One can hence generate a burst with off-the-shelf BLE transceivers by adjusting the length of the payload of a test packet. Accounting for the mandatory preamble, synchronization word, header, and checksum (10 bytes), the minimum duration of an energy burst is 80 μs at a data rate of 1 Mbit/s. Increasing the payload by one byte allows to generate longer durations in steps of 8 μs up to 2120 μs (255 bytes).

B. Decoding

The decoding module uses the `sample_RSS()` function available from the HAL to detect the energy of a specified channel and analyse the stream of measured RSS values. The processing of these values can take place *online* or *offline*, i.e., one can store the stream of RSS values for later analysis, or process them on-the-fly at the cost of a lower sampling rate.

We implement different decoding strategies, starting from a scheme making use of a single static threshold (`single_th`) as in [9]. However, as highlighted in Fig. 1, the use of a single threshold is not suitable for devices returning a non-instantaneous RSS measurement and is hence not generic. We therefore design a decoding scheme (named `multi_th`) based on a configurable number of thresholds ϕ. This scheme outperforms the one based on a single threshold, but its accuracy is still affected by a systematic error introduced by the mismatch between the thresholds (selected beforehand) and the strength of the signal with which a CTC frame is received (which varies at runtime). To compensate for this systematic error, we augment the multiple threshold scheme with an *adaptive duration correction* (`adc_multi_th`). The latter exploits the known duration of bursts in the preamble of a CTC frame to estimate the systematic error and correct the subsequent RSS readings. Fig. 4 shows the performance of this scheme on the same device performing non-instantaneous readings used to illustrate Fig. 1b and 1c (i.e., a TI LaunchPad 2650 used in IEEE 802.15.4 mode). The horizontal white lines represent the median and the blue boxes represent the

25th and 75th percentiles; the vertical whiskers show the remaining measurements, whilst the blue crosses represent statistical outliers. Fig. 4a shows that, regardless of the number of thresholds, the estimated burst duration error γ does not present a systematic offset (as opposed to Fig. 1c). Fig. 4b shows that, when using ϕ=10, γ is consistently below $\pm 25\,\mu$s regardless of the distance between nodes, which hints that this decoding scheme can be used to overcome the heterogeneity of different HW platforms in measuring the RSS.

C. Coding Scheme & Alphabet Computation

X-Burst's coding scheme module currently supports the encoding of $n = \{1, 2, 4\}$ bits of information into a CTC symbol. Each symbol is then mapped to a set of properties (e.g., burst durations) depending on a given alphabet. To automatically compute the latter such that it is supported by several devices with heterogeneous characteristics, we implement an alphabet computation module based on the following algorithm.

1) Given the properties of each device involved in the communication, determine the minimum burst duration that can be generated by each device and select the largest one (d_0). The latter is commonly the minimum burst duration generated by the device employing the technology with the slowest data rate.
2) For each device involved in the communication, calculate the minimum spacing between two consecutive burst durations as $s = 2 \cdot (t_{granularity} + t_{rss})$, where $t_{granularity}$ is the resolution of a clock tick and t_{rss} is the time necessary to sample and process an RSS value. Hence, s embodies the Nyquist rate, i.e., twice the maximum component frequency of the sampled RSS.
3) Select the largest s across all devices involved in the communication (s_{max}) and use it to calculate the set of durations $d_i = \lceil d_{i-1} + s_{max} \rceil$, with $i = 1, ..., 2^n$ and where the ceiling operator rounds d_i to the next duration that can be supported by each radio.

In the current version of X-Burst, the properties of each device involved in the communication are known beforehand and provided at compile time. In principle, however, one can also let devices advertise their HW properties at runtime using a baseline alphabet that works across all technologies, and then let a set of devices use this info to negotiate a more performant alphabet. The creation of such runtime scheme is beyond the scope of this paper and will be addressed in future work.

D. Assembling and Disassembling CTC Frames

X-Burst's frame management module takes care of assembling and disassembling CTC messages based on a given frame structure. In our current implementation, a CTC frame consists in a *preamble* of 5 bursts, as well as a 1-byte *options* bit-mask.

The latter specifies whether a *header* pre-pending the actual *payload* contains the length of the CTC frame in bytes, the network ID, the address of transmitter and receiver, as well as whether an acknowledgement is requested and a checksum is appended at the end. As described in Sect. V, this module is used as an interface with the OS, which provides the CTC data to be transmitted and receives the decoded CTC messages.

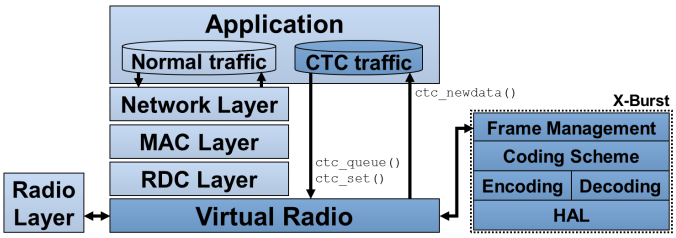


Fig. 5: Integration of X-Burst into Contiki's network stack.

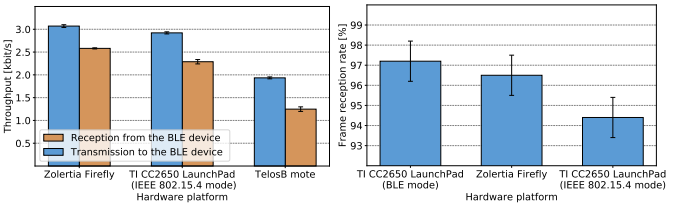
V. Integration into Contiki

As a proof of concept, we integrate X-Burst into the Contiki operating system [24]. We make sure that the integration is seamless, i.e., that no changes to Contiki's core functions and network stack are necessary. This allows developers to extend existing applications with CTC capabilities without the need to re-implement functionality and without leading to compatibility problems. We also ensure that traditional duty-cycled communication is not affected by CTC functionality, thanks to a smart scheduling of the radio access.

Virtual radio. To ensure a seamless integration, we employ the architecture sketched in Fig. 5. We make use of a *virtual radio* that is informed about (or learns) the duty-cycle of a device and that schedules all CTC-related operations such that traditional communications are not affected. As shown in Fig. 5, Contiki's traditional network stack is unchanged (light blue color), as the virtual radio is simply loaded instead of the actual radio driver implementation. In the case of usual communications, the virtual radio only forwards the data between the radio layer and the upper layers. Thus, the application transmits and receives normal traffic as usual. The virtual radio allows the application to queue outgoing CTC traffic and set the CTC configuration at runtime using existing functions of Contiki's network stack, as well as to be notified of incoming CTC traffic by posting `ctc_newdata()` events. All X-Burst's modules (except the HAL) make only use of Contiki's standard functions and `rtimers`: this ensures portability across all platforms supported by Contiki.

Adapting to existing duty-cycling strategies. Our goal is to avoid that developers making use of X-Burst need to modify Contiki's radio duty-cycling (RDC) layer or have detailed knowledge about its implementation. We hence let the virtual radio either exploit pointers to Contiki's configuration files specifying the duty-cycle period, or autonomously learn it by timestamping the calls to Contiki's `radio_on()` functions and by analysing their distribution. Whenever the application fills the CTC queue with new information to be transmitted, the virtual radio waits for a `radio_off()` event and transmits the CTC packet a configurable number of times until the end of the current duty-cycle period. The virtual radio can also allocate a configurable portion of the remaining duration of a duty-cycle period after a `radio_off()` event to let a device perform RSS sampling and look for incoming CTC frames.

We further define several flags indicating whether CTC can be granted a higher priority than normal communication and whether RSS sampling should take place only every n^{th} duty-



(a) Bidirectional communication (b) Broadcast communication

Fig. 6: X-Burst allows heterogeneous devices to make use of the same solution and achieve a bidirectional CTC (a), as well as to broadcast the same CTC frame to multiple recipients (b).

cycle period, so to minimize energy-expenditure. These flags can be used as a starting point for the implementation of more advanced functionality that is beyond the scope of this paper, e.g., strobing mechanisms ensuring that a recipient is ready to receive a CTC frame, as well as the synchronization of the cyclic operations of heterogeneous devices.

VI. Evaluation

We evaluate X-Burst experimentally. We first showcase its functionality by enabling a bidirectional CTC between multiple heterogeneous BLE and IEEE 802.15.4 devices (Sect. VI-A). We then analyse the robustness and throughput of different encoding schemes as a function of different payload and symbol lengths in the absence and presence of radio interference (Sect. VI-B). We finally analyse X-Burst's memory footprint in terms of RAM and ROM usage (Sect. VI-C).

Experimental setup. We make use of the three most popular off-the-shelf IoT platforms supported by Contiki: the TI CC2650 LaunchPad, the Zolertia Firefly (employing a TI CC2538 transceiver), and the TelosB mote (embedding a TI CC2420 radio). We use two versions of the TI CC2650 LaunchPad: one configured in IEEE 802.15.4 mode, and one configured in BLE mode. Unless specified differently, we carry out all experiments in a vacant office with the devices placed at 1 meter distance and using a transmission power of 0 dBm.

A. Enabling multi-platform bidirectional CTC

We first showcase how X-Burst allows devices with diverse HW characteristics and making use of heterogeneous technologies to achieve a bidirectional CTC. We share all X-Burst's layers (except the HAL) across the different HW platforms and configure an experiment in which a TI CC2650 LaunchPad in BLE mode sends and receives data to/from each of the other IEEE 802.15.4 devices individually, using the alphabet enabling the highest data rate for bidirectional communication. We employ a `burst-only` encoding and let each device transmit back-to-back CTC frames with a payload of 16 bytes.

Fig. 6a shows the throughput of each platform when transmitting and receiving data to/from the BLE device. Two aspects are worth of note. First, the throughput when receiving data from a BLE device is slower than when transmitting to it: this is due to the BLE transmitter's higher radio response time, as highlighted in Fig. 2. Second, the throughput of the Zolertia Firefly is higher than that of the TI CC2650 LaunchPad in IEEE 802.15.4 mode, which in turn is higher than that of the TelosB mote. This is due to the diversity in the platforms' RSS sampling and processing speed: lower sampling and processing rates result in a lower throughput.

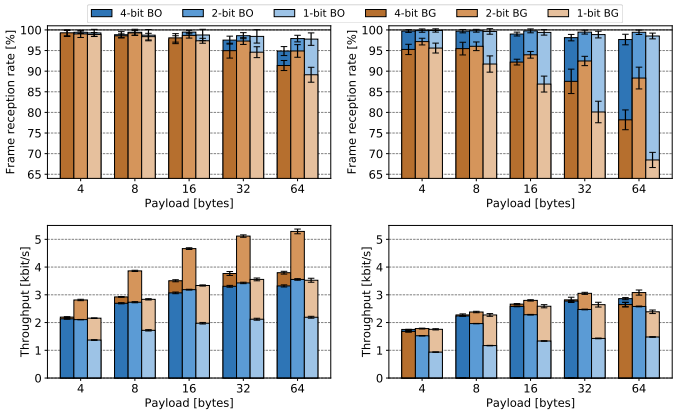
We next let a TelosB mote broadcast CTC data with an automatically-computed alphabet that is supported by all other IEEE 802.15.4 and BLE devices. Fig. 6b shows that all devices can correctly receive more than 94% of the individual CTC frames. Relative differences between the platforms are due to the different RSS sampling and processing rates. Furthermore, the instantaneous RSS measurements of the TI CC2650 LaunchPad in BLE mode allows for a more reliable decoding.

B. Performance of different CTC configurations

We evaluate next the performance of different encoding strategies and coding schemes in the absence and presence of radio interference. We compare the reliability (i.e., the number of individual CTC frames received correctly) and the throughput (effective data rate when sending CTC frames back-to-back) of each direction. We make use of coding schemes with 1, 2, or 4 bits length, as well of `burst-only` and `burst+gaps` encoding schemes. For each configuration, we send 3000 CTC frames with different payload lengths. Due to space constraints, we show only the results relative to the communication between an IEEE 802.15.4-based Zolertia Firefly and a TI CC2650 LaunchPad in BLE mode, but the relative trends apply also to other platforms.

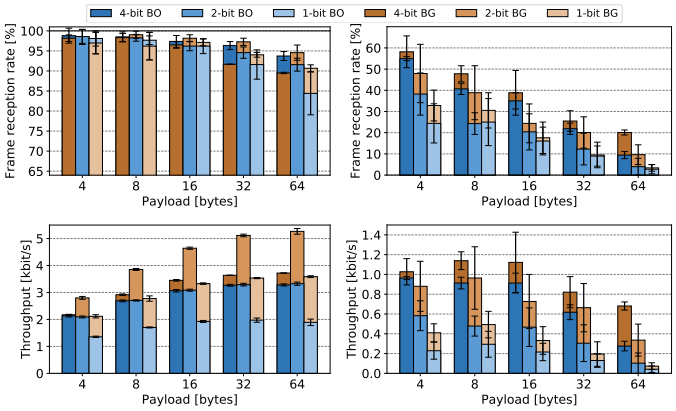
Absence of radio interference. Fig. 7 shows the performance of different CTC configurations in absence of radio interference for both directions. Increasing the payload size, as expected, reduces the chances to correctly receive a frame. The throughput, nevertheless, increases when making use of a higher payload size, due to the lower number of packets sent and hence lower message overhead (caused by the transmission of a CTC preamble, header and checksum). For the same reason, making use of `burst-only` (BO) encoding results in a lower throughput, compared to the use of a `burst+gaps` (BG) encoding: one can indeed embed more information in the latter and minimize the number of frames sent. Fig. 7b shows that `burst-only` is significantly more reliable than `burst+gaps` encoding in the BLE $\rightarrow$ IEEE 802.15.4 direction. This is due to the inaccuracy of Contiki's `wait()` implementation in the TI CC2650 LaunchPad, which is used to generate the gaps between bursts.

The use of a `2-bit` coding scheme consistently offers the highest reliability, with the `4-bit` scheme outperforming the `1-bit` one. The reason for this can be found in the trade-off between the symbol length and the longest duration used for the transmission. On the one hand, having a symbol length of 4 requires an alphabet of at least 16 durations, some of which are very long. On the other hand, a `4-bit` scheme only requires two bursts to convey 1 byte of information. However, for the BLE $\rightarrow$ IEEE 802.15.4 direction and the `burst-only` configuration, the `4-bit` scheme offers the higher throughput, due to the higher response time of the BLE radio.



(a) IEEE 802.15.4 → BLE (b) BLE → IEEE 802.15.4

Fig. 7: Reliability and throughput of different CTC configurations in absence of radio interference.



(a) Bluetooth interference (b) Wi-Fi interference

Fig. 8: Reliability and throughput of different CTC configurations in the presence of Bluetooth and Wi-Fi interference.

Presence of radio interference. We introduce radio interference in the surrounding of the communicating CTC nodes as follows. We generate Bluetooth interference by letting two Raspberry Pi3 create a point-to-point Bluetooth connection and transmit RFCOMM packets with a length of 1000 bytes every 11.034 ms, which results in a data rate of 725 kbit/s. We generate Wi-Fi interference by letting one Raspberry Pi3 placed at about 2 meters from the communicating nodes make use of JamLab-NG [29] emulating a user streaming videos with a transmission power of 30 mW. Fig. 8 shows the performance of different CTC configurations: due to space constraints, we show only the results of the IEEE 802.15.4 → BLE direction, but the relative trends apply also to other direction. One can note that the performance of CTC, as expected, decreases significantly in the presence of the heavier Wi-Fi interference. One can also note that the use of a `4-bit` coding scheme offers better performance under interference. The reason for this is that the `4-bit` encoding uses the fewest amount of durations to convey a CTC frame, as 1 byte can be represented by two durations: this minimizes the probability of decoding errors due to interference. Finally, we observe that a `burst+gaps` encoding is more robust under interference, due to the shorter transmission time of a frame.

C. Memory Footprint

We conclude our evaluation by quantifying the memory footprint of X-Burst in terms of RAM and ROM usage: to this end, we make use of the `gcc-size` command. Table I shows X-Burst's footprint for different HW platforms when supporting a CTC symbol length of 4-bit and an encoding scheme based on the burst durations. With a footprint of only 1 kB of RAM and 7 kB of ROM, X-Burst is lightweight and well-suited for resource-constrained IoT devices. The tiny differences between platforms are due to the diverse HAL implementations, as the HAL is the only HW-specific module.

Hardware platform	ROM [kB]	RAM [kB]
TI CC2650 LaunchPad (BLE)	7.36	1.10
TI CC2650 LaunchPad (IEEE 802.15.4)	7.86	1.17
Zolertia Firefly	6.97	1.26
TelosB mote	6.78	0.75

TABLE I: X-Burst's memory footprint for different platforms.

VII. Related Work

CTC has become an increasingly popular research topic, as it promises to enable a seamless coexistence across heterogeneous wireless technologies operating in crowded ISM bands.

Early CTC research has explored the feasibility of conveying information between devices making use of heterogeneous technologies and highlighted potential applications [9], [10], [11], [12], [30], [31], [32]. These works make use of packet-level information such as duration [9], [23], [26], [31], interval [10], [11] and power [7], [12], [27] to encode information.

Later work in 2017 and 2018 has focused on increasing the throughput of CTC by means of novel techniques based on physical-layer emulation [13], [14], [33]. For example, BlueBee [13] allows an unidirectional BLE → IEEE 802.15.4 communication at high data rate by exploring signal phase shifts. WEBee [14], instead, enables an unidirectional Wi-Fi → IEEE 802.15.4 communication at high data rate by letting a Wi-Fi transmitter adjust its frames' payload so to emulate an IEEE 802.15.4 standard-compliant packet.

As most of the CTC works until 2017 have showcased only unidirectional CTC communications (with FreeBee [10] being a notable exception, as it allows a bidirectional Wi-Fi ↔ IEEE 802.15.4 communication at a few bps), recent CTC research has also started to focus on bi-directionality. For example, Jiang et al. [18] and Wang et al. [16] have allowed a bidirectional Wi-Fi ↔ IEEE 802.15.4 communication at high data rates, whereas XBee [15] provided a first step towards a fast bidirectional BLE ↔ IEEE 802.15.4 communication (enabling receiver-side CTC by means of cross-decoding).

Existing CTC works have hence showcased the ability to allow communication between *heterogeneous technologies* in a unidirectional or bidirectional way. However, to the best of our knowledge, no work has yet tackled the problem of allowing a seamless bidirectional communication between three or more *heterogeneous devices* (i.e., devices with diverse HW characteristics) using *heterogeneous technologies* with the same

solution. X-Burst fills this gap and provides a generic portable framework that allows multiple constrained IoT platforms with diverse characteristics to seamlessly interact using CTC.

A large portion of existing CTC schemes have been implemented on platforms with plentiful resources (e.g., laptops [2], [7], [10], [19], and mobile phones [8], [13]), or on software-defined radios allowing full access to the transceiver [11], [12], [17]. Only a few works have specifically targeted (also) off-the-shelf constrained devices, but have not integrated CTC functionality in existing operating systems in a generic and portable way. X-Burst, instead, is integrated into the popular Contiki operating system, and offers a generic HW-agnostic CTC functionality for constrained IoT devices.

A few works have focused on building interference-resilient CTC schemes by means of robust coding schemes [7] and preamble-based decoding [34], but comparisons across different systems is hard due to the lack of open-source implementations. As we have shown in Sect. VI, X-Burst can be used as a tool to compare the performance of various encoding and decoding schemes on the same platforms, which empowers a more rigorous analysis of existing CTC schemes.

VIII. Conclusions and Future Work

In this paper we have presented X-Burst, a portable framework that allows multiple constrained IoT platforms with diverse characteristics to seamlessly interact using CTC. Thanks to its modular design and its seamless integration into Contiki, X-Burst simplifies the development of CTC implementations and allows to extend IoT applications with CTC capabilities without affecting existing functionality. We have showcased X-Burst's functionality by enabling a bidirectional CTC between three off-the-shelf heterogeneous IoT platforms based on IEEE 802.15.4 and BLE, analysed their memory footprint, and compared the performance of various CTC configurations.

Future work includes the port of X-Burst to additional operating systems and HW platforms, as well as its use to build advanced coexistence and synchronization mechanisms across heterogeneous devices operating in the 2.4 GHz band.

Acknowledgments

This work has been performed within the LEAD project "Dependable Internet of Things in Adverse Environments" funded by Graz University of Technology. This work was also partially funded 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 (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information at https://iktderzukunft.at/en/.

References

[1] S. Wang *et al.*, "Achieving Spectrum Efficient Communication under Cross-Technology Interference," in *Proc. of the ICCCN Conf.*, 2017.
[2] Z. Yin *et al.*, "Explicit Channel Coordination via Cross-technology Communication," in *Proc. of the 16th ACM MobiSys Conf.*, 2018.
[3] X. Guo *et al.*, "ZigFi: Harnessing Channel State Information for Cross-Technology Communication," in *Proc. of the INFOCOM Conf.*, 2018.
[4] G. Aloi *et al.*, "A Mobile Multi-Technology Gateway to Enable IoT Interoperability," in *Proc. of the 1st IoTDI Conf.*, 2016.
[5] T. Zachariah *et al.*, "The Internet of Things Has a Gateway Problem," in *Proc. of the 16th HotMobile Worksh.*, 2015.
[6] S. M. Kim *et al.*, "Free Side-Channel Cross-Technology Communication in Wireless Networks," *IEEE/ACM Trans. on Netw.*, vol. 25, no. 5, 2017.
[7] X. Zheng *et al.*, "StripComm: Interference-Resilient Cross-Technology Communication in Coexisting Environments," in *Proc. of INFOCOM'18*.
[8] I. Rüb *et al.*, "Ad Hoc 802.11-802.15.4 Crosstalk-Based Communication in Practice," in *Proc. of the 3rd MadCom Worksh.*, 2018.
[9] K. Chebrolu and A. Dhekne, "Esense: Communication through Energy Sensing," in *Proc. of the 15th MobiCom Conf.*, 2009.
[10] S. M. Kim and T. He, "FreeBee: Cross-Technology Communication via Free Side-Channel," in *Proc. of the 21st MobiCom Conf.*, 2015.
[11] X. Zhang *et al.*, "Gap Sense: Lightweight Coordination of Heterogeneous Wireless Devices," in *Proc. of the INFOCOM Conf.*, 2013.
[12] Z. Chi *et al.*, "B2W2: N-way Concurrent Communication for IoT Devices," in *Proc. of the 14th ACM SenSys Conf.*, 2015.
[13] W. Jiang *et al.*, "BlueBee: a 10,000x Faster Cross-Technology Communication via PHY Emulation," in *Proc. of the 15th SenSys Conf.*, 2017.
[14] Z. Li and T. He, "WEBee: Physical-Layer Cross-Technology Communication via Emulation," in *Proc. of the 23rd 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] S. Wang *et al.*, "Networking Support For Physical-Layer Cross-Technology Communication," in *Proc. of the 26th ICNP Conf.*, 2018.
[17] Z. Yu *et al.*, "Crocs: Cross-Technology Clock Synchronization for WiFi and ZigBee," in *Proc. of the 15th EWSN Conf.*, 2018.
[18] W. Jiang *et al.*, "Transparent Cross-Technology Communication over Data Traffic," in *Proc. of the 36th INFOCOM Conf.*, 2017.
[19] A. Bereza *et al.*, "Cross-Technology Communication between BLE and Wi-Fi using Commodity Hardware," in *Proc. of the EWSN Conf.*, 2017.
[20] M. Lenders *et al.*, "Connecting the World of Embedded Mobiles: The RIOT Approach to Ubiquitous Networking for the Internet of Things," CoRR, Tech. Rep. arXiv:1801.02833, 2018.
[21] *Specification of the Bluetooth System - v4.1*, https://www.bluetooth.org/en-us/specification/adopted-specifications, SIG Bluetooth, 2013.
[22] *IEEE Standard for Local and Metropolitan Area Networks - Part 15.4*, IEEE 802.15.4 Working Group, 2016.
[23] D. Croce *et al.*, "An Inter-Technology Communication Scheme for WiFi/ZigBee Coexisting Networks," in *Proc. of the MadCom Worksh.*, 2017.
[24] A. Dunkels *et al.*, "Contiki - a Lightweight and Flexible OS for Tiny Networked Sensors," in *Proc. of the 1st EmNetS Worksh.*, 2004.
[25] Z. Yin *et al.*, "C-Morse: Cross-technology Communication with Transparent Morse Coding," in *Proc. of the 36th INFOCOM Conf.*, 2017.
[26] Y. Zhang *et al.*, "HoWiES: A Holistic Approach to ZigBee Assisted WiFi Energy Savings in Mobile Devices," in *Proc. of INFOCOM*, 2013.
[27] X. Guo *et al.*, "Wizig: Cross-technology Energy Communication over a Noisy Channel," in *Proc. of the 36th INFOCOM Conf.*, 2017.
[28] C. A. Boano *et al.*, "JAG: Reliable and Predictable Wireless Agreement under External Radio Interference," in *Proc. of the RTSS Symp.*, 2012.
[29] M. Schuß *et al.*, "JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference," in *Proc. of the* 16^{th} *EWSN Conf.*, 2019.
[30] D. Croce *et al.*, "Demo: Unconventional WiFi-ZigBee Communications without Gateways," in *Proc. of the 9th WiNTECH Worksh.*, 2014.
[31] S. Yin *et al.*, "Interconnecting WiFi Devices with IEEE 802.15.4 Devices without Using a Gateway," in *Proc. of the 15th DCOSS Conf.*, 2015.
[32] X. Zhang *et al.*, "Cooperative Carrier Signaling: Harmonizing Coexisting WPAN and WLAN Devices," *IEEE/ACM Trans. Netw.*, vol. 21, 2013.
[33] Y. Chen, Z. Li, and T. He, "TwinBee: Reliable Physical-Layer CTC with Symbol-Level Coding," in *Proc. of the 37th INFOCOM Conf.*, 2018.
[34] S. Wang *et al.*, "Symbol-level Cross-technology Communication via Payload Encoding," in *Proc. of the 38th IEEE ICDCS Conf.*, 2018.